

Table of Contents

GcPdf Overview	4
Key Features	5
Getting Started	6-8
Quick Start	8-10
License Information	10-11
Upgrade to Latest Version	11-12
Technical Support	12
Contacting Sales	12
Redistribution	12
End-User License Agreement	12
Product Architecture	13-19
Features	20-21
Attachment	21-23
Annotations	23-25
Annotation Types	25-44
Appearance Streams	44-46
Document	46-49
Font	49-54
Forms	54-56
Import and Export Forms Data	56-59
Form XObjects	59-60
Graphics	60-64
Blend Modes	64-67
Output Intents	67-68
Images	68-76
Incremental Update	76-80
Linearization	80-81
Links	81-83
Outline	83-85
Pages	85-89
Security	89-91
Digital Signature	91-99
Soft Mask	99-101
Stamps	101-103

Tagged PDF	103-105
Parse PDF Documents	105-113
Layers	113-117
Text	117-124
Text Search	124-126
Watermark	126-127
Print	127-129
Render HTML to PDF	130-138
Save PDF as Image	139-143
Barcodes in PDF	144-149
GrapeCity Documents PDF Viewer	150-151
Licensing and Redistribution	151-153
View PDF	153
Configure PDF Viewer	153-158
Features	158
Toolbar and Panel Icons	158-160
Custom Context Menu	160-162
Advanced Search Options	162-164
View PDF Elements	164-166
Edit PDF	166-167
Configure PDF Editor	167-177
Editors	177
Annotation Editor	177-185
Redact Annotation	185-187
Stamp Annotation	187-194
Link Annotation	194-200
Form Editor	201-206
Checkbox and Radio button Appearance	206-209
Sticky Buttons	209-211
Features	211
Align	211-213
Copy and Paste	213-215
Convert to Content	215-216
Default Settings	216-222
Comments Tool	222-227
Graphical Signature Tool	227-232

Share and Collaborate	232-239
UI Customizations	239-259
Form Filler	259-277
Fill Custom Form Input Types	277-283
Client API Reference	283
Samples	284-288
Walkthrough	289
Convert HTML to PDF Report	289-295
API Reference	296
Release Notes	297
Breaking Changes	297
GcPdf Release Notes	297
Version 5.1.0.790	297-298
Version 5.0.0.762	298
Version 4.2.0.719	298-299
Version 4.2.0.715	299-300
Version 4.1.0.658	300-301
Version 4.0.0.616	301
Version 3.2.0.548	301-302
Version 3.1.0.508	302-303
Version 3.0.0.414	303
Version 2.2.0.310	303-304
Version 2.1.0.260	304-305
GcPdfViewer Release Notes	305
Version 3.1.2	305-307
Version 3.0.10	307-308
Version 2.2.15	308
Version 2.2.11	308
Version 2.1.14	308-309
Version 2.0.10	309
Version 1.2.88	309-310
Version 1.1.56	310
Version 1.0.42	310-311
version 5. 0. 0. 767	311
ReleaseNotes GcPDFVlewer	312
version 3. 0. 13	312

GcPdf 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 with timestamp from Time Stamp Authorities (TSA).

- **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.

- **Seamless HTML to PDF rendering**

GcPdf library along with GcHtml library, allows you to render HTML text or files to PDF documents.

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.

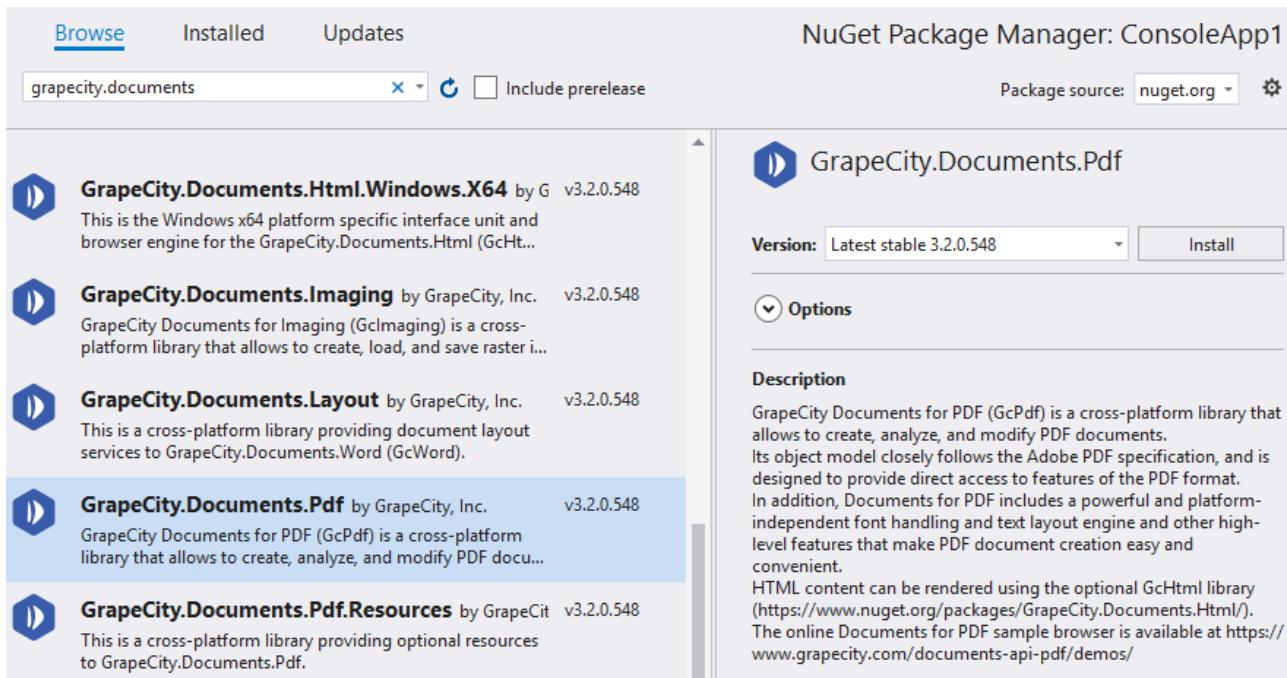
Add reference to GcPdf in your application from NuGet.org

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

- Open Visual Studio for Windows.
- Create a .NET Core Console/.NET Framework Windows Forms Application.
- Right-click the project in Solution Explorer and choose **Manage NuGet Packages**.
- In the **Package source** on top right, select **nuget.org**.
- Click **Browse** tab on top left and search for "GrapeCity.Documents".
- 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 a .NET Core Console/.NET Framework Windows Forms Application.
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+Shift+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

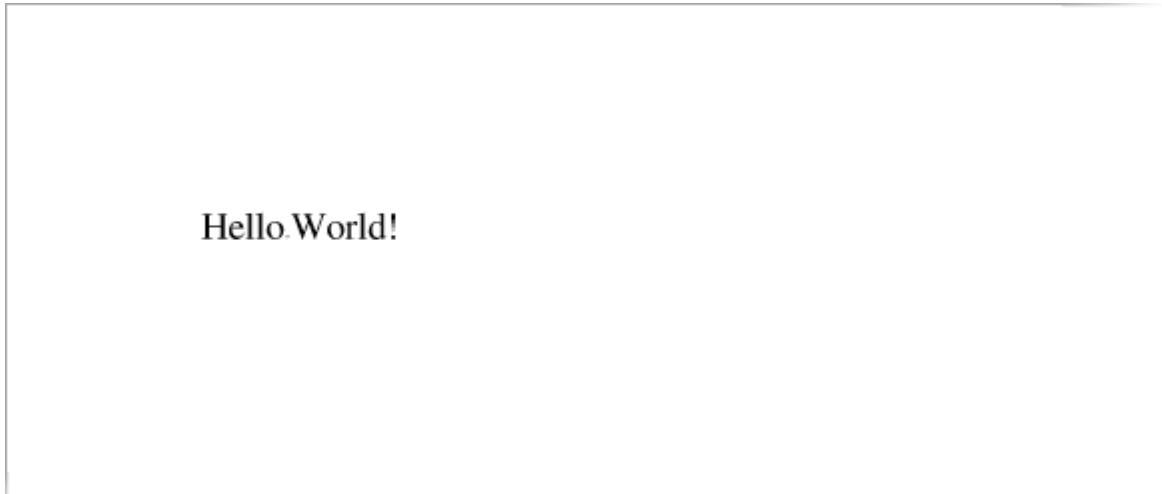
The following quick start sections help you in getting started with the GcPdf library:

- [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](#)



HelloWorld!

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
 - o 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

Contacting Sales

If you would like to find out more about our products, contact our Sales department using one of these methods:

World Wide Web site	https://www.grapecity.com/
E-mail	us.sales@grapecity.com
Phone	(800) 858-2739 or (412) 681-4343 outside the U.S.A.
Fax	(412) 681-4384

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](https://www.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: <ul style="list-style-type: none">• GrapeCity.Documents.Pdf.AcroForms• GrapeCity.Documents.Pdf.Actions• GrapeCity.Documents.Pdf.Annotations• GrapeCity.Documents.Pdf.Graphics• GrapeCity.Documents.Pdf.Log• GrapeCity.Documents.Pdf.Parser

	<ul style="list-style-type: none">• 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.)

GcHtml API Overview

GcHtml is a utility library that renders HTML to PDF file or an image in PNG or JPEG format. It is based on the industry standard Chrome web browser engine working in headless mode, offering the benefit of rendering HTML to PDF or image on any platform - Windows, Linux and macOS. Currently, GcHtml works only on Intel-based 64-bit processor architecture. Also, it doesn't matter whether your .NET application is built for x64, x86 or AnyCPU platform target. The browser is continuously working in a separate process.

The GcHtml library consists of a platform-independent main package that exposes the HTML rendering functionality, and three platform-dependent packages that implement it on Windows, Linux and macOS. The main package must always be added to a project to use GcHtml. One or more platform-dependent packages should be added depending on the target platform(s) (all three can be added). GcHtml will select and use the correct package at runtime automatically.

GcHtml NuGet Packages	Description
GrapeCity.Documents.Html	<p>The GrapeCity.Documents.Html package contains the following namespaces:</p> <ul style="list-style-type: none">• GrapeCity.Documents.Html namespace provides GcHtmlRenderer, PdfSettings, ImageSettings, PngSettings classes etc.• GrapeCity.Documents.Pdf namespace provides the GcPdfGraphicsExt and HtmlToPdfFormat classes.• GrapeCity.Documents.Drawing namespace provides GcGraphicsExt and HtmlToImageFormat class.
GrapeCity.Documents.Html.Windows.X64	It is the interface unit and Chromium browser

	<p>engine for 64-bit Windows platform.</p> <p>Note: Due to Azure Windows AppService and Azure Functions limitations, GcHtml is not supported in these environments.</p>
GrapeCity.Documents.Html.Mac.X64	<p>It is the interface unit and Chromium browser engine for macOS platform.</p>
GrapeCity.Documents.Html.Linux.X64	<p>It is the interface unit and Chromium browser engine for 64-bit Linux platform.</p> <p>Note that to use GcHtml on Linux, Chromium dependencies must be installed. The following command installs the necessary packages on Ubuntu:</p> <pre>sudo apt-get update sudo apt-get install libxss1 libappindicator1 libindicator7 libnss3-dev</pre>

Namespaces

Namespaces	Description
GrapeCity.Documents.Pdf	<p>It provides the extension methods for rendering HTML to PDF file and represents the formatting attributes for rendering HTML to PDF file.</p> <p>The namespace comprises the following classes:</p> <ul style="list-style-type: none"> • GcPdfGraphicsExt • HtmlToPdfFormat
GrapeCity.Documents.Html	<p>It provides methods for converting HTML to PDF or images and defines parameters for the PDF or image.</p> <p>The namespace comprises the following classes:</p> <ul style="list-style-type: none"> • GcHtmlRenderer • ImageSettings • JpegSettings • Margins • PdfSettings • PngSettings
GrapeCity.Documents.Drawing	<p>It provides the extension methods and formatting attributes for rendering HTML to image.</p> <p>The namespace comprises the following classes:</p> <ul style="list-style-type: none"> • GcBitmapGraphicsExt • HtmlToImageFormat

GrapeCity.Documents.Html.GcHtmlRenderer

The [GcHtmlRenderer](#) class provides methods for converting HTML to PDF and images. An instance of this class is created for each HTML page/string to be rendered. Also, it must be disposed after using to delete temporary files.

GcHtmlRenderer class has two constructors, the one that accepts Uri to the source HTML page and the other that accepts HTML as string. GcHtmlRenderer can convert the source HTML to the following formats: PDF, JPEG, PNG. The corresponding methods are [RenderToPdf](#), [RenderToJpeg](#), [RenderToPng](#). All these methods accept the output file path as the first parameter. GcHtmlRenderer always saves the result to file.

GrapeCity.Documents.Html.PdfSettings

The [PdfSettings](#) class represents output settings for rendering HTML to PDF and defines parameters for the Chromium PDF exporter. You could specify the default background color with the [DefaultBackgroundColor](#) property. In case of PDF it doesn't support any transparency. Also, setting the [DisplayBackgroundGraphics](#) property to false prevents drawing any background graphics.

[PageWidth](#) and [PageHeight](#) properties specify the page width and height in inches. If not set, the Letter paper size (8.5 by 11 inches) is used by default. If the page size is specified in the source HTML/CSS it has the priority but can be overridden by settings the [IgnoreCSSPageSize](#) property to true. The [Landscape](#) property indicates the corresponding paper orientation. All these properties can be helpful for some HTML pages and useless for others. Sometimes the markup splits pages at the fixed positions and there are no ways to improve pagination in the Chromium engine.

The [Margins](#) property specifies page margins, in inches. In GcHtml all margins are equal to 0 by default.

The [Scale](#) property effectively scales the content (the scale factor is between 0.1 and 2.0). You might also need to provide the scaled values for [PageWidth](#) and [PageHeight](#) properties to keep the relative size of the resulting pages unchanged.

The [PageRanges](#) property allows to limit the number of pages in the output PDF file. You could specify the desired page numbers as a string, such the following: "1-5, 8, 11-13". Invalid page ranges (e.g. "9-5") are ignored.

Setting the [FullPage](#) property to true allows to export the whole HTML as single PDF page. All other layout settings (except [WindowSize](#) and [Scale](#)) are ignored in that case.

The [WindowSize](#) property specifies the size of hypothetical browser window, in pixels.

GrapeCity.Documents.Pdf.HtmlToPdfFormat

The [HtmlToPdfFormat](#) class contains the formatting attributes for rendering HTML to PDF file on a [GcPdfGraphicsExt](#) class using [DrawHtml](#) extension methods. The HTML is firstly drawn to a temporary PDF as single page (if [FullPage](#) is true) or with the specified page size ([MaxPageWidth](#), [MaxPageHeight](#)), [Scale](#) and [DefaultBackgroundColor](#). It is then loaded into a [GcPdfDocument](#) and trimmed to actual size of the HTML content. The result is rendered on a [GcPdfGraphics](#) as PDF Form XObject.

If [MaxPageWidth](#) or [MaxPageHeight](#) properties are not set explicitly they are assumed to be equal to 200 inches. [DefaultBackgroundColor](#) is equal to [Color.White](#) by default.

GcPdfGraphics Extension Methods

GcHtml provides 4 methods that extend [GcPdfGraphics](#) and allow to render or measure an HTML text or page:

- Draws an HTML text on this [GcPdfGraphics](#) at a specified position:
`bool GcPdfGraphics.DrawHtml(string html, float x, float y, HtmlToPdfFormat format, out SizeF size, int virtualTimeBudget = 0);`
- Draws an HTML page specified by an URI on this [GcPdfGraphics](#) at a specified position:
`bool GcPdfGraphics.DrawHtml(Uri htmlUri, float x, float y, HtmlToPdfFormat format, out SizeF size, int virtualTimeBudget = 0);`
- Measures an HTML text for this [GcPdfGraphics](#):
`SizeF GcPdfGraphics.MeasureHtml(string html, HtmlToPdfFormat format, int virtualTimeBudget = 0);`

- Measures an HTML page specified by an URI for this GcPdfGraphics:
SizeF GcPdfGraphics.**MeasureHtml**(Uri html, HtmlToPdfFormat format, int virtualTimeBudget = 0);

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.

Output Intents

Work with Output Intents and ICC profiles in a PDF document.

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 its orientation and size, and work with content streams.

Security

Encrypt PDF and set permissions

Digital Signature

Add, remove digital signatures and achieve their custom implementation.

Soft Mask

Create soft mask in a PDF document.

Stamps

Add, modify, and delete stamps.

Tagged PDF

Create tagged PDF.

Parse PDF Documents

Parse PDF documents by recognizing their logical text and document structure.

PDF Layers

Work with PDF layers.

Text

Work with text along with paragraph handling.

Text Search

Perform text search in a PDF document.

Watermark

Add watermark.

Print

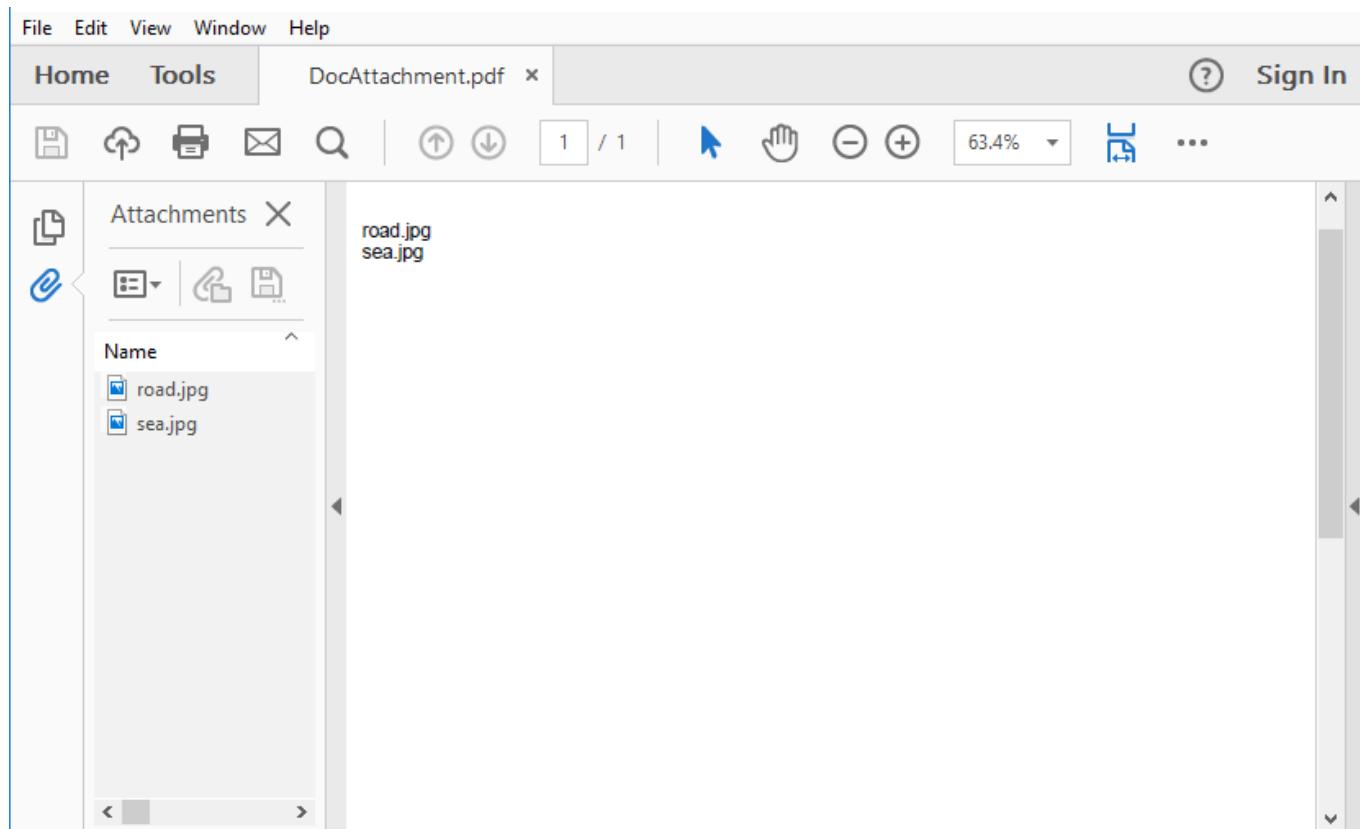
Print a PDF document.

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);

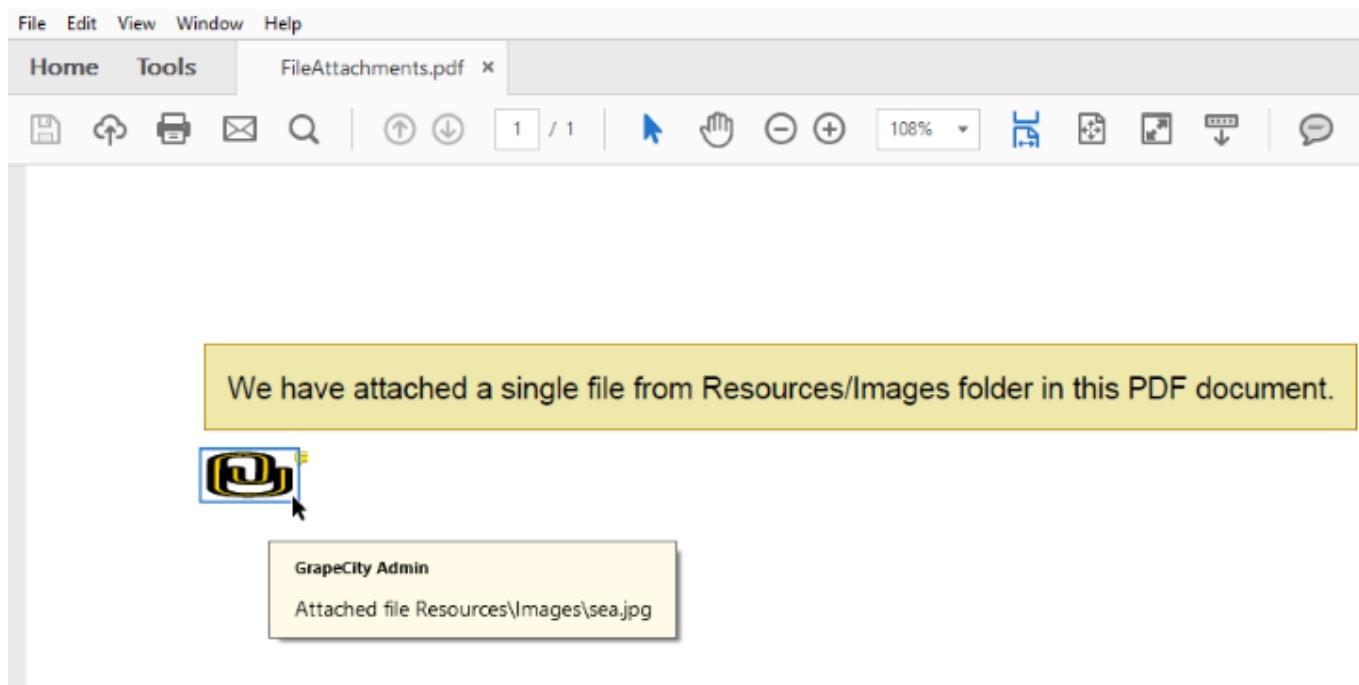
//Add string related to the attachment names
page.Graphics.DrawString(sb.ToString(), new TextFormat(), new PointF(10, 50));

//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](#)

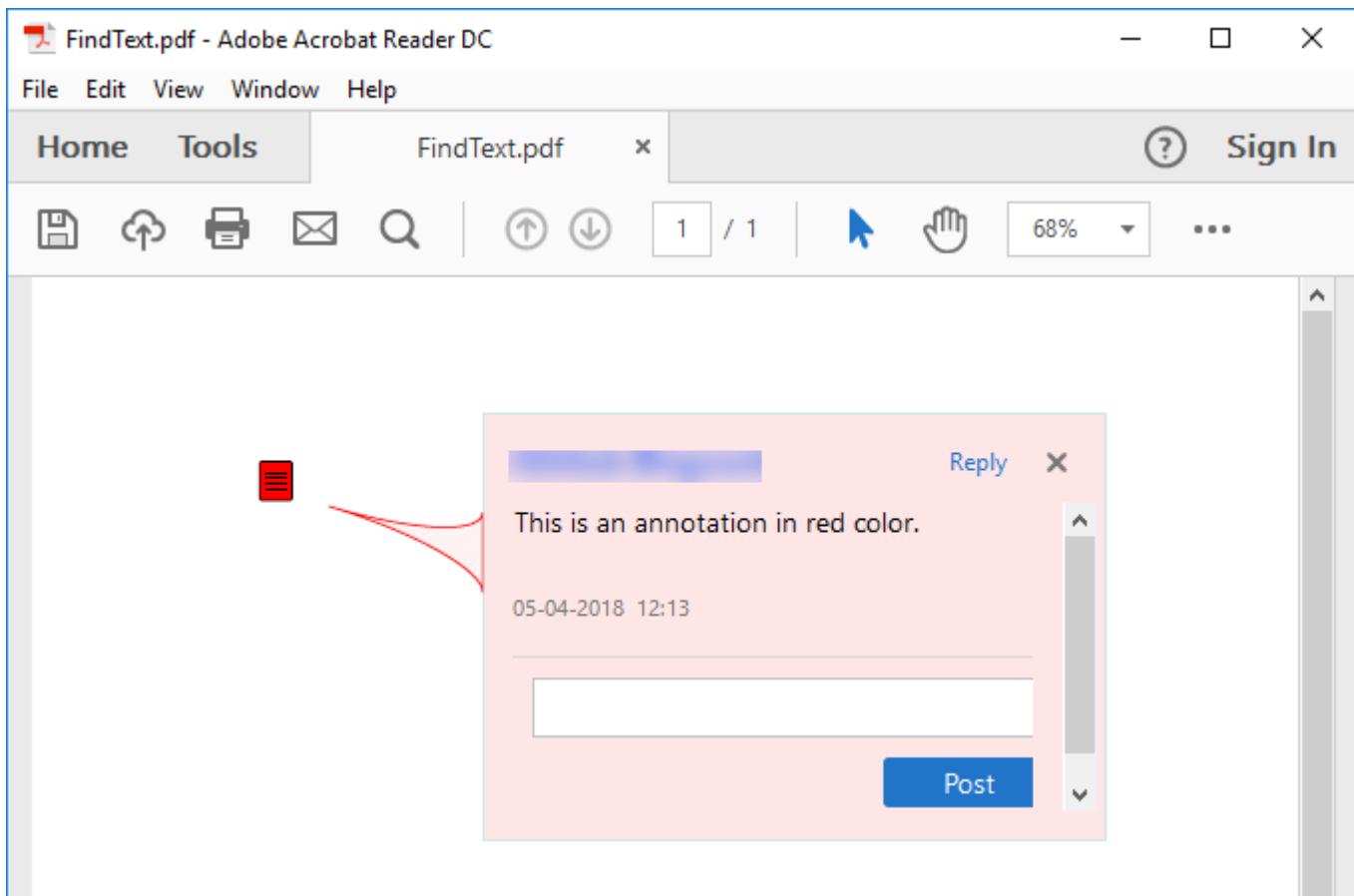
For more information about how to work with file attachments using GcPdf, see [GcPdf sample browser](#).

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](#) property of [AnnotationBase](#) class 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.

A red text annotation initially open is placed  to the right of this note.

Jamie Smith

This is a text annotation in red color.

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

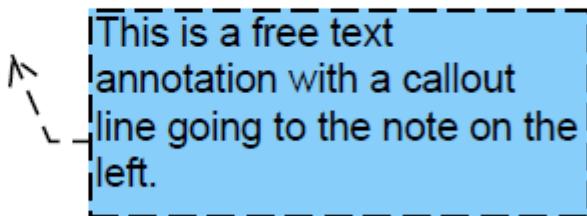
```
C#  
  
public void CreateTextAnnotation()  
{  
    GcPdfDocument doc = new GcPdfDocument();  
    Page page = doc.NewPage();  
    RectangleF rc = new RectangleF(50, 50, 200, 50);  
    page.Graphics.DrawString("A red text annotation initially open is placed to the  
right of this note.",  
  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

A blue free text annotation is placed below and to the right, with a callout going from it to this note



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

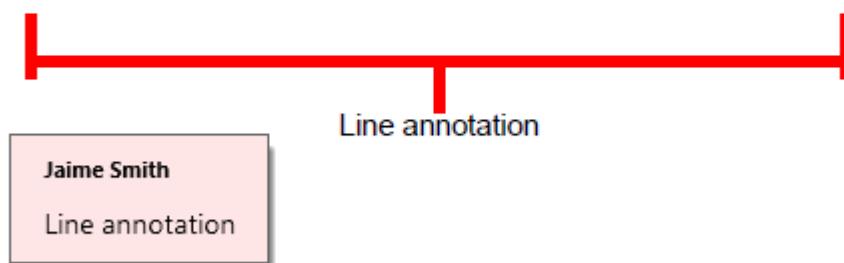
```
C#  
  
public void CreateFreeTextAnnotation()  
{  
    GcPdfDocument doc = new GcPdfDocument();  
    Page page = doc.NewPage();  
    RectangleF rc = new RectangleF(50, 50, 200, 50);  
    page.Graphics.DrawString  
        ("A blue free text annotation is placed below and to the right, " +  
        "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.

A line annotation is drawn around this note which illustrates the effect of including leader lines and caption in a line annotation



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 illustrates 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.

A square annotation drawn with a 3pt wide orange line around this note has a rich text associated with it.

Jaime Smith

This **rich text** is associated with the square annotation around a text note.

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()
```

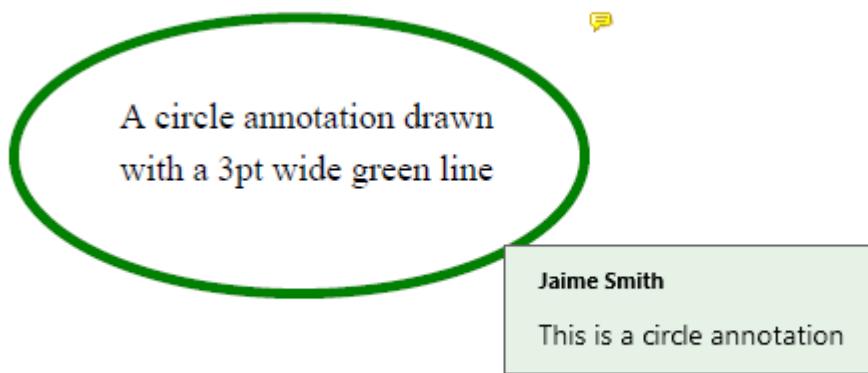
```
{  
    UserName = "Jaime Smith",  
    Rect = rc,  
    LineWidth = 3,  
    Color = Color.Orange,  
    RichText =  
        "<body><p>This <b><i>rich text</i></b> is associated with the square annotation  
around a text note.</p></body>"  
};  
page.Annotations.Add(squareAnnot); //Add the square 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);
```

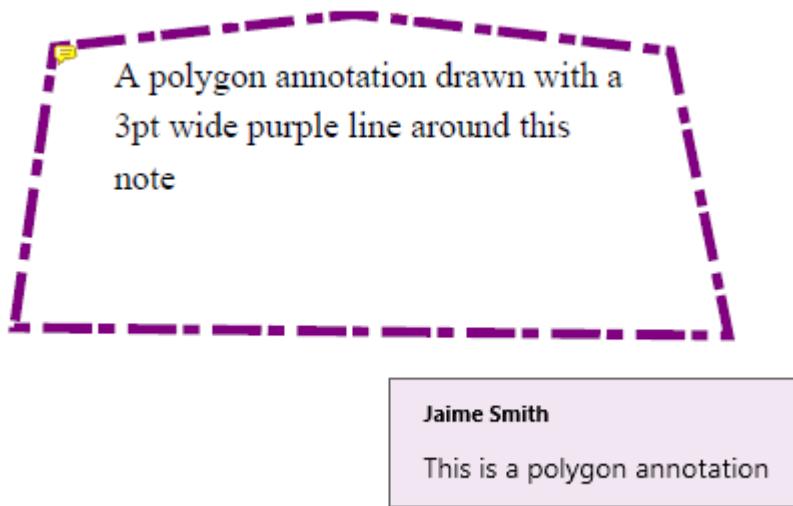
```
//Create an instance of CircleAnnotation class and set its relevant properties
var circleAnnot = new CircleAnnotation()
{
    UserName = "Jaime Smith",
    Rect = rc,
    LineWidth = 3,
    Color = Color.Green,
    Contents = "This is a circle annotation",
};

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.

This sample creates an ink annotation and shows how to use the `InkAnnotation.Paths` property



Jaime Smith

This is an ink annotation drawn via `InkAnnotation.Paths`.

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.

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



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.

A red sound annotation is placed to the right of this note. Double click the icon to play the sound.



Jaime Smith

Sound annotation with an AIFF track.

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 = SoundObjectFromFile(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 enable the users to apply widget annotations to the PDF file.

Text field:**Initial TextField value**

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.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.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.

Why JavaScript Frameworks 101?

Overall, this e-book has a singular, focused goal: to help you decide which JavaScript framework works best for you and your team by providing a technical, current, and informative summary of major JavaScript "MVC" frameworks available in 2017.

So why do you need to know about JavaScript frameworks?

1. Within the last 12 months alone, JavaScript framework usage has exploded astronomically. Using a framework when starting a new web project is the norm. From the smallest static websites to the largest stateful web apps, frameworks are utilized across the board for their unbeatable utility and software design principles. The recent explosion in popularity has diversified the features offered in the most commonly used frameworks. As such, picking the right framework for your project requires a deep knowledge of *all* available frameworks and how they compare.

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 removes the content from a PDF document that is not supposed to be shared. It can be applied in two phases:

Mark Redact Area

When you mark the redact area, a marking or highlight appears in the place of content to show that the region has been marked for redaction. With GcPdf class library, you can find all instances of texts and mark the content for redaction. This allows anyone in charge for redaction to apply redactions on the marked content. GcPdf provides [RedactAnnotation](#) class to enable the users to mark redact area in 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 mark a redact area in 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();
    //search the text(e.g employee name) which needs to be redacted
    var l = doc.FindText(new FindTextParams("Jaime Smith", true, false), null);

    // add the text's fragment area to the annotation
    redactAnnotation.Area.Add(l[0].Bounds[0]);
    redactAnnotation.Justification = VariableTextJustification.Centered;
    redactAnnotation.MarkBorderColor = Color.Black;

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

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

[Back to Top](#)

Apply Redaction

Once the areas in a PDF document are marked for redaction, redaction can be applied to those areas to remove the content from PDF documents. After the PDF content is redacted, it cannot be extracted, copied or pasted in other documents. However, you can add overlay text in the place of redacted content.

GcPdf allows you to apply redact to areas marked for redaction in PDF documents by using the [Redact](#) method of **GcPdfDocument** class. The Redact method has three overloads which provides you with the option to apply redaction to all the areas marked for redaction, to a particular area marked for redaction or a list of areas marked for redaction in a PDF document.

EMPLOYEE NAME:		TITLE: REDACTED			
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 apply redaction in a PDF document.

C#

```
var doc = new GcPdfDocument();  
using (var fs = new FileStream(Path.Combine("Resources", "PDFs",  
"TimeSheet_Redacted.pdf"), FileMode.Open, FileAccess.Read))  
{  
    // Load the PDF containing redact annotations (areas marked for redaction)
```

```
doc.Load(fs);

//mark new redact area
var rc = new RectangleF(280, 150, 100, 30);

var redact = new RedactAnnotation()
{
    Rect = rc,
    Page = doc.Pages[0],
    OverlayFillColor = Color.PaleGoldenrod,
    OverlayText = "REDACTED",
    OverlayTextRepeat = true
};

// Apply all redacts (above redact and existing area marked for redaction)
doc.Redact();

doc.Save(stream);
return doc.Pages.Count;
}
```

 **Note:** Once redact annotations are applied, they no longer exist in the PDF document. It is a destructive change, the content marked for redaction is removed from the PDF along with the redact annotations that were used to mark it.

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.

how you can create **Text markup** annotation.

Jaime Smith
This is a Text markup annotation

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));

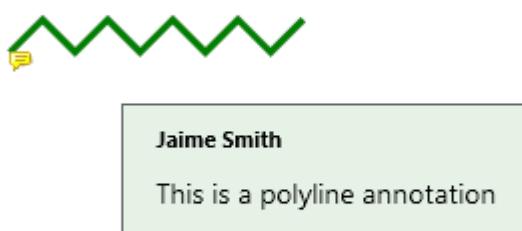
    //Create an instance of TextMarkupAnnotation class and set its relevant
    properties
    var textMarkupAnnot = new TextMarkupAnnotation();
    textMarkupAnnot.MarkupType = TextMarkupType.Highlight;
    textMarkupAnnot.UserName = "Jaime Smith";
    textMarkupAnnot.Contents = "This is a Text markup annotation";
    //search the text(e.g employee name) which needs to be redacted
    var found = doc.FindText(new FindTextParams("Text markup", true, false), null);
    foreach (var f in found)
        textMarkupAnnot.Area.Add(f.Bounds[0]);
    textMarkupAnnot.Color = Color.Yellow;

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

[Back to Top](#)

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");
}
```

Back to Top

For more information about how to work with annotations using GcPdf, see [GcPdf sample browser](#).

Apearance Streams

Annotations have their own set of appearance properties like border, color, shape etc. However, it is not necessary that all PDF viewers render the exact same appearance of an annotation. Hence, the need of appearance streams. They allow you to represent annotations visually as a set of drawing commands so that the viewer knows how to render an annotation precisely.

An annotation may have following appearances associated with their states:

- Normal Appearance: When the annotation is not interacting with the user. It is also used for printing the annotation.
- Rollover Appearance: When the user moves the cursor into the annotation's active area without pressing the mouse button.
- Down Appearance: When the mouse button is pressed or held down within the annotation's active area.

An annotation may have one or several appearance streams associated with it. An appearance stream is basically a FormXObject which is a rectangular area with graphics and anything drawn on that graphics is displayed when the annotation is in a normal, rollover or down state.

GcPdf provides an **AppearanceStreams** class which has **Normal**, **Rollover** and **Down** properties corresponding to their states. The type of these properties is Appearance and it provides access to the Default FormXObject, and a dictionary of FormXObject which is associated with specific states.

The following code illustrates how to add an appearance stream to a stamp annotation in a PDF document.

C#

```
var doc = new GcPdfDocument();
// Load an existing PDF to which we will add a stamp annotation
var jsFile = Path.Combine("Resources", "PDFs", "The-Rich-History-of-JavaScript.pdf");
using (var fs = new FileStream(jsFile, FileMode.Open, FileAccess.Read))
{
    doc.Load(fs);
    var rect = new RectangleF(PointF.Empty, doc.Pages[0].Size);
    // Create a FormXObject to use as the stamp appearance
    var fxo = new FormXObject(doc, rect);
    // Get an image from the resources, and draw it on the FormXObject's graphics
    using (var image = Image.FromFile(Path.Combine("Resources", "ImagesBis", "draft-
copy-450x72.png")))
    {
        var center = new Vector2(fxo.Bounds.Width / 2, fxo.Bounds.Height / 2);
        fxo.Graphics.Transform =
            Matrix3x2.CreateRotation((float)(-55 * Math.PI) / 180f, center) *
            Matrix3x2.CreateScale(6, center);
        fxo.Graphics.DrawImage(image, fxo.Bounds, null, ImageAlign.CenterImage);
        // Loop over pages, add a stamp to each page
        foreach (var page in doc.Pages)
        {
            // Create a StampAnnotation over the whole page
            var stamp = new StampAnnotation()
            {
                Icon = StampAnnotationIcon.Draft.ToString(),
                Name = "draft",
                Page = page,
                Rect = rect,
                UserName = "Jaime Smith"
            };
            // Add a custom appearance stream by re-using the same FormXObject on all
            // pages
            stamp.AppearanceStreams.Normal.Default = fxo;
        }
        doc.Save(stream);
    }
}
```

The resulting PDF document looks like below:

The Rich History of JavaScript

By Christian Gaetano

JavaScript wasn't always the grand language used to build massive framework systems that it is today. For a long time following its inception, JavaScript was mostly used for gimmicky website effects, like firework animations. It's come a long way from its archaic beginnings. The best way to see this dramatic, if gradual, improvement is to look at the ECMAScript standardization of JavaScript.

If you know what JavaScript is, then you also know what ECMAScript is. These two titles both refer to the same programming language. The colloquial name "JavaScript" is a strategic misnomer. Even though JavaScript syntax bears some resemblance to Java, the languages

vary widely on their core principles. **Brendan Eich**, a former Netscape employee credited with creating JavaScript in 1995, coined the name JavaScript due to Java's immense popularity at the time. *Without this marketing ploy, JavaScript may not have been adopted by the community at large.* On the other

If you know what JavaScript is, then you also know what ECMAScript is.



hand, that original name has left web developers with a confusing conundrum in modern-day usage of the language. Even though "JavaScript" is now universally recognized, its etymology often remains a mystery.

The European Computer Manufacturers Association

Shortly after JavaScript's creation, the European Computer Manufacturers Association (ECMA), which puts forth standards for many modern technological protocols and programming languages, was tasked with standardizing the language. From this effort was borne the ECMAScript (ES) specification. Although related, ECMAScript and JavaScript are not synonymous. JavaScript is an *implementation* of the ECMAScript specification. (Other implementations of the ES specification exist, though they're used much less widely than JavaScript.) Nonetheless, because of its widespread usage, JavaScript is the "poster child" of ECMAScript. Generally, and especially in this e-book, any time you see a specific ECMAScript standard revision



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

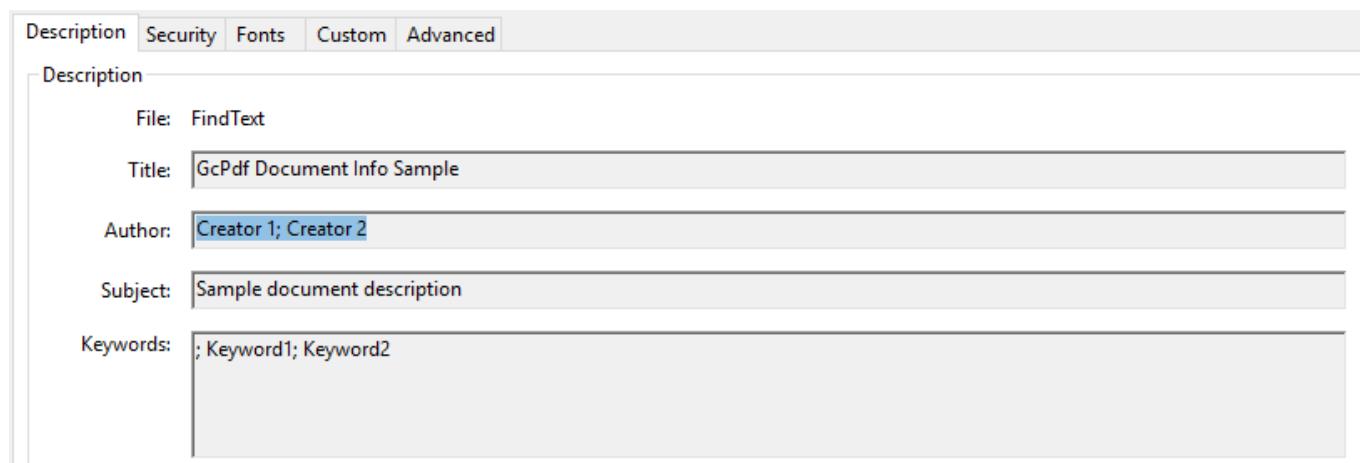
GcPdf contains [OpenAction](#) method which provides a value specifying an action that should be performed when a document is opened.

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 of GcPdfDocument class.
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, use [MergeWithDocument](#) 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](#)

Apply Redact Annotation

GcPdf allows you to apply redaction in PDF documents through document.[Redact](#) method and remove content from the document. Redaction is performed firstly by adding Redact annotation on the PDF document that marks content for redaction and then using [document.Redact](#) method to remove the content from PDF. To see more details about how to apply redact annotations, refer [Annotation Types](#).

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 [FromFile](#) method of the **Font** class.
2. Use the font (for example, Gabriola) to render a text with the [DrawString](#) method of GcPdfGraphics class.

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 of [FontCollection](#) class.. Also, you can register one or more directories containing fonts using the [RegisterDirectory](#) method of [FontCollection](#) class.
3. Assign the font collection instance to [GcGraphics.FontCollection](#) property.
4. Use [DrawString](#) method of [GcPdfGraphics](#) class 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 of [SystemFontCollection](#) class.
3. Add the list of fallback font families to global [SystemFonts](#) using [AppendFallbackFontFamilies](#) method of [SystemFontCollection](#) class.
4. Load your fallback file that includes Japanese glyphs using [AppendFallbackFonts](#) method of [SystemFontCollection](#) class.
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](#).

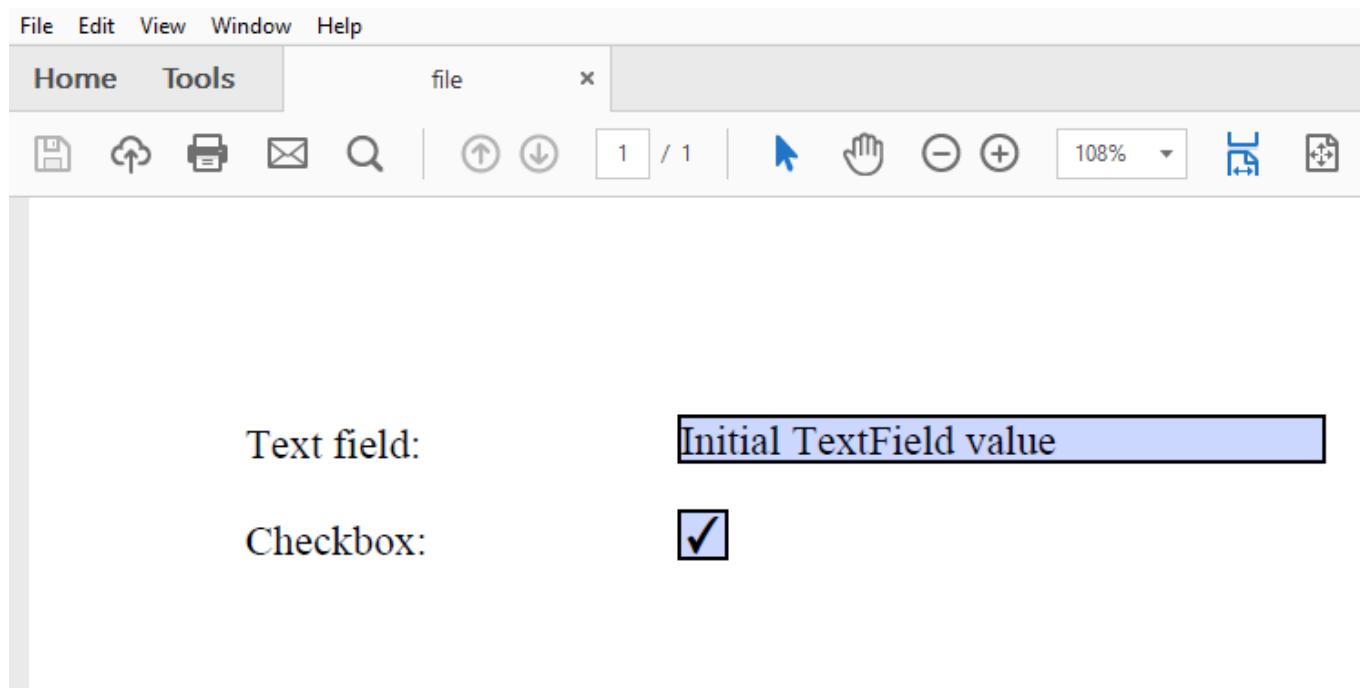
[Back to Top](#)

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](#)

Define Tab Order of Form Fields

You can navigate the form fields in a PDF document by using the 'Tab' key on the keyboard. The order of navigation of form fields can be set by using the **AnnotationTabsOrder** property which can be set to any of the below mentioned **AnnotationTabsOrder** enumeration values:

- **RowOrder:** Form fields shall be visited in rows running horizontally across the page.
- **ColumnOrder:** Form fields shall be visited in columns running vertically up and down the page.
- **StructureOrder:** Form fields shall be visited in the order in which they appear in the structure tree.

To know more about tab orders, see [PDF Specification 1.7](#) (refer section 12.5.1)

C#

```
// Set tab order of form fields to row  
page.AnnotationsTabsOrder = AnnotationsTabsOrder.RowOrder;
```

 **Note:** The tab order, set in the PDF document, can be viewed as well as edited in GcDocs PDF Viewer. Refer [Form Editor](#) for more details.

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_BankForm.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_BankForm.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_BankForm.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** class 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();  
}
```

[Back to Top](#)

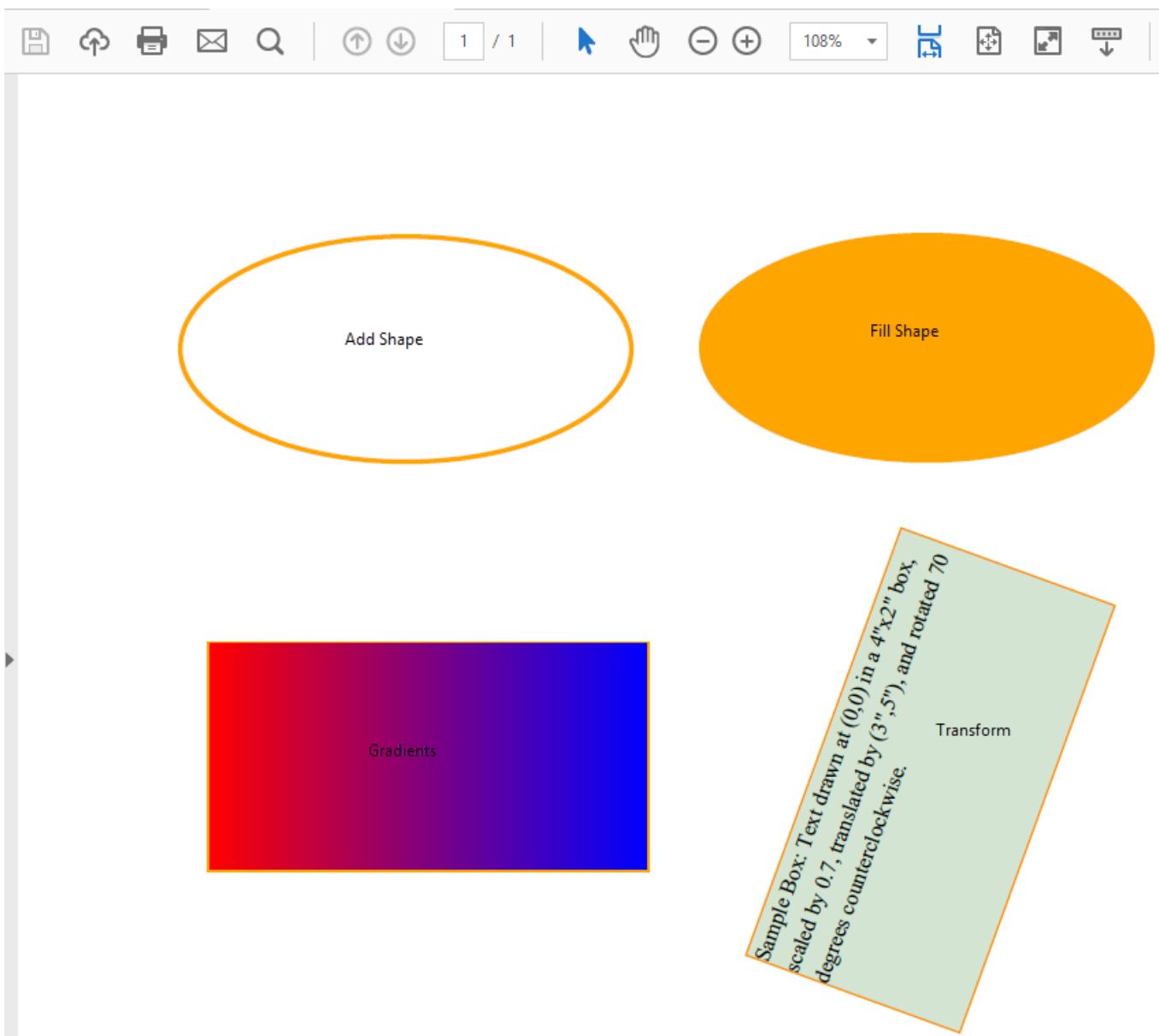
For more information about implementation of Form XObject feature using GcPdf, see [GcPdf sample browser](#).

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 of `GcGraphics` class.

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](#).

Blend Modes

Blend modes are used in computer graphics to determine how colors are blended, or mixed, with each other when drawing on a surface that already contains color information. In other words, a blend mode determines the resulting color of a colored pixel when another color is applied to it. The default is BlendMode.Normal, which simply replaces the original color with the new one.

The **GcPdfGraphics** class supports blend modes by providing PDF-specific overrides of its base **GcGraphics** class's abstract members:

- **BlendMode** Property: Gets or sets the current blend mode. The blend mode that is set using this property remains in effect for all the subsequent drawing on the current GcPdfGraphics, until changed to another value. Note that the current blend mode affects all drawing on the graphics (including text), not just images.
- **IsBlendModeSupported** Method: All blend modes are not supported in PDF (in particular, Hue, Saturation, Color and Luminosity are not supported). This method allows to programmatically check whether a particular blend mode is supported or not. An attempt to set an unsupported blend mode will throw an exception.

The below image shows different blend modes applied to two images in a PDF document:



To apply blend modes:

1. Load images on which blend modes will be applied using **FromFile** method of **Image** class.
2. Create a text layout which will be used to add captions for different blend modes by instantiating **CreateTextLayout** method and using different properties of **TextLayout** class.
3. Apply different blend modes on the loaded images by using the **BlendMode** property and rendering all the images in a grid.

C#

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

var iorchid = Image.FromFile(Path.Combine("Resources", "ImagesBis",
"orchid.jpg"));
var ispectr = Image.FromFile(Path.Combine("Resources", "ImagesBis", "spectrum-
pastel-500x500.png"));

const int margin = 36;
const int NCOLS = 4;
var w = (int)((page.Size.Width - margin * 2) / NCOLS);
var h = (int)((iorchid.Height * w) / iorchid.Width);

// Text layout for captions:
```

```
var tl = g.CreateTextLayout();
tl.DefaultFormat.Font = Font.FromFile(Path.Combine("Resources", "Fonts",
"cour.ttf"));
tl.DefaultFormatFontSize = 12;
tl.ParagraphAlignment = ParagraphAlignment.Center;
tl.TextAlignment = TextAlignment.Center;
tl.MaxWidth = w;
tl.MaxHeight = h;
tl.MarginTop = h - g.MeasureString("QWERTY", tl.DefaultFormat).Height * 1.4f;

int row = 0, col;
// Render all blending modes in a grid:
var modes = Enum.GetValues(typeof(BlendMode));
for (int i = 0; i < 2; ++i)
{
    row = col = 0;
    Image iback, ifore;
    if (i == 0)
    {
        iback = ispectr;
        ifore = iorchid;
    }
    else // i == 1
    {
        iback = iorchid;
        ifore = ispectr;
        page = doc.Pages.Add();
        g = page.Graphics;
    }
    foreach (var mode in modes)
    {
        var blendMode = (BlendMode)mode;
        if (!g.IsBlendModeSupported(blendMode))
            continue;

        int x = margin + w * col;
        int y = margin + h * row;
        var r = new RectangleF(x, y, w, h);

        g.BlendMode = BlendMode.Normal;
        g.DrawImage(iback, r, null, ImageAlign.StretchImage);
        g.BlendMode = blendMode;
        g.DrawImage(efore, r, null, ImageAlign.StretchImage);
        g.BlendMode = BlendMode.Normal;

        // Caption:
        tl.Clear();
        tl.Append(blendMode.ToString());
        tl.PerformLayout(true);
        var rc = tl.ContentRectangle;
        rc.Offset(x, y);
```

```
        rc.Inflate(4, 2);
        g.FillRectangle(rc, Color.White);
        g.DrawTextLayout(tl, new PointF(x, y));
        nextRowCol();
    }
}
doc.Save(stream);
// 
void nextRowCol()
{
    if (++col == NCOLS)
    {
        col = 0;
        ++row;
    }
}
```

Limitation

The Hue, Saturation, Color, and Luminosity blend modes are not supported in GcPdf as they are not supported in the PDF specification.

Output Intents

Output intents describe the color characteristics of output devices on which the PDF document might be rendered.

GcPdf uses the **OutputIntents** property to specify the output intents of a **GcPdfDocument** class. The **OutputIntent** class represents a PDF output intent and provides the **Create** method to create an output intent. The output intent subtype can also be set by using the **Subtype** property of **OutputIntent** class and has the following types:

- GTS_PDFX
- GTS_PDFA1
- ISO_PDFE1

An output intent can be used in conjunction with the ICC profiles to convert the source colors to those required for the intended output. The ICC profiles are used for describing the output intents in PDF/A, PDF/X and PDF/VT standards. GcPdf provides the **ICCProfile** class which represents the ICC profile required to create the output intent. To know more about Output Intents, see [PDF Specification 1.7](#) (refer section 14.11.5)

Refer to the following code example to set output intents and ICC profiles in a PDF document:

```
C#
// The different versions of the ICC Probe profile
var profiles = new (string, string)[] {
    ("Probev2_ICCv4.icc", @"https://www.color.org/probeprofile.xalter"),
    ("Probevl_ICCv4.icc", @"https://www.color.org/probeprofile.xalter"),
    ("Probevl_ICCv2.icc", @"https://www.color.org/probeprofile.xalter"),
};

var doc = new GcPdfDocument();
var page = doc.NewPage();
var g = page.Graphics;
var sb = new StringBuilder();
const string bullet = "\x2022\x2003";
sb.AppendLine("This document contains the following output intents (first one is the default):");
int i = 0;
foreach (var profile in profiles)
{
    sb.AppendLine($"{bullet}{profile.Item1}, source: {profile.Item2}");
    using (FileStream fs = File.OpenRead(Path.Combine("Resources", "Misc", profile.Item1)))
    {
        var oi = OutputIntent.Create($"Output intent testing {i++}", "", "http://www.color.org", profile.Item1, fs);
        doc.OutputIntents.Add(oi);
    }
}
var rc = Common.Util.AddNote(sb.ToString(), page);
g.DrawImage(Image.FromFile(Path.Combine("Resources", "Images", "roofs.jpg")),
    new RectangleF(rc.Left, rc.Bottom + 24, rc.Width, rc.Width), null, ImageAlign.StretchImage);

doc.Save(stream);
```

The above code example uses the ICC Probe Profile whose colors are deliberately distorted after processing, so that it is easy to visually confirm that a profile is being used. To see the effect in the PDF, you can open it in Adobe Acrobat Reader DC and set Edit > Preferences > Page Display > Use Overprint Preview to 'Always', as shown below:

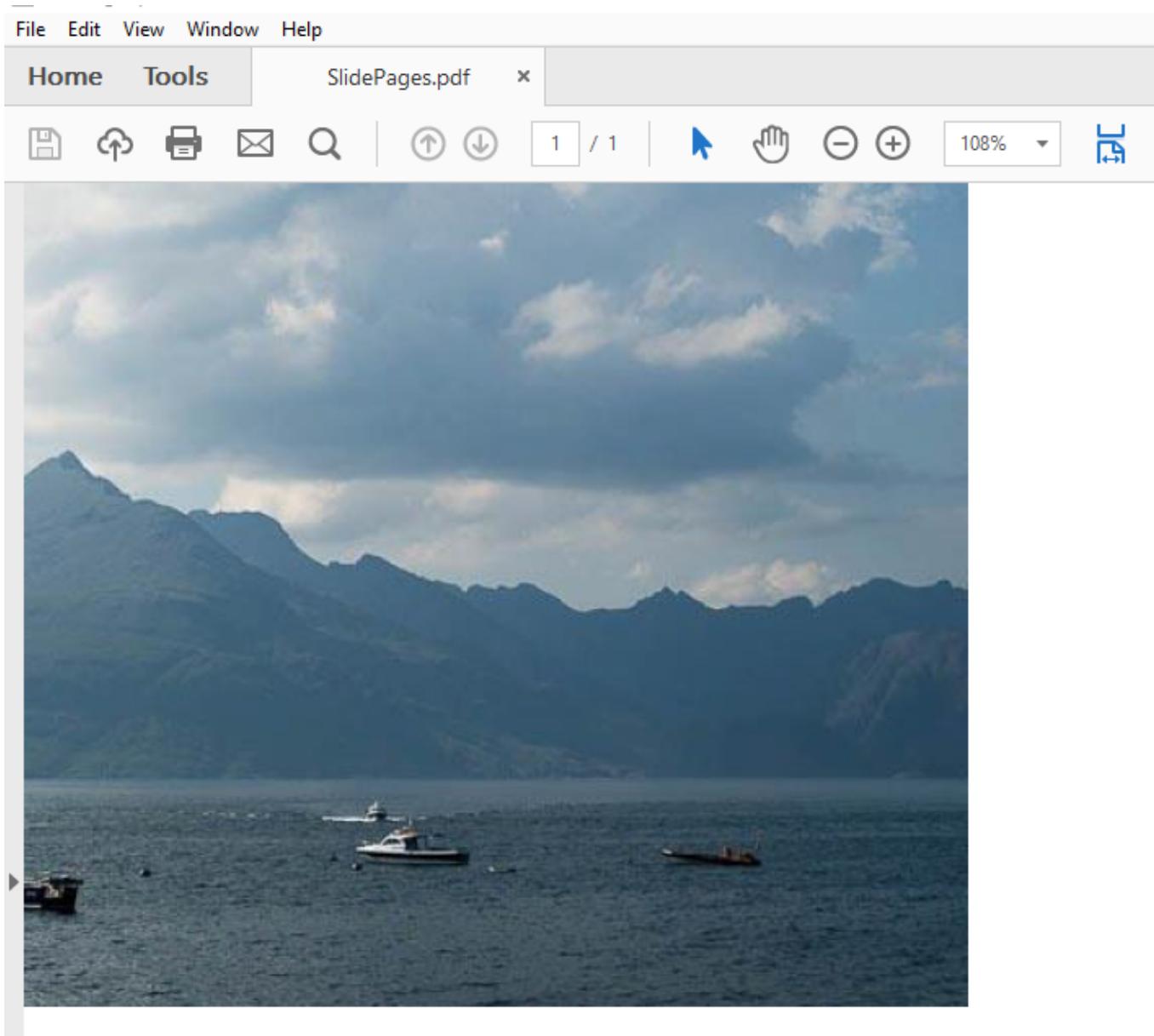
Before setting the 'Use Overprint Preview' option	After setting the 'Use Overprint Preview' option
<p>This document contains the following output intents (first one is the default):</p> <ul style="list-style-type: none">• Probev2_ICCv4.icc, source: https://www.color.org/probeprofile.xalter• Probev1_ICCv4.icc, source: https://www.color.org/probeprofile.xalter• Probev1_ICCv2.icc, source: https://www.color.org/probeprofile.xalter 	<p>This document contains the following output intents (first one is the default):</p> <ul style="list-style-type: none">• Probev2_ICCv4.icc, source: https://www.color.org/probeprofile.xalter• Probev1_ICCv4.icc, source: https://www.color.org/probeprofile.xalter• Probev1_ICCv2.icc, source: https://www.color.org/probeprofile.xalter 

 **Note:** The ICC profiles used in the above code can be downloaded from [ICC Profile Registry](#).

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, SVG, 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);
}
```

[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();
```

[Back to Top](#)

For more information about implementation of images using GcPdf, see [GcPdf sample browser](#).

Create SVG Image using Code

To create an SVG image using code:

1. Create a new SVG document by creating an instance of **GcSvgDocument**.
2. Create an instance of **SvgPathBuilder** class. This class provides methods to execute the path commands.
3. Define the path to draw the outline of shape to be drawn on SVG using methods such as **AddMoveTo** and **AddCurveTo**.
4. Add these elements into root collection of 'svg' element using the **Add** method.
5. Provide the **SvgPathData** using **ToPathData** method of the **SvgPathBuilder** class which represents sequence of instructions for drawing the path.
6. Define other properties of each path such as, Fill, Stroke etc.
7. Save the document as SVG by using **Save** method of the **GcSvgDocument** class.

C#

```
public static GcSvgDocument DrawCarrot()
{
    // Create a new SVG document
    var doc = new GcSvgDocument();
    var svg = doc.RootSvg;
    svg.ViewBox = new SvgViewBox(0, 0, 313.666f, 164.519f);

    //Create an instance of SvgPathBuilder class.
    var pb = new SvgPathBuilder();

    //Define the path
    pb.AddMoveTo(false, 29.649f, 9.683f);
    pb.AddCurveTo(true, -2.389f, -0.468f, -4.797f, 2.57f, -6.137f, 5.697f);
    pb.AddCurveTo(true, 2.075f, -2.255f, 3.596f, -1.051f, 4.91, 5f, -0.675f);
    pb.AddCurveTo(true, -2.122f, 2.795f, -4f, 5.877f, -7.746f, 5.568f);
    pb.AddCurveTo(true, 2.384f, -6.014f, 2.963f, -12.977f, 0.394f, -17.78f);
    pb.AddCurveTo(true, -1.296f, 2.591f, -1.854f, 6.054f, -5.204f, 7.395f);
    pb.AddCurveTo(true, 3.575f, 2.455f, 0.986f, 7.637f, 1.208f, 11.437f);
    pb.AddCurveTo(false, 11.967f, 21.17f, 6.428f, 16.391f, 9.058f, 10.67f);
    pb.AddCurveTo(true, -3.922f, 8.312f, -2.715f, 19.745f, 4.363f, 22.224f);
    pb.AddCurveTo(true, -3.86f, 4.265f, -2.204f, 10.343f, 0.209f, 13.781f);
    pb.AddCurveTo(true, -0.96f, 1.808f, -1.83f, 2.546f, -3.774f, 3.195f);
    pb.AddCurveTo(true, 3.376f, 1.628f, 6.612f, 4.866f, 11.326f, 3.366f);
    pb.AddCurveTo(true, -1.005f, 2.345f, -12.389f, 9.499f, -15.16f, 10.35f);
    pb.AddCurveTo(true, 3.216f, 0.267f, 14.492f, -2.308f, 16.903f, -5.349f);
    pb.AddCurveTo(true, -1.583f, 2.84f, 1.431f, 2.28f, 2.86f, 4.56f);
    pb.AddCurveTo(true, 1.877f, -3.088f, 3.978f, -2.374f, 5.677f, -3.311f);
    pb.AddCurveTo(true, -0.406f, 4.826f, -2.12f, 9.27f, -5.447f, 13.582f);
    pb.AddCurveTo(true, 2.834f, -4.894f, 6.922f, -5.367f, 10.474f, -5.879f);
    pb.AddCurveTo(true, -0.893f, 4.245f, -3.146f, 8.646f, -7.077f, 10.479f);
    pb.AddCurveTo(true, 5.359f, 0.445f, 11.123f, -3.934f, 13.509f, -9.944f);
    pb.AddCurveTo(true, 12.688f, 3.209f, 28.763f, -1.932f, 39.894f, 7.084f);
    pb.AddCurveTo(true, 1.024f, 0.625f, 1.761f, -4.98f, 1.023f, -5.852f);
    pb.AddCurveTo(false, 72.823f, 55.357f, 69.273f, 68.83f, 52.651f, 54.498f);
    pb.AddCurveTo(true, -0.492f, -0.584f, 1.563f, -5.81f, 1f, -8.825f);
    pb.AddCurveTo(true, -1.048f, -3.596f, -3.799f, -6.249f, -7.594f, -6.027f);
```

```
pb.AddCurveTo(true, -2.191f, 0.361f, -5.448f, 0.631f, -7.84f, 0.159f);
pb.AddCurveTo(true, 2.923f, -5.961f, 9.848f, -4.849f, 12.28f, -11.396f);
pb.AddCurveTo(true, -4.759f, 2.039f, -7.864f, -2.808f, -12.329f, -1.018f);
pb.AddCurveTo(true, 1.63f, -3.377f, 4.557f, -2.863f, 6.786f, -3.755f);
pb.AddCurveTo(true, -3.817f, -2.746f, -9.295f, -5.091f, -14.56f, -0.129f);
pb.AddCurveTo(false, 33.228f, 18.615f, 32.064f, 13.119f, 29.649f, 9.683f);

//Add elements into Children collection of SVG
svg.Children.Add(new SvgPathElement()
{
    FillRule = SvgFillRule.EvenOdd,
    Fill = new SvgPaint(Color.FromArgb(0x43, 0x95, 0x39)),
    PathData = pb.ToPathData(),
});

pb.Reset();
pb.AddMoveTo(false, 29.649f, 9.683f);
pb.AddCurveTo(true, -2.389f, -0.468f, -4.797f, 2.57f, -6.137f, 5.697f);
pb.AddCurveTo(true, 2.075f, -2.255f, 3.596f, -1.051f, 4.915f, -0.675f);
pb.AddCurveTo(true, -2.122f, 2.795f, -4f, 5.877f, -7.746f, 5.568f);
pb.AddCurveTo(true, 2.384f, -6.014f, 2.963f, -12.977f, 0.394f, -17.78f);
pb.AddCurveTo(true, -1.296f, 2.591f, -1.854f, 6.054f, -5.204f, 7.395f);
pb.AddCurveTo(true, 3.575f, 2.455f, 0.986f, 7.637f, 1.208f, 11.437f);
pb.AddCurveTo(false, 11.967f, 21.17f, 6.428f, 16.391f, 9.058f, 10.67f);
pb.AddCurveTo(true, -3.922f, 8.312f, -2.715f, 19.745f, 4.363f, 22.224f);
pb.AddCurveTo(true, -3.86f, 4.265f, -2.204f, 10.343f, 0.209f, 13.781f);
pb.AddCurveTo(true, -0.96f, 1.808f, -1.83f, 2.546f, -3.774f, 3.195f);
pb.AddCurveTo(true, 3.376f, 1.628f, 6.612f, 4.866f, 11.326f, 3.366f);
pb.AddCurveTo(true, -1.005f, 2.345f, -12.389f, 9.499f, -15.16f, 10.35f);
pb.AddCurveTo(true, 3.216f, 0.267f, 14.492f, -2.308f, 16.903f, -5.349f);
pb.AddCurveTo(true, -1.583f, 2.84f, 1.431f, 2.28f, 2.86f, 4.56f);
pb.AddCurveTo(true, 1.877f, -3.088f, 3.978f, -2.374f, 5.677f, -3.311f);
pb.AddCurveTo(true, -0.406f, 4.826f, -2.12f, 9.27f, -5.447f, 13.582f);
pb.AddCurveTo(true, 2.834f, -4.894f, 6.922f, -5.367f, 10.474f, -5.879f);
pb.AddCurveTo(true, -0.893f, 4.245f, -3.146f, 8.646f, -7.077f, 10.479f);
pb.AddCurveTo(true, 5.359f, 0.445f, 11.123f, -3.934f, 13.509f, -9.944f);
pb.AddCurveTo(true, 12.688f, 3.209f, 28.763f, -1.932f, 39.894f, 7.084f);
pb.AddCurveTo(true, 1.024f, 0.625f, 1.761f, -4.98f, 1.023f, -5.852f);
pb.AddCurveTo(false, 72.823f, 55.357f, 69.273f, 68.83f, 52.651f, 54.498f);
pb.AddCurveTo(true, -0.492f, -0.584f, 1.563f, -5.81f, 1f, -8.825f);
pb.AddCurveTo(true, -1.048f, -3.596f, -3.799f, -6.249f, -7.594f, -6.027f);
pb.AddCurveTo(true, -2.191f, 0.361f, -5.448f, 0.631f, -7.84f, 0.159f);
pb.AddCurveTo(true, 2.923f, -5.961f, 9.848f, -4.849f, 12.28f, -11.396f);
pb.AddCurveTo(true, -4.759f, 2.039f, -7.864f, -2.808f, -12.329f, -1.018f);
pb.AddCurveTo(true, 1.63f, -3.377f, 4.557f, -2.863f, 6.786f, -3.755f);
pb.AddCurveTo(true, -3.817f, -2.746f, -9.295f, -5.091f, -14.56f, -0.129f);
pb.AddCurveTo(false, 33.228f, 18.615f, 32.064f, 13.119f, 29.649f, 9.683f);
pb.AddClosePath();

//Add elements into Children collection of SVG
svg.Children.Add(new SvgPathElement())
{
```

```
        Fill = SvgPaint.None,
        Stroke = new SvgPaint(Color.Black),
        StrokeWidth = new SvgLength(2.292f),
        StrokeMiterLimit = 14.3f,
        PathData = pb.ToPathData(),
    });

pb.Reset();
pb.AddMoveTo(false, 85.989f, 101.047f);
pb.AddCurveTo(true, 0f, 0f, 3.202f, 3.67f, 8.536f, 4.673f);
pb.AddCurveTo(true, 7.828f, 1.472f, 17.269f, 0.936f, 17.269f, 0.936f);
pb.AddCurveTo(true, 0f, 0f, 2.546f, 5.166f, 10.787f, 7.338f);
pb.AddCurveTo(true, 8.248f, 2.168f, 17.802f, 0.484f, 17.802f, 0.484f);
pb.AddCurveTo(true, 0f, 0f, 8.781f, 1.722f, 19.654f, 8.074f);
pb.AddCurveTo(true, 10.871f, 6.353f, 20.142f, 2.163f, 20.142f, 2.163f);
pb.AddCurveTo(true, 0f, 0f, 1.722f, 3.118f, 14.11f, 9.102f);
pb.AddCurveTo(true, 12.39f, 5.982f, 14.152f, 2.658f, 28.387f, 4.339f);
pb.AddCurveTo(true, 14.232f, 1.672f, 19.36f, 5.568f, 30.108f, 7.449f);
pb.AddCurveTo(true, 10.747f, 1.886f, 25.801f, 5.607f, 25.801f, 5.607f);
pb.AddCurveTo(true, 0f, 0f, 4.925f, 0.409f, 12.313f, 6.967f);
pb.AddCurveTo(true, 7.381f, 6.564f, 18.453f, 4.506f, 18.453f, 4.506f);
pb.AddCurveTo(true, 0f, 0f, -10.869f, -6.352f, -15.467f, -10.702f);
pb.AddCurveTo(true, -4.594f, -4.342f, -16.901f, -11.309f, -24.984f, -
15.448f);
    pb.AddCurveTo(true, -8.079f, -4.14f, -18.215f, -7.46f, -30.233f, -11.924f);
    pb.AddCurveTo(true, -12.018f, -4.468f, -6.934f, -6.029f, -23.632f, -
13.855f);
    pb.AddCurveTo(true, -16.695f, -7.822f, -13.662f, -8.565f, -28.347f, -
10.776f);
    pb.AddCurveTo(true, -14.686f, -2.208f, -6.444f, -11.933f, -23.917f, -
16.356f);
    pb.AddCurveTo(true, -17.479f, -4.423f, -11.037f, -4.382f, -26.016f, -
9.093f);
    pb.AddCurveTo(true, -14.97f, -4.715f, -10.638f, -10.104f, -26.665f, -
13.116f);
    pb.AddCurveTo(true, -14.149f, -2.66f, -21.318f, 0.468f, -27.722f, 11.581f);
    pb.AddCurveTo(false, 73.104f, 89.075f, 85.989f, 101.047f, 85.989f,
101.047f);
    // Add elements into Children collection of SVG
    svg.Children.Add(new SvgPathElement())
    {
        FillRule = SvgFillRule.EvenOdd,
        Fill = new SvgPaint(Color.FromArgb(0xFF, 0xC2, 0x22)),
        PathData = pb.ToPathData(),
    });

pb.Reset();
pb.AddMoveTo(false, 221.771f, 126.738f);
pb.AddCurveTo(true, 0f, 0f, 1.874f, -4.211f, 4.215f, -6.087f);
pb.AddCurveTo(true, 2.347f, -1.868f, 2.812f, -2.339f, 2.812f, -2.339f);
pb.AddMoveTo(false, 147.11f, 105.122f);
```

```
pb.AddCurveTo(true, 0f, 0f, 0.882f, -11.047f, 6.765f, -15.793f);
pb.AddCurveTo(true, 5.879f, -4.745f, 10.882f, -5.568f, 10.882f, -5.568f);
pb.AddMoveTo(false, 125.391f, 86.008f);
pb.AddCurveTo(true, 0f, 0f, 2.797f, -6.289f, 6.291f, -9.081f);
pb.AddCurveTo(true, 3.495f, -2.791f, 4.194f, -3.49f, 4.194f, -3.49f);
pb.AddMoveTo(false, 181.153f, 124.8f);
pb.AddCurveTo(true, 0f, 0f, -1.206f, -4.014f, -0.709f, -6.671f);
pb.AddCurveTo(true, 0.493f, -2.66f, 0.539f, -3.256f, 0.539f, -3.256f);
pb.AddMoveTo(false, 111.704f, 107.641f);
pb.AddCurveTo(true, 0f, 0f, -1.935f, -6.604f, -1.076f, -10.991f);
pb.AddCurveTo(true, 0.862f, -4.389f, 0.942f, -5.376f, 0.942f, -5.376f);
pb.AddMoveTo(false, 85.989f, 101.047f);
pb.AddCurveTo(true, 0f, 0f, 3.202f, 3.67f, 8.536f, 4.673f);
pb.AddCurveTo(true, 7.828f, 1.472f, 17.269f, 0.936f, 17.269f, 0.936f);
pb.AddCurveTo(true, 0f, 0f, 2.546f, 5.166f, 10.787f, 7.338f);
pb.AddCurveTo(true, 8.248f, 2.168f, 17.802f, 0.484f, 17.802f, 0.484f);
pb.AddCurveTo(true, 0f, 0f, 8.781f, 1.722f, 19.654f, 8.074f);
pb.AddCurveTo(true, 10.871f, 6.353f, 20.142f, 2.163f, 20.142f, 2.163f);
pb.AddCurveTo(true, 0f, 0f, 1.722f, 3.118f, 14.11f, 9.102f);
pb.AddCurveTo(true, 12.39f, 5.982f, 14.152f, 2.658f, 28.387f, 4.339f);
pb.AddCurveTo(true, 14.232f, 1.672f, 19.36f, 5.568f, 30.108f, 7.449f);
pb.AddCurveTo(true, 10.747f, 1.886f, 25.801f, 5.607f, 25.801f, 5.607f);
pb.AddCurveTo(true, 0f, 0f, 4.925f, 0.409f, 12.313f, 6.967f);
pb.AddCurveTo(true, 7.381f, 6.564f, 18.453f, 4.506f, 18.453f, 4.506f);
pb.AddCurveTo(true, 0f, 0f, -10.869f, -6.352f, -15.467f, -10.702f);
pb.AddCurveTo(true, -4.594f, -4.342f, -16.901f, -11.309f, -24.984f, -
15.448f);
pb.AddCurveTo(true, -8.079f, -4.14f, -18.215f, -7.46f, -30.233f, -11.924f);
pb.AddCurveTo(true, -12.018f, -4.468f, -6.934f, -6.029f, -23.632f, -
13.855f);
pb.AddCurveTo(true, -16.695f, -7.822f, -13.662f, -8.565f, -28.347f, -
10.776f);
pb.AddCurveTo(true, -14.686f, -2.208f, -6.444f, -11.933f, -23.917f, -
16.356f);
pb.AddCurveTo(true, -17.479f, -4.423f, -11.037f, -4.382f, -26.016f, -
9.093f);
pb.AddCurveTo(true, -14.97f, -4.715f, -10.638f, -10.104f, -26.665f, -
13.116f);
pb.AddCurveTo(true, -14.149f, -2.66f, -21.318f, 0.468f, -27.722f, 11.581f);
pb.AddCurveTo(false, 73.104f, 89.075f, 85.989f, 101.047f, 85.989f,
101.047f);
pb.AddClosePath();

//Add elements into Children collection of SVG
svg.Children.Add(new SvgPathElement()
{
    Fill = SvgPaint.None,
    Stroke = new SvgPaint(Color.Black),
    StrokeWidth = new SvgLength(3.056f),
    StrokeMiterLimit = 11.5f,
    PathData = pb.ToPathData(),
```

```
    });
    //Save the document as svg
    doc.Save("demo.svg");
    return doc;
}
```

Render SVG Image to PDF File

The GrapeCity.Documents.Svg namespace provides **GcSvgDocument** class which can be used to render SVG files on PDF pages.

To render an SVG image file to a PDF document:

1. Load an SVG image in a PDF document by using the **FromFile** method of **GcSvgDocument** class.
2. Draw the specified SVG document at a location in PDF document by using **DrawSvg** method of **GcGraphics** class.
3. Save the document containing SVG image using **Save** method of the **GcPdfDocument** class.

C#

```
var doc = new GcPdfDocument();
var g = doc.NewPage().Graphics;
var prevT = g.Transform;
g.Transform = Matrix3x2.CreateScale(factor);
using var svg = GcSvgDocument.FromFile("Rectangle.svg");
g.DrawSvg(svg, new PointF(72 / factor, 72 / factor));
g.Transform = prevT;
doc.Save("SVGImage.pdf");
```

[Back to Top](#)

For more information about rendering SVG images to PDF files using GcPdf, see [GcPdf sample browser](#).

You can also render an SVG image to a PNG file, create, load, inspect, modify, and save the internal structure of an SVG image. For more information, refer [Work with SVG Files](#) topic in GclImaging docs.

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.

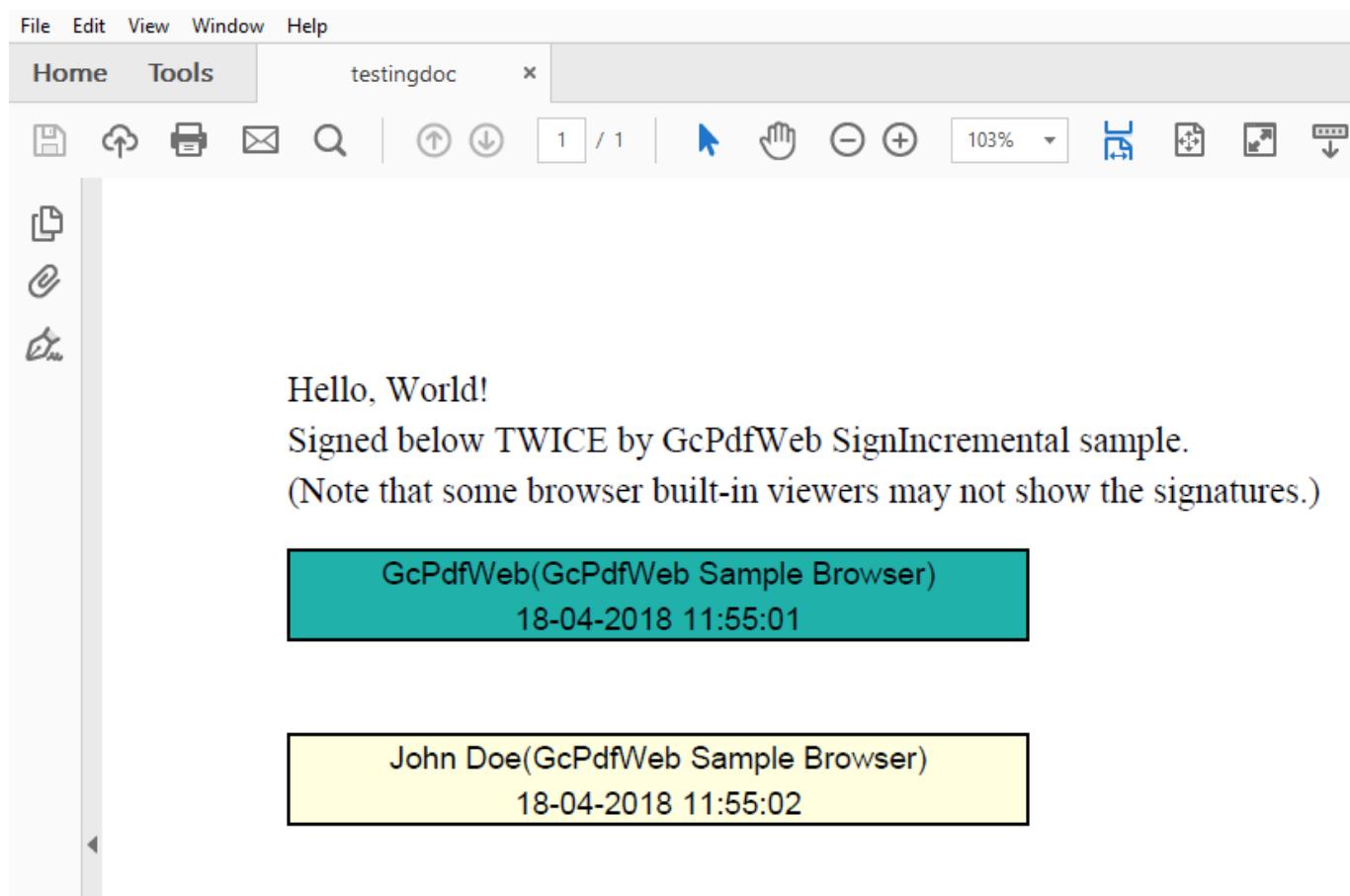
GcPdf allows you to incrementally update and save a modified document using **Save** method. By default, this method saves the document without an incremental update. However, you can save the document incrementally by setting the **incrementalUpdate** parameter of the **Save** method to true. The **Save** method provides two more overloads which take **SaveMode** enumeration as a parameter. The **SaveMode** enumeration gives you option to save the PDF documents in default mode, linearized mode or incremental update mode. To save a document with incremental updates, you can also set the **SaveMode** enumeration to **IncrementalUpdate** while passing it as a parameter to the **Save** method. Note that incremental updates can not be included in a linearized document. That is, linearized and incremental update modes are mutually exclusive modes. For more information regarding linearized mode, see [Linearization](#).

Additionally, GcPdf provides **Sign** method to sign and save a document which by default updates the document incrementally. Alternatively, you can also set the **SaveMode** enumeration to **IncrementalUpdate** and pass it as a parameter to **Sign** method. Both these methods let 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 of GcPdfDocument class.
3. Modify the document. For example, add some text or graphical element to the document.
4. Save the document using [Save](#) method of GcPdfDocument class 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);

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);

// Alternatively, use the below overload.
// doc.Save("IncUpdate", SaveMode.IncrementalUpdate);
}
```

[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 of GcPdfDocument class.
7. Load the signed document using the [Load](#) method of GcPdfDocument class.
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 document 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",
    "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. It makes the web viewing faster and user does not need to wait for the entire PDF to load to start viewing the document. GcPdf allows you to linearize the PDF documents and also lets you fetch the linearized status of a document. For more information on linearization, see [PDF specification 1.7](#) (Annexure F).

Linearize a Document

With GcPdf library, you can generate a linearized PDF by using **Save** method of the **GcPdfDocument** class. The Save method provides overloads which takes **SaveMode** enumeration as a parameter along with other mandatory parameters. The SaveMode enumeration gives you option to save the PDF documents in default mode, linearized mode or incremental update mode. Linearized document cannot contain incremental updates. That is, linearized and

incremental update modes are mutually exclusive modes. To save a document as linearized PDF, you can set the **SaveMode** enumeration to **Linearized** while passing it as a parameter to the Save method. For more information regarding incremental update mode, see [Incremental Update](#).

 **Note:** For customers using **Linearized** property to linearize PDF documents, please note that the **Linearized** property has been changed to readonly in v5.0 release. If you are using v5.0 build or later, you must now use the `Save(xxx, SaveMode)` method as mentioned above in order to linearize the documents.

The example below shows how to linearize an existing PDF document.

C#

```
// Create a new PDF document:  
GcPdfDocument doc = new GcPdfDocument();  
  
// Modify the document as required  
// ...  
  
// Save the document in linearized mode  
doc.Save("Demo.pdf", SaveMode.Linearized);
```

Get Linearized State

To fetch the linearized state of a PDF document, you can use **Linearized** property of the **GcPdfDocument** class.

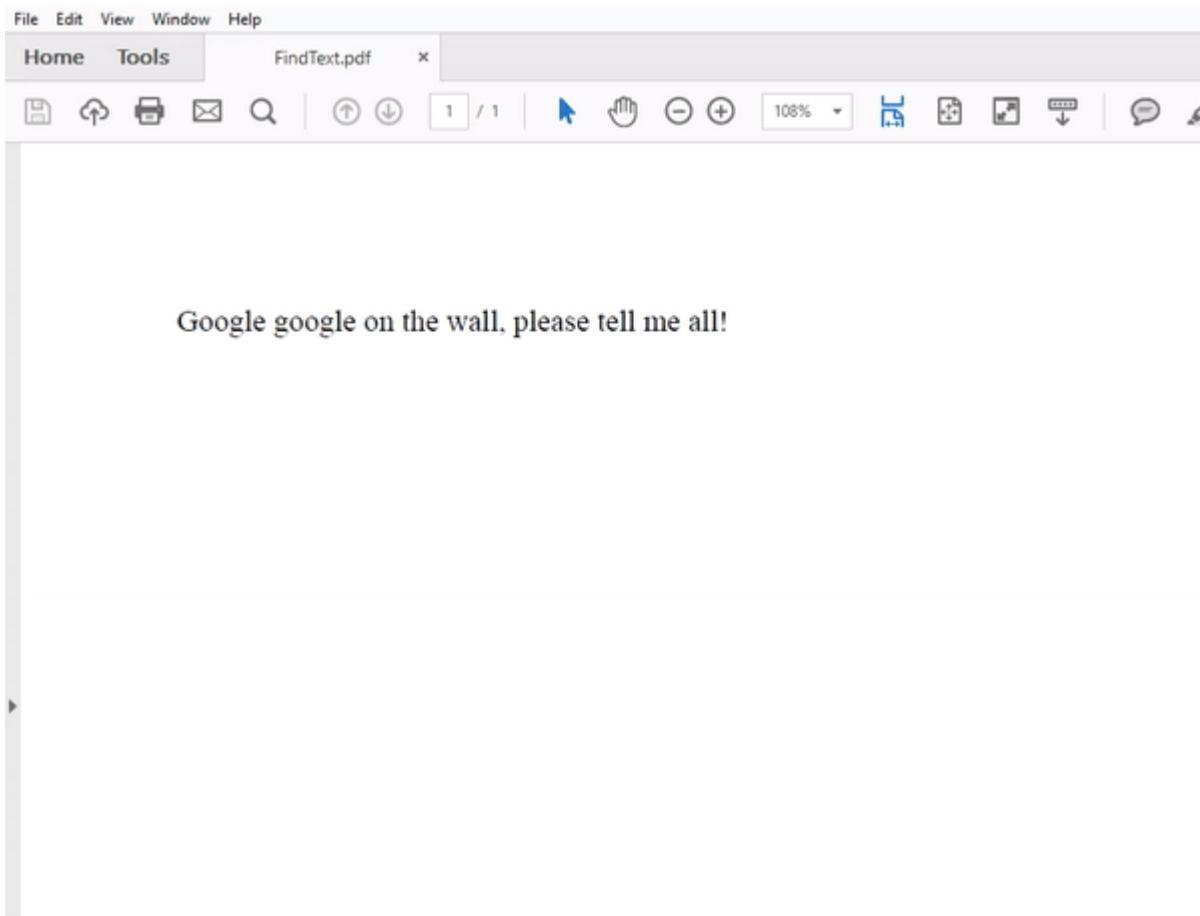
C#

```
var fs = new FileStream(@"../../docu01.pdf", FileMode.Open, FileAccess.Read);  
var pdfDoc = new GcPdfDocument();  
pdfDoc.Load(fs);  
Console.WriteLine($"Linearized: {pdfDoc.Linearized}");
```

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);
}
```

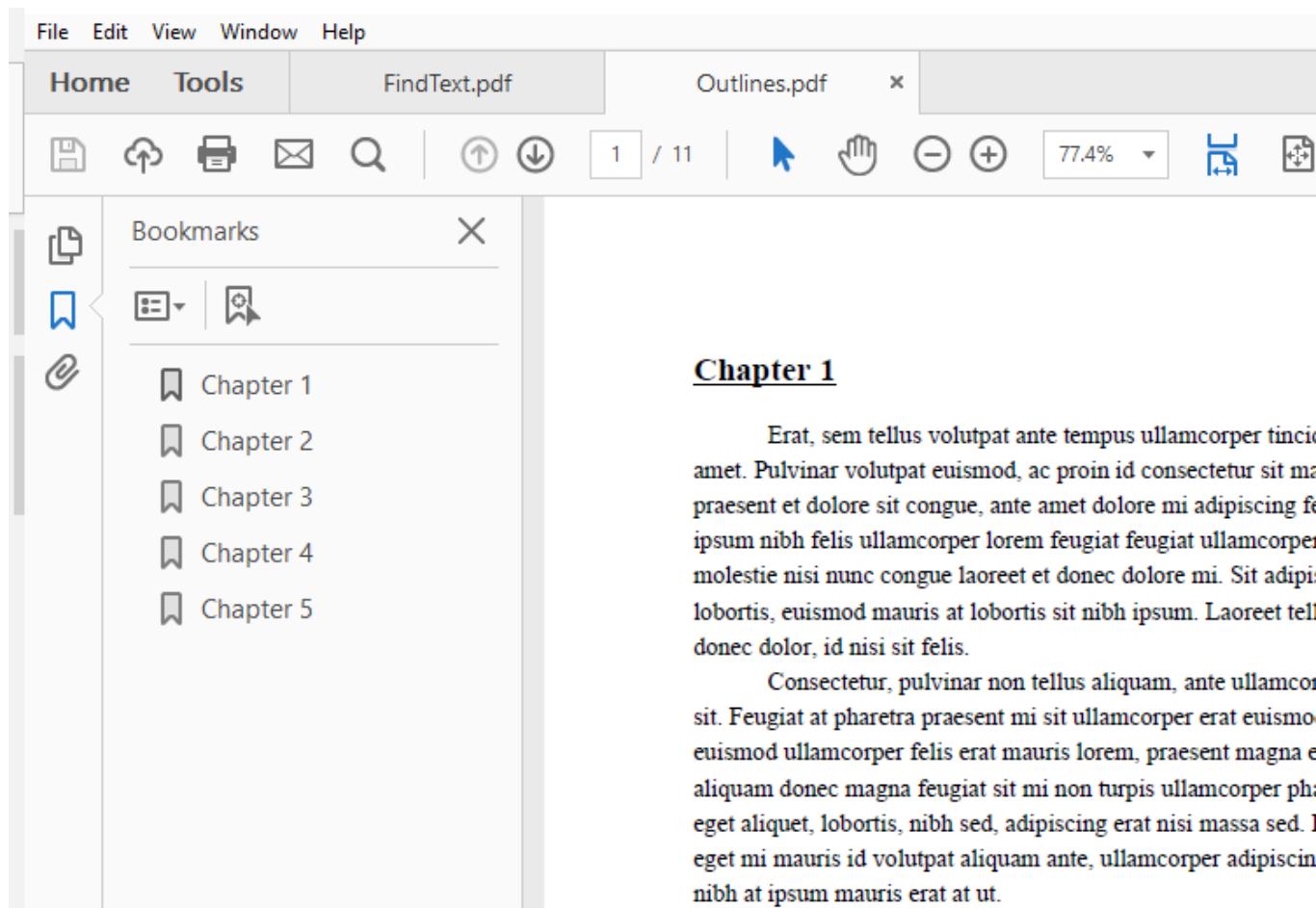
[Back to Top](#)

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 of GcDdfDocument class 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 of GcPdfDocument class.
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 of [GcPdfGraphics](#) class.
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 and setting permission levels 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 using Standard Security Handler Revision 4:

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 EncryptionAlgorithm
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](#)

GcPdf also supports Standard Security Handler Revision 6 (defined in the PDF 2.0 specification) which uses AES encryption with 256 bit key length.

To encrypt a PDF file using Standard Security Handler Revision 6:

1. Create an object of [StandardSecurityHandlerRev6](#) class.
2. Set the required properties of the StandardSecurityHandlerRev6 object, such as password, printing permission, 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 ssh = new StandardSecurityHandlerRev6();
    ssh.OwnerPassword = "password";
    ssh.PrintingPermissions = PrintingPermissions.Enabled;
    // Set the EncryptHandler property
    doc.Security.EncryptHandler = ssh;
    // 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](#)

For more information on applying security using GcPdf, see [GcPdf sample browser](#).

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. You can also add digital signatures with timestamps to mark the time and date of the signature in the PDF document. GcPdf supports legal stamps created by trustworthy authority like the Time Stamp Authority (TSA). GcPdf provides [Sign](#) method to sign and save a document which by default updates the document incrementally. Alternatively, you can also set the **SaveMode** enumeration to **IncrementalUpdate** and pass it as a parameter to [Sign](#) method. Both these methods let 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.

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 of GcPdfDocument class. It also saves the document.

C#

```
public static 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";
    // Add timestamp
    sp.TimeStamp = new TimeStamp("https://freetlsa.org/tsr");

    // 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);

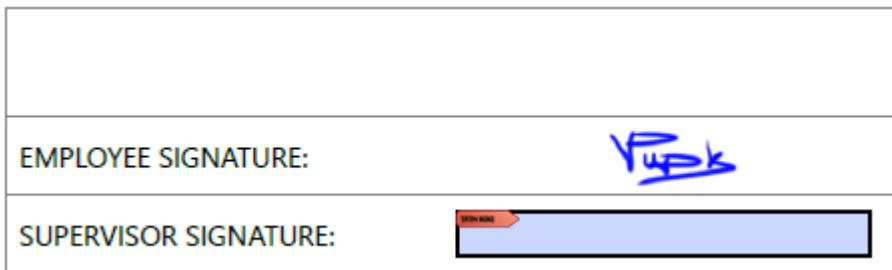
// Rewind the stream to read the document just created
// into another GcPdfDocument and verify the signature:
stream.Seek(0, SeekOrigin.Begin);
GcPdfDocument doc2 = new GcPdfDocument();
doc2.Load(stream);
SignatureField sf2 = (SignatureField)doc2.AcroForm.Fields[0];
if (!sf2.Value.VerifySignature())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

Extract Signature Properties

GcPdf allows you to extract signature information from a digital signature in a PDF document by using **Content** property of **Signature** class. The signature information provides necessary details about the signature which can be used to verify its validity. Some of the information fields which can be extracted from a signature are Issuer, IssuerName, SerialNumber, Subject, Thumbprint, NotAfter, NotBefore, SignatureAlgorithm etc.

To extract signature information from a digital signature in PDF document, follow these steps:

1. Load the signed document and get the signature using **Fields** property of **AcroForm** class.
2. Use the **Content** property of **Signature** class to get the additional information about the signature.

C#

```
MemoryStream ms = new
MemoryStream(File.ReadAllBytes(@"AdobePDFWithEmptySignatureField.pdf"));
GcPdfDocument doc = new GcPdfDocument();
doc.Load(ms);

//initialize a certificate
X509Certificate2 cert = new X509Certificate2(@"User.pfx", "User12");
SignatureProperties sp = new SignatureProperties();
sp.Location = "MACHINE";
sp.SignerName = "USER";
sp.SigningDateTime = null;
sp.SignatureField = doc.AcroForm.Fields["EmptySignatureField"];

using (MemoryStream ms2 = new MemoryStream())
{
```

```
//sign document
doc.Sign(sp, ms2, false);
ms2.Seek(0, SeekOrigin.Begin);

//load signed document
GcPdfDocument doc2 = new GcPdfDocument();
doc2.Load(ms2);

//get signature field and signature
SignatureField sf2 =
(SignatureField)doc2.AcroForm.Fields["EmptySignatureField"];
var sk = sf2.Value.Content;

//get certificate and print its props
var sc = sk.SigningCertificate;
Console.WriteLine($"Subject: {sc.Subject}");
Console.WriteLine($"Issuer: {sc.Issuer}");
Console.WriteLine($"GetEffectiveDateString: {sc.GetEffectiveDateString()}");
Console.WriteLine($"GetExpirationDateString:
{sc.GetExpirationDateString()}");
}
```

[Back to Top](#)

Custom Implementation of Digital Signature

GcPdf provides **ISignatureBuilder** and **IPkcs7SignatureGenerator** interfaces which can be used to achieve the custom implementation of digital signatures. The **Pkcs7SignatureBuilder** class implements the **ISignatureBuilder** interface and provides various methods and properties such as:

- **Pkcs7SignatureBuilder.Format** property which can be set to **SignatureFormat** enumeration values such as **adbe_pkcs7_detached**, **ETSI_CAdES_detached** and **adbe_pkcs7_sha1** to create the respective signatures
- **Pkcs7SignatureBuilder.Crls** property which can be used to embed certificate revocation lists into the signature
- **Pkcs7SignatureBuilder.IncludeOcsp** property which can be used to embed OCPS information into the signature
- **Pkcs7SignatureBuilder.CertificateChain** property which can be used to embed full chain of certificates into the signature

Some of the custom signature implementations are described below:

Sign Document using Certificate from .p12 file

To sign a document using certificate from .p12 file, follow these steps:

1. Instantiate **SignatureProperties** class and use its object to initialize **Pkcs7SignatureBuilder** class.
2. Build a chain of certificates by passing a .p12 filename and its password to **GetCertificateChain** method of **SecurityUtils** class.
3. Add the signature field to the document using **Fields** property of **AcroForm** class.
4. Sign and save the PDF document using **Sign** method of **GcPdfDocument** class.

C#

```
using (FileStream fs = new FileStream(@"AdobePDFWithEmptySignatureField.pdf",
 FileMode.Open))
```

```
{  
    GcPdfDocument doc = new GcPdfDocument();  
    doc.Load(fs);  
  
    SignatureProperties sp = new SignatureProperties();  
    sp.SignatureBuilder = new Pkcs7SignatureBuilder()  
    {  
        CertificateChain = SecurityUtils.GetCertificateChain("1571753451.p12",  
        "test"),  
    };  
    sp.SignatureField = doc.AcroForm.Fields[0];  
    doc.Sign(sp, "signed.pdf");  
}
```

[Back to Top](#)

Sign Document using USB Token

You can sign a document using a USB token with a valid certificate. For details, please refer to this [demo](#).

Sign Document using Certificate from Azure Key Vault

You can sign a document using a certificate stored in Azure Key Vault. For details, please refer to this [demo](#).

PDF Advanced Electronic Signatures

GcPdf lets you digitally sign PDF documents using PDF Advanced Electronic Signatures (PAdES). PAdES is a set of standards referring to a group of extensions and restrictions used when PDF documents are signed electronically. The documents signed using PAdES format remain valid for longer periods.

In PAdES, the following levels of verification of digital signatures are supported by GcPdf:

- B-Level: Indicates that an electronic signature was executed with a signing certificate that was valid on a date.
- T-Level: Similar to B-Level, only adds an additional time-stamp to prove that the signature existed at a certain date and time.
- LT-Level: Building up on T-level, it further adds verification related information to the Documents Security Store(DSS). In GcPdf, DSS is represented by the **DocumentSecurityStore** class.
- LTA-Level: Requires to add time stamp token also to the DSS in addition to the verification related information, thus establishing evidence that the validation data existed at the indicated time.

In GcPdf, you can use **CreatePAdES_B_B** and **CreatePAdES_B_T** methods of **SignatureProperties** class to create B-B and B-T level of signatures in a PDF document. It further provides **GrapeCity.Documents.Pdf.Security.DocumentSecurityStore** class and **GcPdfDocument.TimeStamp()** method to facilitate creation of advanced electronic signatures such as B-LT and B-LTA levels.

Create PAdES B-B Signature

To create a PAdES B-B signature, follow these steps:

1. Initialize a certificate using the **X509Certificate2** class and pass the certificate file name and password to access the certificate.
2. Pass the certificate instance to **CreatePAdES_B_B** method of **SignatureProperties** class to create a PAdES B-B signature.
3. Set the first AcroForm field to store the signature using **SignatureField** property.
4. Sign and save the PDF document using **Sign** method of GcPdfDocument class.

C#

```
using (FileStream fs = new FileStream(@"AdobePDFWithEmptySignatureField.pdf",
    FileMode.Open))
{
    GcPdfDocument doc = new GcPdfDocument();
    doc.Load(fs);

    X509Certificate2 cert = new X509Certificate2("User.pfx", "User12");
    SignatureProperties sp = SignatureProperties.CreatePAdES_B_B(cert);
    sp.SignatureAppearance.Caption = "PAdES B-B";
    sp.SignatureField = doc.AcroForm.Fields[0];
    doc.Sign(sp, "signed_PAdES_B_B.pdf");
}
```

[Back to Top](#)

Create PAdES B-T Signature

To create a PAdES B-T signature, follow these steps:

1. Initialize a certificate using the **X509Certificate2** class and pass the certificate file name and password to access the certificate.
2. Pass the timestamp and certificate instance to **CreatePAdES_B_T** method of **SignatureProperties** class to create a PAdES B-T signature.
3. Set the first AcroForm field to store the signature using **SignatureField** property.
4. Sign and save the PDF document using **Sign** method of **GcPdfDocument** class.

C#

```
using (FileStream fs = new FileStream(@"AdobePDFWithEmptySignatureField.pdf",
    FileMode.Open))
{
    GcPdfDocument doc = new GcPdfDocument();
    doc.Load(fs);

    X509Certificate2 cert = new X509Certificate2("User.pfx", "User12");
    SignatureProperties sp = SignatureProperties.CreatePAdES_B_T(new
TimeStamp("https://freetlsa.org/tsr"), cert);
    sp.SignatureAppearance.Caption = "PAdES B-T";
    sp.SignatureField = doc.AcroForm.Fields[0];
    doc.Sign(sp, "signed_PAdES_B_T.pdf");
}
```

[Back to Top](#)

Create PAdES B-LT Signature

B-LT signature is built on the B-T signature by adding all the properties required for long-term validation of the signature. To create a PAdES B-LT signature, follow these steps:

1. **Create a PAdES B-T signature** and save the PDF document.
2. Add LTV information to the signatures using **AddVerification** method of the **DocumentSecurityStore** class.
3. Sign and save the document in **incremental update** mode using the **Save** method.

C#

```
public int CreatePDF(Stream stream)
{
    var doc = new GcPdfDocument();
    using var s = File.OpenRead(Path.Combine("Resources", "PDFs",
"SignPAdESBT.pdf"));
    doc.Load(s);

    //Add a B-T Level signature
    var pfxPath = Path.Combine("Resources", "Misc", "GcPdfTest.pfx");
    var cert = new X509Certificate2(pfxPath, "qq");
    var sp = SignatureProperties.CreatePAdES_B_T(new
TimeStamp("https://freetlsa.org/tsr"), cert);
    sp.SignatureAppearance.Caption = "PAdES B-LT";
    sp.SignatureField = doc.AcroForm.Fields[0];
    doc.Sign(sp, stream);
    doc.Load(stream);

    // Adds LTV information which makes the signature compliant with PAdES B-LT:
    SignatureField signField = (SignatureField)sp.SignatureField;
    var sig = signField.Value;
    var vp = new DocumentSecurityStore.VerificationParams();
    vp.Certificates = new X509Certificate2[] { s_caCertRoot };
    if (!doc.SecurityStore.AddVerification(sig, vp))
        throw new Exception($"Could not add verification for {sig.Name}.");
    doc.Sign(sp, stream, SaveMode.IncrementalUpdate);

    //Done.
    return doc.Pages.Count;
}
```

Create PAdES B-LTA Signature

B-LTA signature is built on the B-LT signature by adding time stamp token on the validation material. To create a PAdES B-LTA signature, follow these steps:

1. **Create a PAdES B-LT signature** and save the PDF document.
2. Add timestamp to the B-LT signed PDF by using **TimeStampProperties** class to make it compliant with PAdES B-LTA level.
3. Call the **TimeStamp()** method and save the document with time stamp properties.

C#

```
public int CreatePDF(Stream stream)
{
    var doc = new GcPdfDocument();
    using var s = File.OpenRead(Path.Combine("Resources", "PDFs",
"SignPAdESBT.pdf"));
    doc.Load(s);

    //Add a B-T Level signature
    var pfxPath = Path.Combine("Resources", "Misc", "GcPdfTest.pfx");
    var cert = new X509Certificate2(pfxPath, "qq");
```

```
    var sp = SignatureProperties.CreatePAdES_B_T(new  
TimeStamp("https://freetlsa.org/tsr"), cert);  
    sp.SignatureAppearance.Caption = "PAdES B-LTA";  
    sp.SignatureField = doc.AcroForm.Fields[0];  
    doc.Sign(sp, stream);  
    doc.Load(stream);  
  
    // Adds LTV information  
    SignatureField signField = (SignatureField)sp.SignatureField;  
    var sig = signField.Value;  
    var vp = new DocumentSecurityStore.VerificationParams();  
    vp.Certificates = new X509Certificate2[] { s_caCertRoot };  
    if (!doc.SecurityStore.AddVerification(sig, vp))  
        throw new Exception($"Could not add verification for {sig.Name}.");  
    doc.Sign(sp, stream, SaveMode.IncrementalUpdate);  
  
    doc.Load(stream);  
    // Adds time stamp to a signed PDF which makes the document compliant with B-  
    LTA level  
    TimeStampProperties ts = new TimeStampProperties()  
{  
    TimeStamp = new TimeStamp(@"http://ts.ssl.com"),  
};  
// Save the PDF to a file adding a time stamp to it:  
doc.TimeStamp(ts, stream);  
  
// Done.  
return doc.Pages.Count;  
}
```

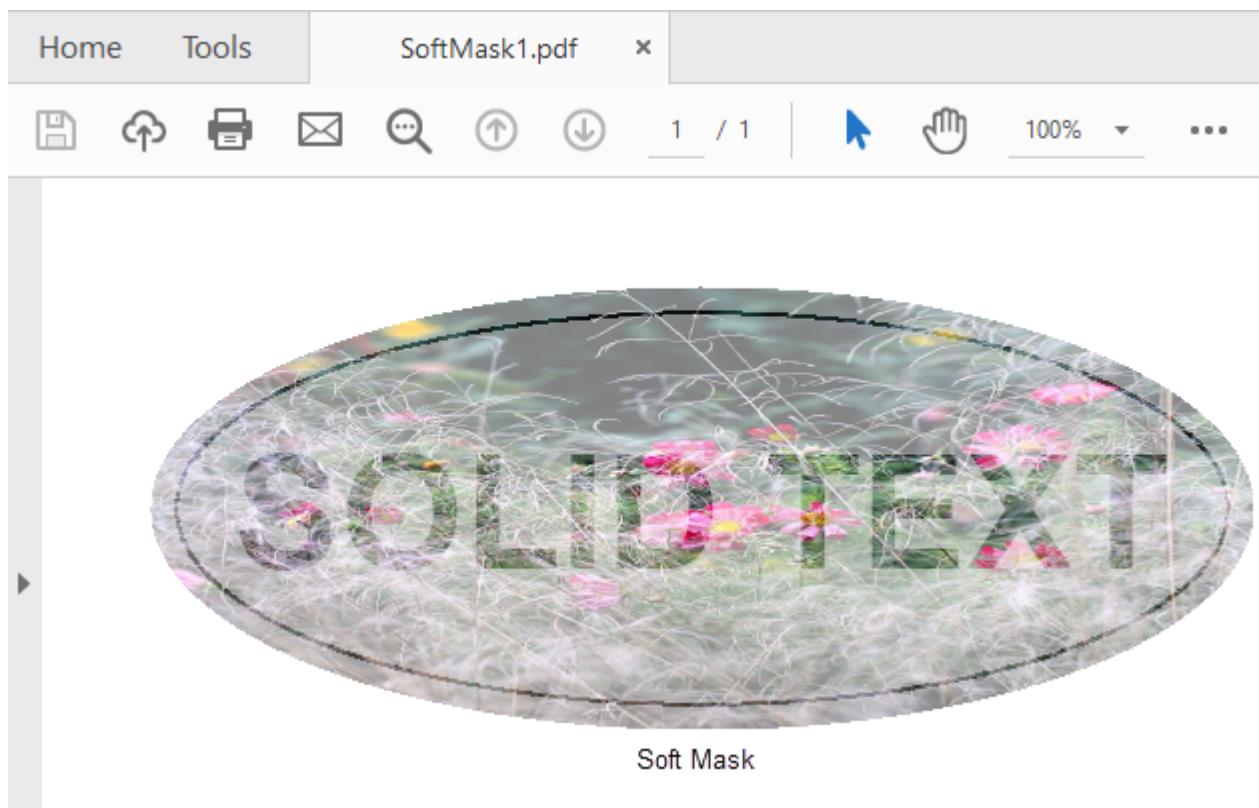
Soft Mask

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 [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'.



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;
}
```

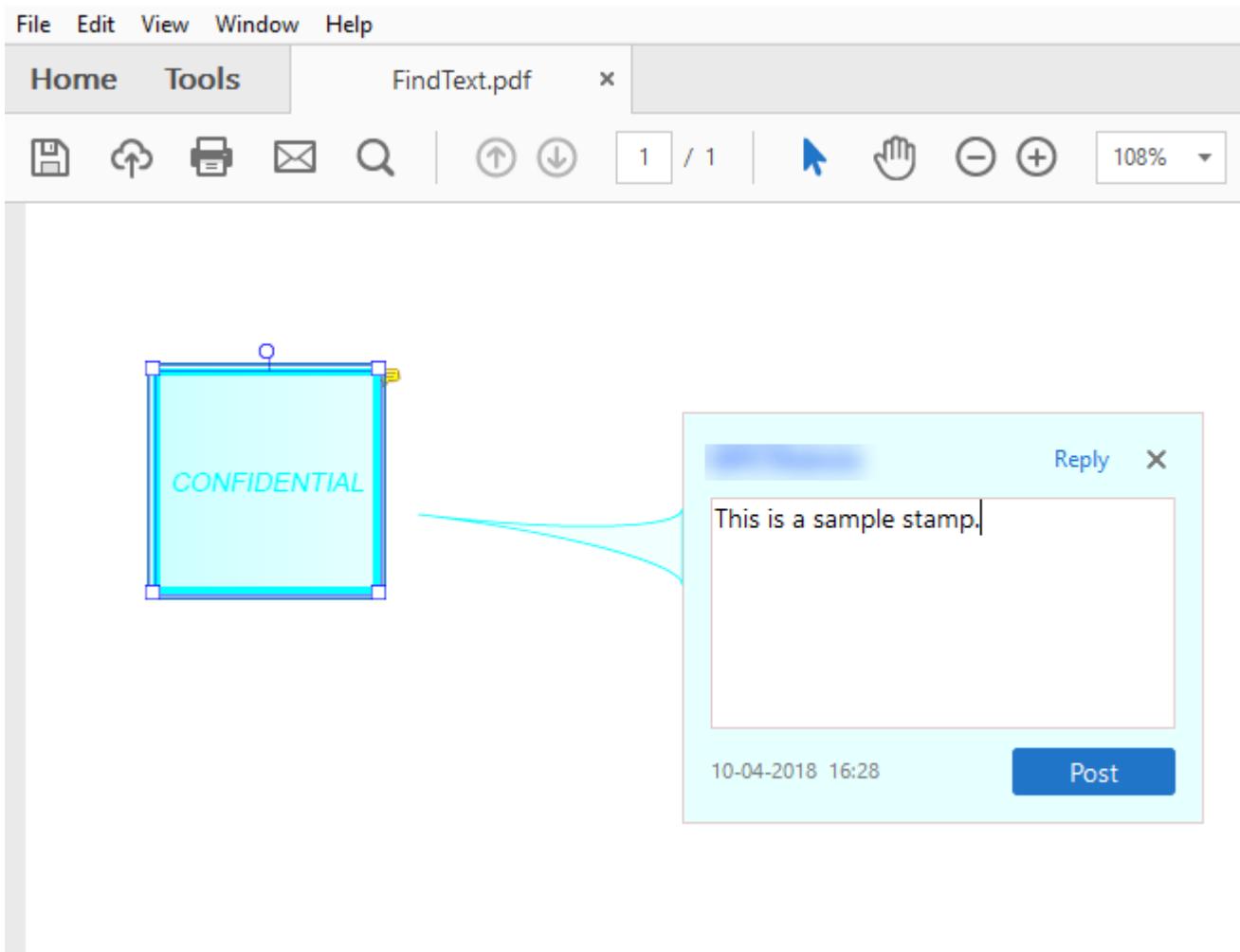
[Back to Top](#)

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,  
    };  
    page.Annotations.Add(stamp);  
    doc.Save(stream);  
}
```

```
    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** property of **AnnotationBase** class 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](#).

Parse PDF Documents

GcPdf allows you to parse PDF documents by recognizing their logical text and document structure. The content elements like plain text, tables, paragraphs and elements in tagged PDF documents can be extracted by using GcPdf API as explained below:

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);  
    t1.AppendLine($"Looked for text inside rectangle x={tx0:F2}\\", y={ty0:F2}\\", \" +  
        $"width={tx1 - tx0:F2}\\", height={ty1 - ty0:F2}\\", found:");  
    t1.AppendLine(text1);  
    t1.AppendLine();  
  
    // Get all text fragments and their locations on the page:  
    t1.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);  
        t1.AppendLine($"Text at ({coords.B.X / 72:F2}\\", {coords.B.Y / 72:F2}\"):\t");  
        t1.AppendLine(tmap.GetText(tlf));  
    }  
    // Print the results:  
    t1.PerformLayout(true);  
}
```

Extract Text Paragraphs

GcPdf allows extracting text paragraphs from a PDF document by using [Paragraphs](#) property of [ITextMap](#) interface. It returns a collection of [ITextParagraph](#) objects associated with the text map.

Sometimes, PDF documents might contain some repeating text (for example, overlap of same text to show it as bold) but GcPdf extracts such text without returning the redundant lines. Also the tables with multi-line text in cells are correctly recognized as text paragraphs.

The example code below shows how to extract all text paragraphs of a PDF document:

C#

```
GcPdfDocument doc = new GcPdfDocument();
var page = doc.NewPage();
var tl = page.Graphics.CreateTextLayout();
tl.MaxWidth = doc.PageSize.Width;
tl.MaxHeight = doc.PageSize.Height;

//Text split options for widow/orphan control
TextSplitOptions to = new TextSplitOptions(tl)
{
    MinLinesInFirstParagraph = 2,
    MinLinesInLastParagraph = 2,
};

//Open a PDF, load it into a temp document and get all page texts
using (var fs=new FileStream("Wetlands.pdf", FileMode.Open, FileAccess.Read))
{
    var doc1 = new GcPdfDocument();
    doc1.Load(fs);

    for (int i = 0; i < doc1.Pages.Count; ++i)
    {
        tl.AppendLine(string.Format("Paragraphs from page {0} of the original PDF:", i + 1));

        var pg = doc1.Pages[i];
        var pars = pg.GetTextMap().Paragraphs;
        foreach (var par in pars)
        {
            tl.AppendLine(par.GetText());
        }
    }

    tl.PerformLayout(true);
    while (true)
    {
        //'rest' will accept the text that did not fit
        var splitResult = tl.Split(to, out TextLayout rest);
        doc.Pages.Last.Graphics.DrawTextLayout(tl, PointF.Empty);
        if (splitResult != SplitResult.Split)
            break;
        tl = rest;
        doc.NewPage();
    }
}
```

```
        }
        //Append the original document for reference
        doc.MergeWithDocument(doc1, new MergeDocumentOptions());
    }
    //Save document
    doc.Save(stream);
    return doc.Pages.Count;
```

Limitations

- The structure elements of a PDF are not taken into account.
- The order of paragraphs can be wrong sometimes, especially in complex cases where there are nested tables etc.
- The text paragraphs found by GetTextMap() cannot span pages, which means that a page break will always break the last paragraph even if logically it is continued on the next page.
- Graphics elements, particularly table borders, are not considered. So, sometimes text in a table layout may be parsed incorrectly.
- In some situations, paragraphs found by GcPdf may not correspond correctly to the logical paragraphs as would be recognized by a human.

Extract Data from Tables

GcPdf allows you to extract data from tables in PDF documents. The [GetTable](#) method in [Page](#) class extracts data from the area specified as a table. The method takes table area as a parameter, parses that area and returns the data of rows, columns, cells and their textual content. You can also pass [TableExtractOptions](#) as a parameter to specify table formatting options like column width, row height, distance between rows or columns.

The example code below shows how to extract data from a table in a PDF document:

C#

```
const float DPI = 72;
const float margin = 36;
var doc = new GcPdfDocument();
var tf = new TextFormat()
{
    Font = Font.FromFile(Path.Combine("segoeui.ttf")),
    FontSize = 9,
    ForeColor = Color.Black
};

var tfRed = new TextFormat(tf) { ForeColor = Color.Red };
var fs = File.OpenRead(Path.Combine("zugferd-invoice.pdf"));
{
    // The approx table bounds:
    var tableBounds = new RectangleF(0, 3 * DPI, 8.5f * DPI, 3.75f * DPI);

    var page = doc.NewPage();
    page.Landscape = true;
    var g = page.Graphics;

    var tl = g.CreateTextLayout();
    tl.MaxWidth = page.Bounds.Width;
    tl.MaxHeight = page.Bounds.Height;
```

```
tl.MarginAll = margin;
tl.DefaultTabStops = 150;
tl.LineSpacingScaleFactor = 1.2f;

var docSrc = new GcPdfDocument();
docSrc.Load(fs);

var itable = docSrc.Pages[0].GetTable(tableBounds);

if (itable == null)
{
    tl.AppendLine($"No table was found at the specified coordinates.", tfRed);
}
else
{
    tl.Append($"\\nThe table has {itable.Cols.Count} column(s) and
{itable.Rows.Count} row(s), table data is:", tf);
    tl.AppendParagraphBreak();
    for (int row = 0; row < itable.Rows.Count; ++row)
    {
        var tfmt = row == 0 ? tf : tf;
        for (int col = 0; col < itable.Cols.Count; ++col)
        {
            var cell = itable.GetCell(row, col);
            if (col > 0)
                tl.Append("\\t", tfmt);
            if (cell == null)
                tl.Append("<no cell>", tfRed);
            else
                tl.Append(cell.Text, tfmt);
        }
        tl.AppendLine();
    }
}
TextSplitOptions to = new TextSplitOptions(tl) { RestMarginTop = margin,
MinLinesInFirstParagraph = 2, MinLinesInLastParagraph = 2 };
tl.PerformLayout(true);
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;
    doc.NewPage().Landscape = true;
}
// Append the original document for reference
doc.MergeWithDocument(docSrc);
doc.Save(stream);
```

 **Note:** The font files used in the above sample can be downloaded from [Get Table Data demo](#).

Limitation

- Tables cannot be searched automatically in a PDF document. Their area needs to be specified.

Extract Content from Tagged PDF

GcPdf can recognize the logical structure of a source document from which the PDF document is generated. This structure recognition is further used to extract content elements from tagged PDF documents.

Based on the PDF specification, GcPdf recognizes the logical structure by using **LogicalStructure** class. It represents a parsed logical structure of a PDF document which is created on the basis of tags in the PDF structure tree. The **StructElement** property of **Element** class can be used to get the element type, such as TR for table row, H for headings, P for paragraphs etc.

The example code below shows how to extract headings, tables and TOC elements from a tagged PDF document:

C#

```
static void ShowTable(Element e)
{
    List<List<IList<ITextParagraph>>> table = new List<List<IList<ITextParagraph>>>();
    // select all nested rows, elements with type TR
    void SelectRows(IList<Element> elements)
    {
        foreach (Element ec in elements)
        {
            if (ec.HasChildren)
            {
                if (ec.StructElement.Type == "TR")
                {
                    var cells = ec.Children.FindAll((e_) => e_.StructElement.Type ==
"TD").ToArray();
                    List<IList<ITextParagraph>> tableCells = new
List<IList<ITextParagraph>>();
                    foreach (var cell in cells)
                        tableCells.Add(cell.GetParagraphs());
                    table.Add(tableCells);
                }
                else
                    SelectRows(ec.Children);
            }
        }
    }
    SelectRows(e.Children);

    // show table
    int colCount = table.Max((r_) => r_.Count);
    Console.WriteLine();
    Console.WriteLine();
    Console.WriteLine($"Table: {table.Count}x{colCount}");
    Console.WriteLine($"-----");
    foreach (var r in table)
```

```
{  
    foreach (var c in r)  
    {  
        var s = c == null || c.Count <= 0 ? string.Empty : c[0].GetText();  
        Console.WriteLine(s);  
        Console.WriteLine("\t");  
    }  
    Console.WriteLine();  
}  
  
}  
  
static void Main(string[] args)  
{  
  
    GcPdfDocument doc = new GcPdfDocument();  
  
    using (var s = new FileStream("C1Olap QuickStart.pdf", FileMode.Open,  
        FileAccess.Read, FileShare.Read))  
    {  
        doc.Load(s);  
  
        // get the LogicalStructure and top parent element  
        LogicalStructure ls = doc.GetLogicalStructure();  
        Element root = ls.Elements[0];  
  
        // select all headings  
        Console.WriteLine("TOC:");  
        Console.WriteLine("----");  
        // iterate over elements and select all heading elements  
        foreach (Element e in root.Children)  
        {  
            string type = e.StructElement.Type;  
            if (string.IsNullOrEmpty(type) || !type.StartsWith("H"))  
                continue;  
            int headingLevel;  
            if (!int.TryParse(type.Substring(1), out headingLevel))  
                continue;  
            // get the element text  
            string text = e.GetText();  
            if (string.IsNullOrEmpty(text))  
                text = "H" + headingLevel.ToString();  
            text = new string(' ', (headingLevel - 1) * 2) + text;  
            Console.WriteLine(text);  
  
        }  
  
        // select all tables  
        var tables = root.Children.FindAll(e_ => e_.StructElement.Type ==  
            "Table").ToArray();  
        foreach (var t in tables)  
        {  
    }
```

```
        ShowTable(t);
    }
}
}
```

The example code below shows how to extract all paragraphs from a PDF document and save them to a Word document:

C#

```
// restore word document from pdf
using (var s = new FileStream("CharacterFormatting.pdf", FileMode.Open,
FileAccess.Read, FileShare.Read))
{
    doc.Load(s);

    // get the LogicalStructure and top parent element
    LogicalStructure ls = doc.GetLogicalStructure();
    Element root = ls.Elements[0];

    GcWordDocument wdoc = new GcWordDocument();

    // iterate over elements and select all paragraphs
    foreach (Element e in root.Children)
    {
        if (e.StructElement.Type != "P")
            continue;
        var tps = e.GetParagraphs();
        if (tps == null)
            continue;

        foreach (var tp in tps)
        {
            // build a Word paragraph from a ITextParagraph
            Paragraph p = wdoc.Body.Paragraphs.Add();
            foreach (var tr in tp.Runs)
            {
                var range = p.GetRange();
                var run = range.Runs.Add(tr.GetText());
                run.Font.Size = tr.Attrs.FontSize;
                if (tr.Attrs.NonstrokeColor.HasValue)
                    run.Font.Color.RGB = tr.Attrs.NonstrokeColor.Value;

                tr.Attrs.Font.GetFontAttributes(out string fontFamily,
                    out FontWeight? fontWeight,
                    out FontStretch? fontStretch,
                    out bool? fontItalic);
                if (!string.IsNullOrEmpty(fontFamily))
                    run.Font.Name = fontFamily;
                if (fontWeight.HasValue)
                    run.Font.Bold = fontWeight.Value >= FontWeight.Bold;
                if (fontItalic.HasValue)
                    run.Font.Italic = fontItalic.Value;
            }
        }
    }
}
```

```
        }
    }
}

wdoc.Save("CharacterFormatting.docx");
}
```

Refer to [Tagged PDF](#) to know how to create tagged PDF files using GcPdf.

Layers

The layers (optional content) in a PDF document allow you to selectively view or hide the specific content sections. The main purpose of layers is to control the visibility of graphics objects rendered in a PDF document. A few examples where PDF layers can be particularly useful, are:

- PDF document containing multi-lingual content, each in different layers.
- PDF document containing animation that appears on a separate layer.
- License agreement in a PDF document which needs to be agreed upon before the content can be viewed.
- PDF document containing a watermark, which may not show onscreen but is always printed and exported to other applications.
- PDF document containing details over a product design.

GcPdf allows you to create layers, associate content (such as content stream, FormXObject and annotation) with different layers and set their properties. The **OptionalContentProperties** class provides various methods and properties which can be used to implement layers in PDF documents.

Create Layers in a PDF document

To create layers in a PDF and draw on them:

1. Create and name layers by using the **AddLayer** method of **OptionalContentProperties** class.
2. To begin drawing on a specific layer, call the **BeginLayer** method of GcPdfGraphics, specifying the target layer's name.
3. Add content to the layer by using any of the GcPdfGraphics' Draw* methods (e.g. DrawString).
4. To end drawing on the current layer, call the **EndLayer** method of GcPdfGraphics.
5. Similarly, you can add content to other layers by using the BeginLayer/EndLayer methods and drawing on those other layers.

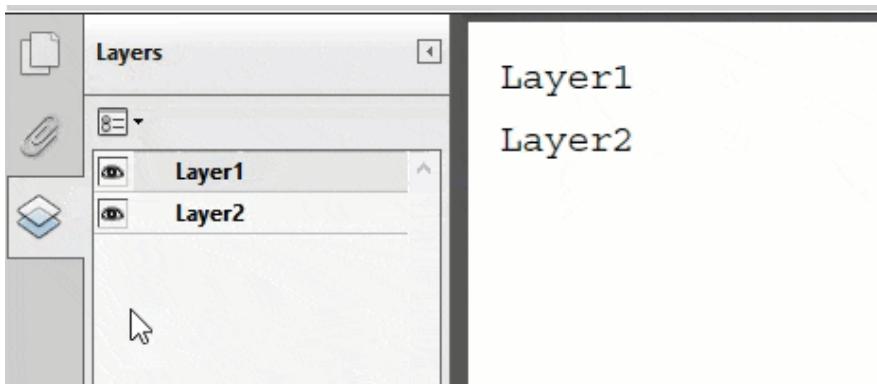
C#

```
GcPdfDocument doc = new GcPdfDocument();
doc.OptionalContent.AddLayer("Layer1");
doc.OptionalContent.AddLayer("Layer2");

var g = doc.NewPage().Graphics;
g.BeginLayer("Layer1");
g.DrawString("Layer1", new TextFormat() { Font = StandardFonts.Courier }, new
PointF(10, 10));
g.EndLayer();

g.BeginLayer("Layer2");
g.DrawString("Layer2", new TextFormat() { Font = StandardFonts.Courier }, new
PointF(10, 30));
g.EndLayer();
doc.Save("SimpleLayers.pdf");
```

The output of above code example will look like:



Create Layers and Associate FormXObject

A FormXObject can also be associated with a specific layer. The following code example demonstrates this:

1. Create layer or layers as described above, storing the reference to the created layer object returned by the **AddLayer** method.
2. Create a FormXObject (see [FormXObject](#)).
3. Set the **Layer** property of the **FormXObject** to the layer you want the FormXObject to be associated with.

C#

```
GcPdfDocument doc = new GcPdfDocument();
var l1 = doc.OptionalContent.AddLayer("Layer1");
var l2 = doc.OptionalContent.AddLayer("Layer2");

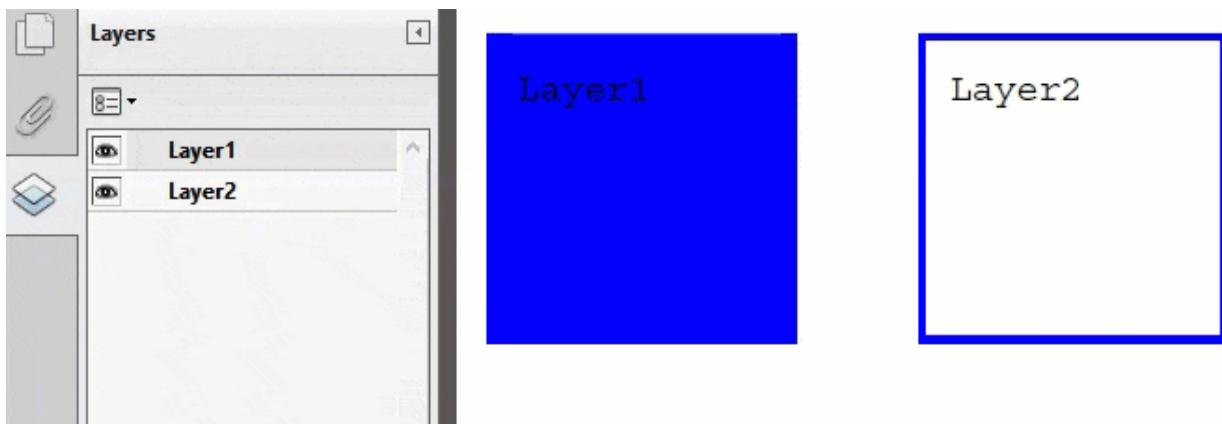
var g = doc.NewPage().Graphics;

FormXObject fxo1 = new FormXObject(doc, new RectangleF(0, 0, 100, 100));
fxo1.Graphics.FillRectangle(new RectangleF(0, 0, 100, 100), Color.Blue);
fxo1.Graphics.DrawString(l1.Name, new TextFormat() { Font = StandardFonts.Courier },
new PointF(10, 10));
fxo1.Layer = l1;

FormXObject fxo2 = new FormXObject(doc, new RectangleF(0, 0, 100, 100));
fxo2.Graphics.DrawRectangle(new RectangleF(0, 0, 100, 100), Color.Blue, 4);
fxo2.Graphics.DrawString(l2.Name, new TextFormat() { Font = StandardFonts.Courier },
new PointF(10, 10));
fxo2.Layer = l2;

g.DrawForm(fxo1, new RectangleF(10, 10, 100, 100), null, ImageAlign.StretchImage);
g.DrawForm(fxo2, new RectangleF(150, 10, 100, 100), null, ImageAlign.StretchImage);
doc.Save("FormXObjectLayers.pdf");
```

The output of above code example will look like:



Create Layers and Associate Annotation

An annotation can also be associated with a specific layer. The below example code demonstrates this by setting the **Layer** property of the **AnnotationBase** class to the desired layer.

C#

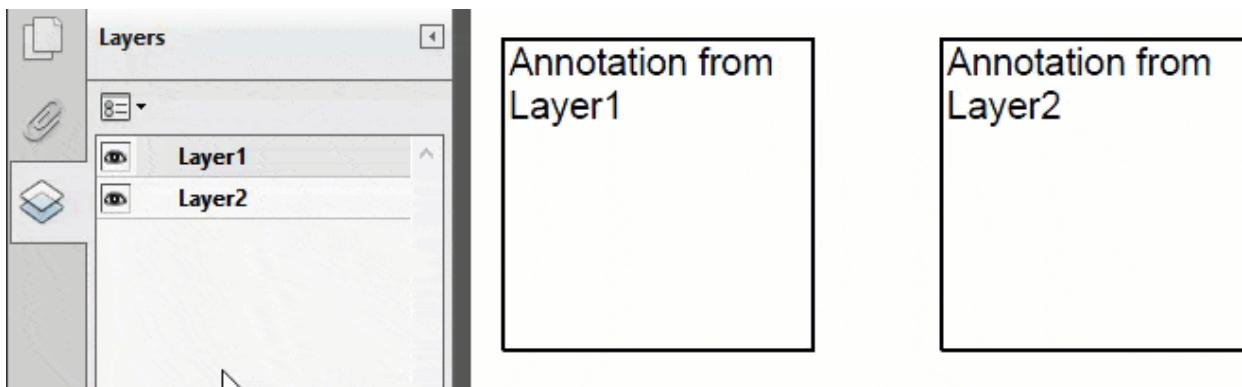
```
GcPdfDocument doc = new GcPdfDocument();
var l1 = doc.OptionalContent.AddLayer("Layer1");
var l2 = doc.OptionalContent.AddLayer("Layer2");

var p = doc.NewPage();

var ft = new FreeTextAnnotation();
ft.UserName = "UserName";
ft.Rect = new RectangleF(10, 10, 100, 100);
ft.Contents = $"Annotation from {l1.Name}";
ft.Layer = l1;
p.Annotations.Add(ft);

ft = new FreeTextAnnotation();
ft.UserName = "UserName";
ft.Rect = new RectangleF(150, 10, 100, 100);
ft.Contents = $"Annotation from {l2.Name}";
ft.Layer = l2;
p.Annotations.Add(ft);
doc.Save("AnnotationLayers.pdf");
```

The output of above code example will look like:



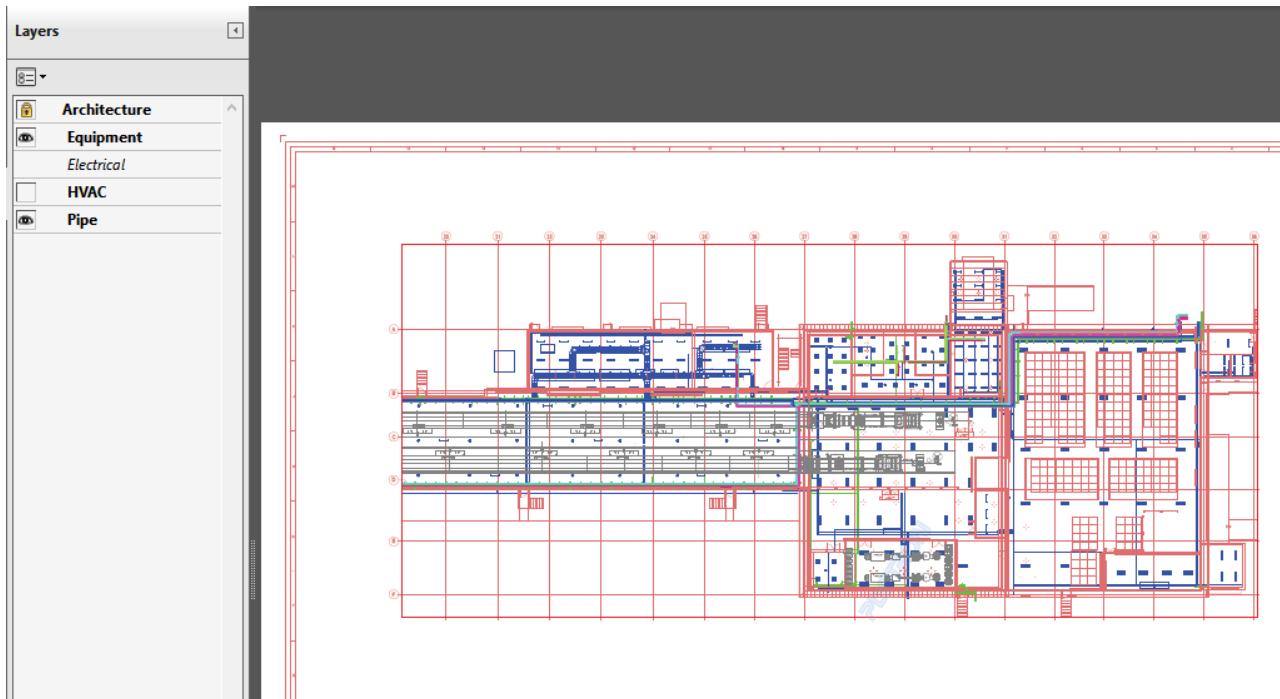
Set Layer Properties

You can use the Set* utility methods of the **OptionalContentProperties** class to change the various properties of different layers. The below example code demonstrates this:

C#

```
GcPdfDocument doc = new GcPdfDocument();
using (FileStream fs = new FileStream("construction_drawing-final.pdf", FileMode.Open))
{
    doc.Load(fs);
    doc.OptionalContent.SetLayerLocked("Architecture", true);
    doc.OptionalContent.SetLayerInitialState("Equipment",
    LayerInitialViewState.VisibleWhenOn);
    doc.OptionalContent.Groups.FindByName("Electrical").Intent = new string[] { "Design" };
}
doc.OptionalContent.SetLayerDefaultState("HVAC", false);
doc.OptionalContent.SetLayerPrintState("Pipe", LayerPrintState.Never);
doc.Save("SetLayerProperties.pdf");
}
```

The output of above code example will look like:



Note: GcDocs PDF Viewer lets you enable the layers panel which can be used to hide or display the PDF layers. For more information, see [Enable Layers Panel](#).

Perform PDF Operations on Layer

In GcPdf, you can perform a particular PDF operation such as searching text, extracting text or drawing annotation on the specified layer. This can be achieved by using the **ViewState** class which allows to define environment settings and fetches the state of layer in the environment. To perform an operation on a specific layer, you can pass an instance of this class to method corresponding to that operation.

For instance, following sample code shows how to search a text in the specified layer of PDF document.

C#

```
void FindInLayer(GcPdfDocument doc, string layer, string text, Color highlight)
{
    // Create a view state with just the specified layer visible:
    var ViewState = new ViewState(doc);
    ViewState.SetLayersUIStateExcept(false, layer);
    ViewState.SetLayersUIState(true, layer);

    // Create a FindTextParams using our custom view state
    // so that the search is limited to the specified layer only:
    var ftp = new FindTextParams(text, false, false, ViewState, 72f, 72f, true, true);

    // Find all occurrences of the search text:
    var finds = doc.FindText(ftp, OutputRange.All);

    // Highlight all occurrences on the specified layer only:
    foreach (var find in finds)
        foreach (var ql in find.Bounds)
    {
        var g = doc.Pages[findPageIndex].Graphics;
        g.BeginLayer(layer);
        doc.Pages[findPageIndex].Graphics.FillPolygon(ql, highlight);
        g.EndLayer();
    }
}
```

Remove Layer and Associated Content

While dealing with multi-layer PDF documents, sometimes you might want to choose whether or not to delete the associated content while removing a layer. GcPdf provides **OptionalContent.RemoveLayer** method which lets you choose whether to remove content associated with the layer while removing it. The method accepts **removeContent** property and array of the **OptionalContentGroup** objects as its parameters. To remove the content associated to a layer, you can also use **Page.RemoveLayersContent** method.

C#

```
// Remove all layers except the last one with their content:
var layers = doc.OptionalContent.Groups.Take(doc.OptionalContent.Groups.Count - 1).ToArray();
doc.OptionalContent.RemoveLayers(true, layers);
// Remove the single remaining layer, leaving its content in place:
doc.OptionalContent.RemoveLayer(doc.OptionalContent.Groups[0]);
```

Limitation

GcPdf does not support layers when rendering the PDF document, meaning that there is no ability to hide layers. The complete content is rendered regardless of whether it belongs to any layer or not.

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

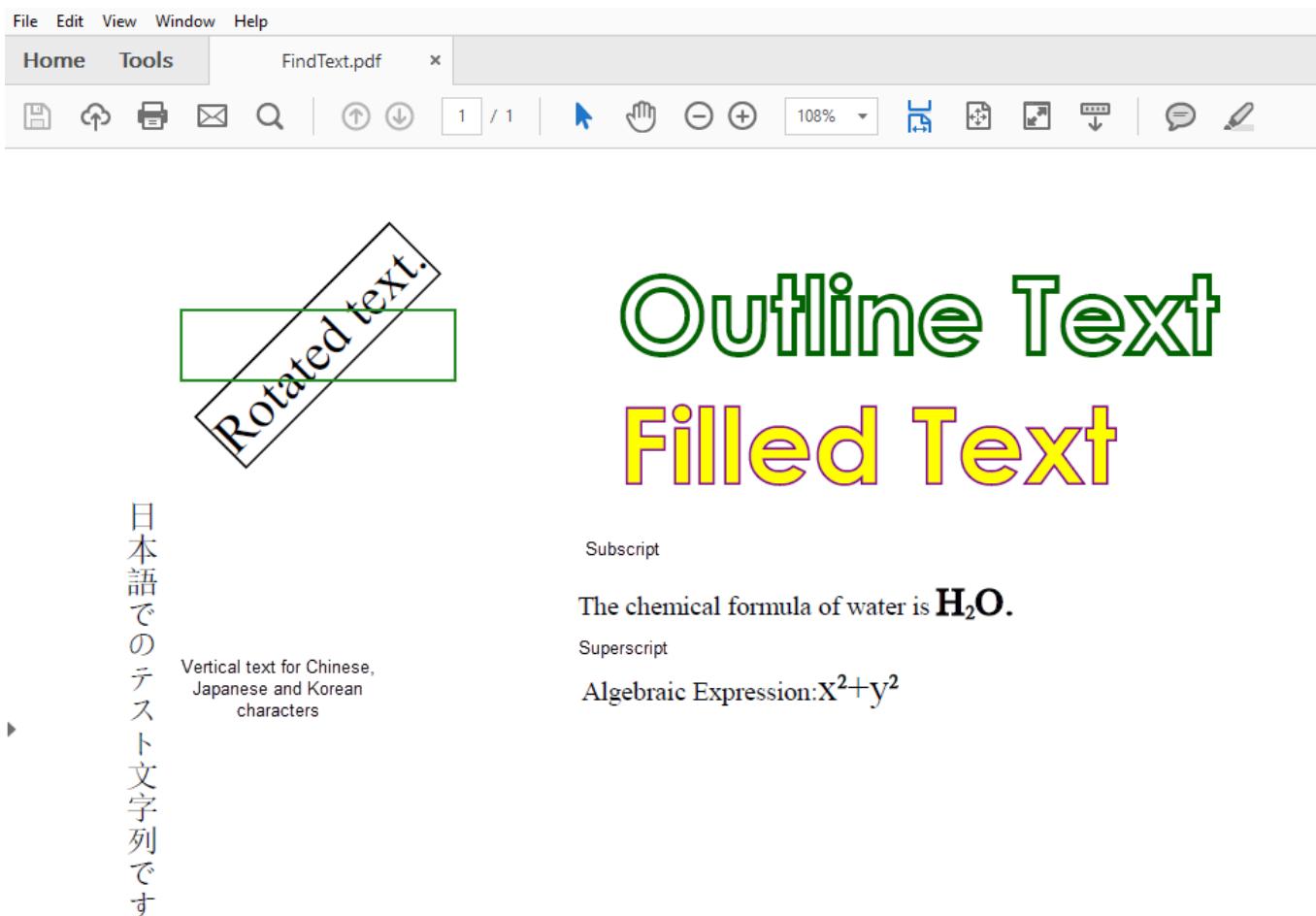
GcPdf offers [TrimmingGranularity](#) and [WrapMode](#) properties of the [TextLayout](#) class that allows trimming the text at word or character, and wrap text in a PDF document. This allows you to display ellipsis or any other character at the end of a text line that does not fit in the allocated 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 H₂O) and [Superscript](#) property to set text as superscript (for example, "3" in x³).

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](#)

For more information about rendering text using GcPdf, see [GcPdf sample browser](#).

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("日本語でのテキスト文字列です", 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 tfsup = 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", tfsup);  
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);

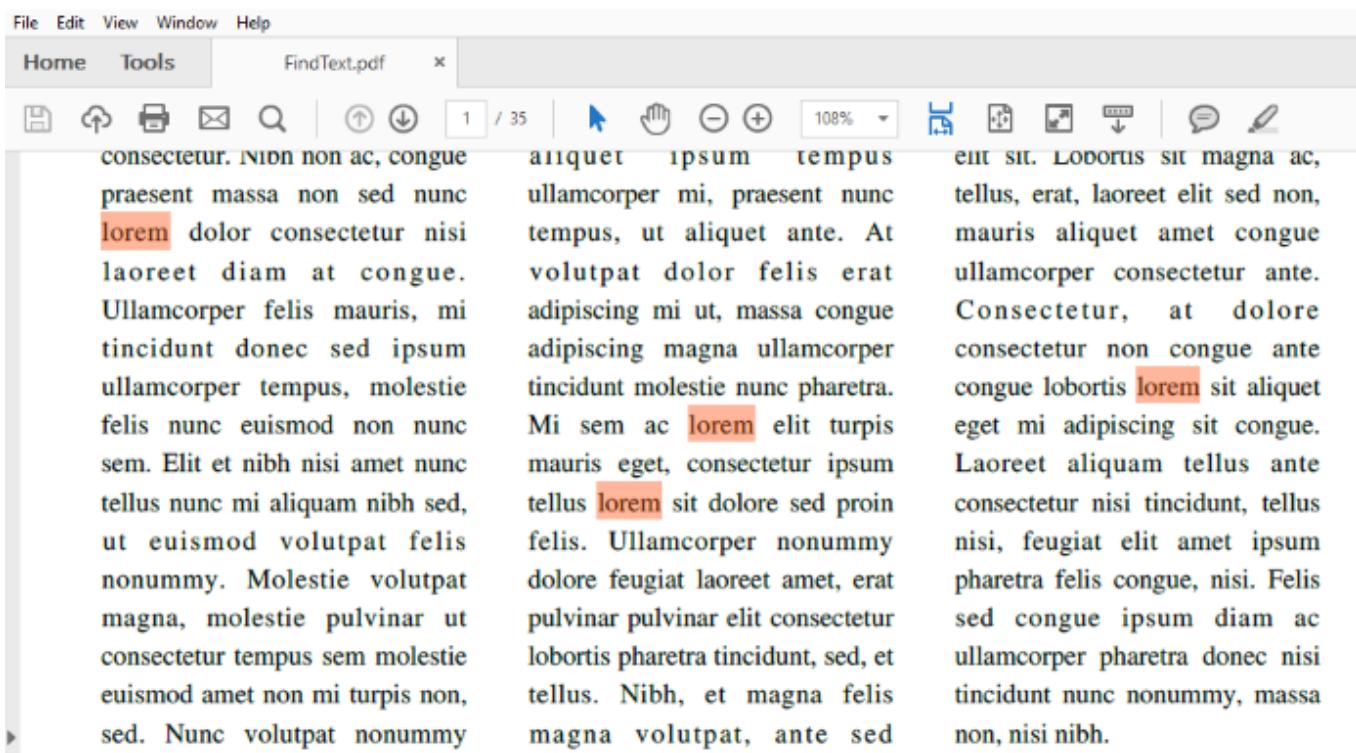
// Set first line offset
tl.FirstLineIndent = 72 / 2;
// Set line spacing
tl.LineSpacingScaleFactor = 1.5f;
tl.PerformLayout(true);
g.DrawTextLayout(tl, ip);
// Save document
doc.Save(stream);
}
```

Back to Top

For more information about extracting table data from PDF documents by using GcPdf, see [GcPdf sample browser](#).

Text Search

GcPdf allows text search in a PDF document to find all occurrences of the specified text and highlights them. It also works across line breaks, so logically connected text that is rendered on different text lines can also be found. The [FindText](#) method of [GcPdfDocument](#) class can be used for the same. 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 of GcPdfDocument class 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[findPageIndex].Graphics.FillPolygon  
(find.Bounds[0], Color.FromArgb(100, Color.OrangeRed));  
    }  
}
```

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

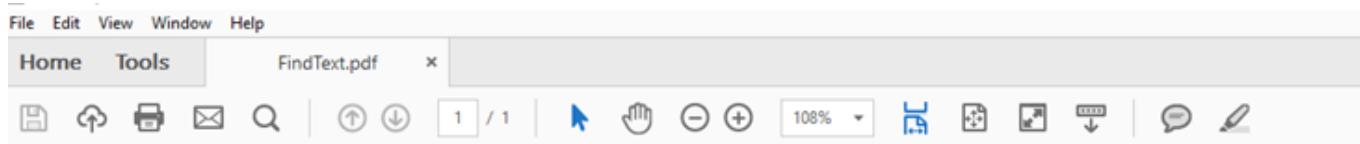
[Back to Top](#)

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](#)

Print

Printing a PDF document is a basic and one of the most commonly used feature. In most cases, printing is supported through a PDF Viewer, that is, you can open your PDF document in a viewer and give a print document from viewer itself. However, in some cases you might want to print a document directly without using a viewer. In case of macOS and Linux, operating systems provide built-in print functionality and can print simply by using a command line. However in case of Windows, there is no such built-in feature available. Hence, we have created a product sample that uses Direct2D technology to print the document without using a viewer.

Print on Windows OS

To demonstrate direct printing through GcPdf on Windows operating system, a sample named "Print PDF on Windows using Direct2D" is hosted as a part of the [online demo sample](#). The sample has been built up on **GcD2DBitmap** and **GcD2DGraphics** classes which reside in **GrapeCity.Documents.Imaging.Windows** package.

The sample includes following ready to use utilities required to print a document and implement print settings. These

can be used as it is in your application.

- **GcPdfDocumentPrintExt**, a static class which provides extension methods to print a GcPdfDocument object.
- **GcPdfPrintManager** class implements printing services used by the GcPdfDocumentPrintExt class.
- **PageScaling** enumeration specifies how pages are scaled when printed.

The online demo sample has been designed to create a single-page PDF and return it. The actual code for printing the generated PDF on a local printer has been blocked inside a false condition as shown in the code below.

C#

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

    Common.Util.AddNote(
        "This sample uses Direct2D to implement direct printing on Windows. ",
        doc.NewPage());

    // Include this block to print the PDF on Windows:
    if (false)
    {
        GcPdfPrintManager pm = new GcPdfPrintManager();
        pm.Doc = doc;
        pm.PrinterSettings = new PrinterSettings();
        // Set the printer name and/or other settings if not using the defaults.
        // pm.PrinterSettings.PrinterName = "my printer";
        pm.PageScaling = PageScaling.FitToPrintableArea;
        pm.Print();
    }

    // Save the PDF:
    doc.Save(stream);
}
```

To print a PDF, you need to download the demo sample, edit the source to remove the false condition and call the GcPdfPrintManager.Print() method. You can adjust the GcPdfPrintManager and PrinterSettings properties to get the desired output.

Print on Linux and macOS

Command line to print a PDF document on Linux:

```
lp filename.pdf
```

Command line to print a PDF document on macOS:

```
lp filename.pdf
```

OR

```
#!/bin/bash
cd /path/to/your/PDFfiles
for pdffile in *.pdf;
lpr -P MY_PRINTER -o media=A4 -o sides=two-sided-long-edge -o InputSlot=tray-3
"$pdffile";
```

done

Render HTML to PDF

GcPdf library along with **GcHtml** library, lets you easily render HTML content to PDF document. With a utility library like GcHtml, you can convert HTML files like invoices and reports to PDF documents and print them without worrying about disarranged layouts, styles or formats. You can also add HTML content to PDF documents.

GcHtml is based on the industry standard Chrome web browser engine working in headless mode, offering advantage of rendering HTML to PDF on any platform - Windows, Linux and macOS. Currently, GcHtml works only on Intel-based 64-bit processor architecture. It doesn't matter whether your .NET application is built for x64, x86 or AnyCPU platform target. The browser is always working in a separate process.

The GcHtml library consists of platform dependent and platform independent NuGet packages. You can add a platform-dependent Html package depending on the platform you are working. Alternatively, you can add all the Html packages and let GcHtml automatically select the correct package at runtime.

GcHtml NuGet Packages	Description
GrapeCity.Documents.Html	<p>The GrapeCity.Documents.Html package contains the following namespaces:</p> <ul style="list-style-type: none">• GrapeCity.Documents.Html namespace provides GcHtmlRenderer, PdfSettings, ImageSettings, PngSettings classes etc.• GrapeCity.Documents.Pdf namespace provides the GcPdfGraphicsExt and HtmlToPdfFormat classes.• GrapeCity.Documents.Drawing namespace provides GcBitmapGraphicsExt and HtmlToImageFormat class.
GrapeCity.Documents.Html.Windows.X64	<p>It is the interface unit and Chromium browser engine for 64-bit Windows platform.</p> <p>Note: Due to Azure Windows AppService and Azure Functions limitations, GcHtml is not supported in these environments.</p>
GrapeCity.Documents.Html.Mac.X64	<p>It is the interface unit and browser engine for macOS platform.</p>
GrapeCity.Documents.Html.Linux.X64	<p>It is the interface unit and browser engine for 64-bit Linux platform.</p> <p>Note that to use GcHtml on Linux, Chromium dependencies must be installed. The following command installs the necessary packages on Ubuntu:</p> <div data-bbox="634 1617 1460 1729" style="background-color: #f0f0f0; padding: 10px;"><pre>sudo apt-get update sudo apt-get install libxss1 libappindicator1 libindicator7 libnss3-dev</pre></div>

Install GcHtml Package

Refer the steps below to install GcHtml package in your project:

1. Open Visual Studio and create a .Net Core Console application.
2. Right-click **Dependencies** and select **Manage NuGet Packages**.
3. With the **Package source** set to Nuget website, search for GrapeCity.Documents.Pdf under the **Browse** tab and

click **Install**.

- Similarly, install GrapeCity.Documents.Html and GrapeCity.Documents.Html.Windows.X64 (depending on the platform you are working).

 **Note:** During installation, you'll receive two confirmation dialogs. Click **OK** in the **Preview Changes** dialog box and click **I Agree** in the **License Acceptance** dialog box to proceed installation.

- Once, the GcHtml package has been installed successfully, add the namespace in Program.cs file.

C#

```
using GrapeCity.Documents.Html;
using GrapeCity.Documents.Pdf;
using GrapeCity.Documents.Drawing;
```

- Apply GcPdf license to **GcHtmlRenderer** class of GcHtml library to convert HTML to PDF. Without proper license, the count is limited to only 5 PDF conversions. The license can be applied in one of the following ways as shown below:

- To license the instance being created

```
var uri = new Uri(@"https://www.grapecity.com/controls/documents-pdf/whats-new");
var render = new GcHtmlRenderer(uri);
var re= new GcHtmlRenderer();
re.ApplyGcPdfLicenseKey("key");
```

- To license all the instances

```
GcHtmlRenderer.SetGcPdfLicenseKey("key");
```

- Write the sample code.

Render HTML Webpage to PDF Document

GcHtml can render HTML webpage to PDF document. GcPdf provides the **PdfSettings** class and **RenderToPdf** method of **GcHtmlRenderer** class to render HTML files to PDF documents.

To render the HTML webpage to a PDF document, follow the steps below:

- Specify the PDF file path for rendering HTML webpage.
- Specify the HTML source (URI).
- Define the PDF document settings using the **PdfSettings** class.
- Convert the HTML webpage to PDF file using **RenderToPdf** method of **GcHtmlRenderer** class.

C#

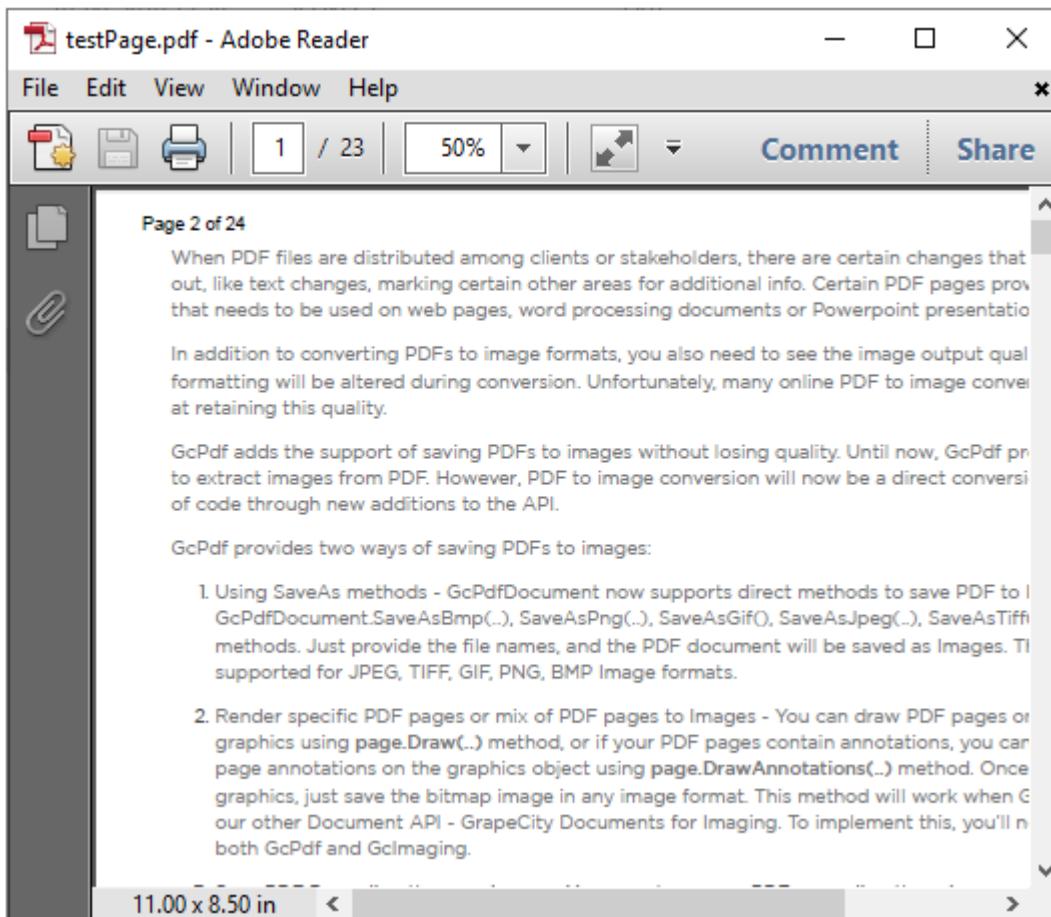
```
//This method demonstrates how to render a web page to a PDF file
public void RenderWebpageToPdf()
{
    // Specify the PDF file path to which the HTML webpage should be rendered
    var file = @"testPage.pdf";

    //Specify the HTML source
    var uri = new Uri(@"https://www.grapecity.com/controls/documents-pdf/whats-new");
```

```
using (var re = new GcHtmlRenderer(uri))
{
    //Define parameters for the PDF generator
    var pdfSettings = new PdfSettings()
    {
        // Skip the first page which is basically empty
        PageRanges = "2-100",
        // Sets the page width in inches
        PageWidth = 8.5f,
        // Sets the page height in inches
        PageHeight = 11f,
        // Sets page margins all around (default is no margins)
        Margins = new Margins(0.5f),
        // Ignores the page size defined by CSS
        IgnoreCSSPageSize = true,
        // Use landscape orientation to make sure long code lines are not
        truncated
        Landscape = true,
        // Sets the background color of the HTML page
        DefaultBackgroundColor = Color.Azure,
        // Sets the scale of the webpage
        Scale = 1.1f,
        // Displays headers and footers on each page
        DisplayHeaderFooter = true,
        // Specify custom header
        HeaderTemplate =
"<div style=\"font-size:12em;width:1000px;margin-left:30px;margin-right:30px\">" +
"<span style=\"float:left\">Page <span class=\"pageNumber\"></span> of <span
class=\"totalPages\"></span></span>" +
"<span style=\"float:right\"><span class=\"date\"></span></span>" +
"</div>"
    };
    // Create a PDF file from the source HTML
    re.RenderToPdf(file, pdfSettings);
}
}
```

Back to Top

The snapshot of the resulting PDF document is depicted below:



Note: In order to render an HTML page to PDF document, the fonts used on that page should be already installed on your system.

Render HTML String to PDF Document

GcHtml can render an HTML string to PDF document.

To render the HTML string to a PDF document, follow the steps below:

1. Specify the PDF file path for rendering HTML string.
2. Specify the HTML string.
3. Define the PDF document settings using the **PdfSettings** class.
4. Convert the HTML string to PDF file using the **RenderToPdf** method of **GcHtmlRenderer** class.

C#

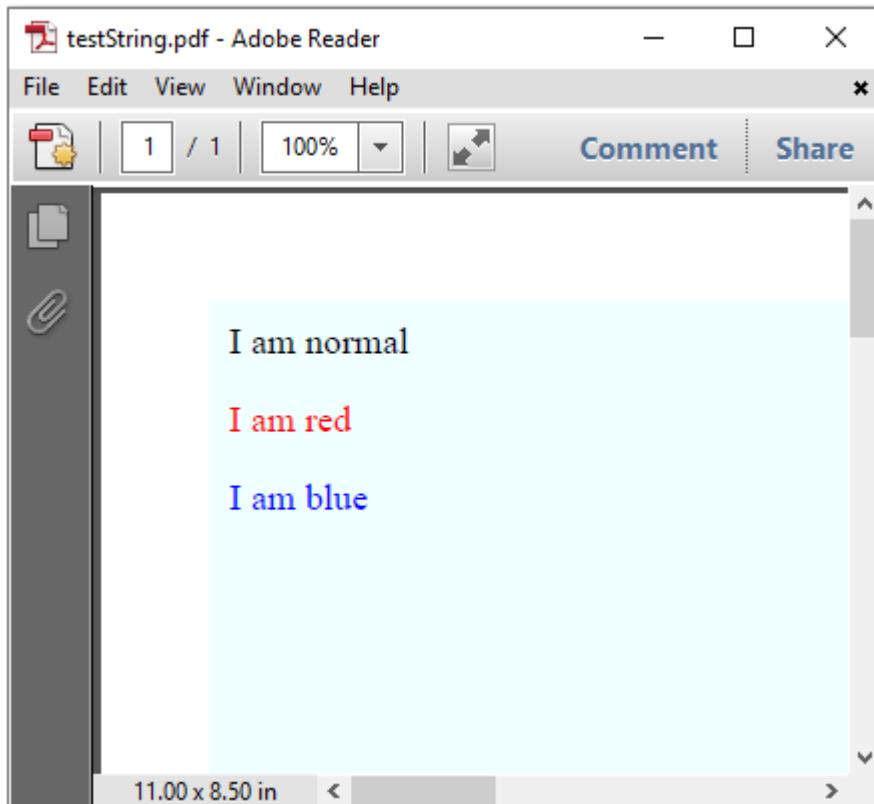
```
//This method demonstrates how to render a web page to a PDF file
public void RenderHTMLStringToPdf()
{
    //Specify the PDF file path to which the HTML webpage should be rendered
    var file = @"testString.pdf";

    //Specify the HTML source
    var htmlString =
"<html><body><p>I am normal</p><p style = 'color:red;'> I am red</p><p style =
'color:blue;'>I am blue</p>" +
"<p style = 'font-size:50px;'></p></body></html>";
```

```
using (var re = new GcHtmlRenderer(htmlString))
{
    //Define parameters for the PDF generator
    var pdfSettings = new PdfSettings()
    {
        // Skip the first page which is basically empty
        PageRanges = "2-100",
        // Sets the page width in inches
        PageWidth = 8.5f,
        // Sets the page height in inches
        PageHeight = 11f,
        // Sets page margins all around (default is no margins)
        Margins = new Margins(0.5f),
        // Ignores the page size defined by CSS
        IgnoreCSSPageSize = true,
        // Use landscape orientation to make sure long code lines are not
truncated
        Landscape = true,
        //Sets the background color of the HTML page
        DefaultBackgroundColor = Color.Azure,
    };
    //Create a PDF file from the source HTML
    re.RenderToPdf(file, pdfSettings);
}
}
```

Back to Top

The snapshot of the resulting PDF document is depicted below:



Render HTML Table to PDF Document

GcHtml can render an HTML table to PDF document.

To render the HTML table to a PDF document, follow the steps below:

1. Specify the PDF file path for rendering HTML table.
2. Create the HTML table.
3. Define the PDF document settings using the **PdfSettings** class.
4. Convert the HTML table to PDF file using the **RenderToPdf** method of **GcHtmlRenderer** class.

C#

```
public void RenderTable_ToPdf()
{
    //Specify the PDF file path to which the HTML webpage should be rendered
    var file = @"testTable.pdf";

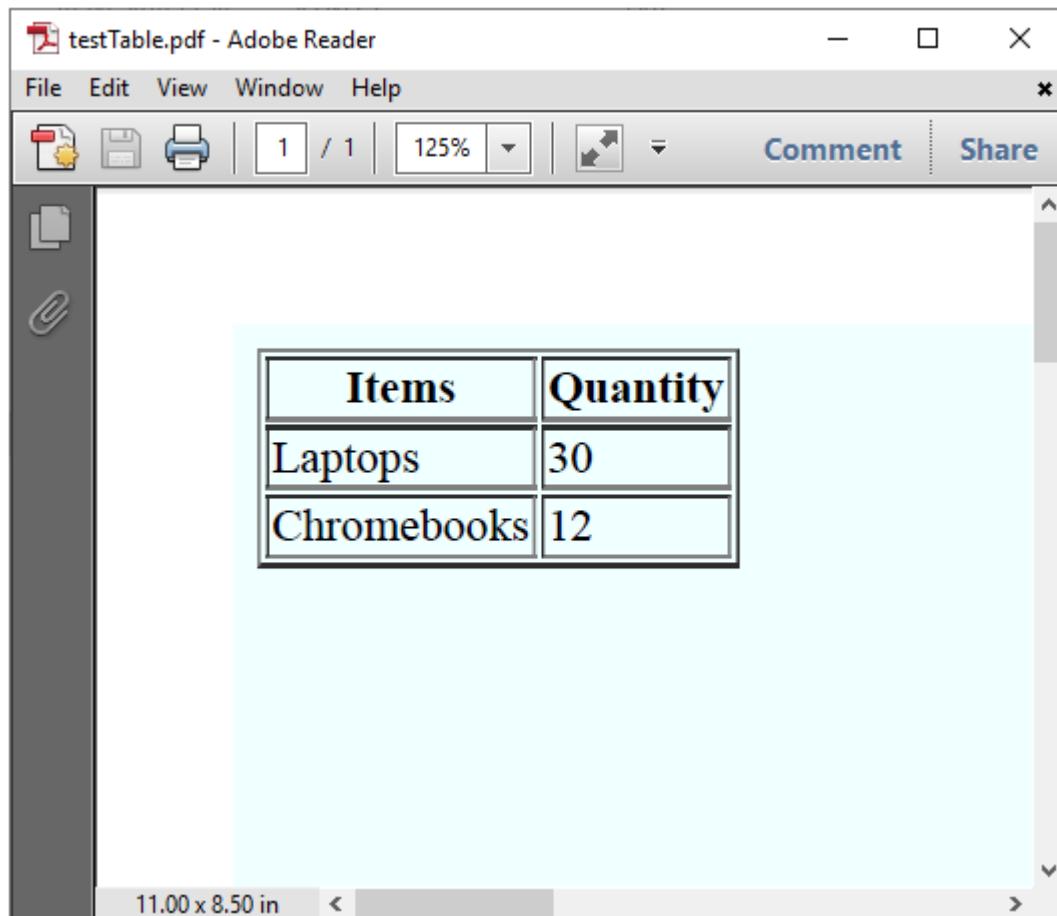
    //Create HTML Table
    var htmlTable = "<table border='1'><tr><th>Items</th><th>Quantity</th>" +
        "</tr><tr><td>Laptops</td><td>30</td></tr><tr><td>Chromebooks</td>" +
        "<td>12</td></tr></table>";

    using (var re = new GcHtmlRenderer(htmlTable))
    {
        //Define parameters for the PDF generator
        var pdfSettings = new PdfSettings()
        {
            // Sets the page width in inches
            PageWidth = 8.5f,
```

```
// Sets the page height in inches  
PageHeight = 11f,  
// Sets page margins all around (default is no margins)  
Margins = new Margins(0.5f),  
// Ignores the page size defined by CSS  
IgnoreCSSPageSize = true,  
// Use landscape orientation to make sure long code lines are not  
truncated  
Landscape = true,  
//Sets the background color of the HTML page  
DefaultBackgroundColor = Color.Azure,  
};  
  
//Create a PDF file from the source HTML  
re.RenderToPdf(file, pdfSettings);  
}  
}
```

Back to Top

The snapshot of the resulting PDF document is depicted below:



Add HTML Content to PDF Document

You can add an HTML page or string to a PDF document using the `DrawHtml` method of `GcPdfGraphicsExt` class.

To add an HTML formatted string to a PDF document, follow the steps below:

1. Create an instance of **GcPdfDocument** class.
2. Configure the PDF settings using the **HtmlToPdfFormat** class.
3. Add an HTML string to a PDF document using the **DrawHtml** method of **GcPdfGraphicsExt** class. You can also add a webpage to the PDF document in this manner. This is shown in the code snippet.
4. Save the PDF file using the **Save** method of **GcPdfDocument** class.

C#

```
//This method demonstrates how to add an HTML formatted string to a PDF file
public void DrawHtmlToPdf()
{
    //Configure PDF document
    var doc = new GcPdfDocument();
    var page = doc.Pages.Add();
    var g = page.Graphics;

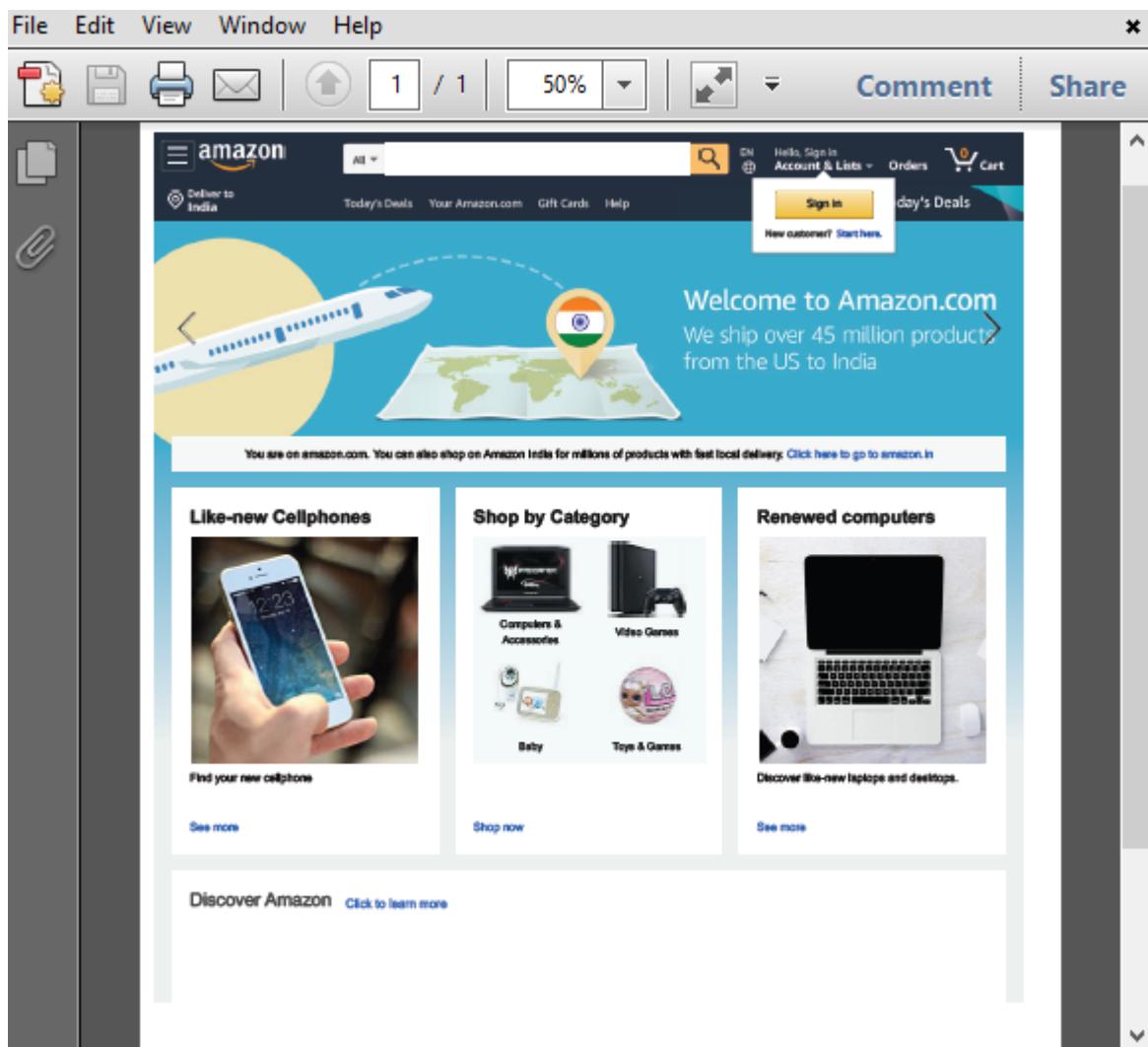
    //Configure PDF settings
    HtmlToPdfFormat pdfFormat = new HtmlToPdfFormat(false);
    pdfFormat.DefaultBackgroundColor = Color.White;
    pdfFormat.MaxPageHeight = 8;
    pdfFormat.MaxPageWidth = 8;

    //Draw HTML string
    GcPdfGraphicsExt.DrawHtml(g, new Uri("https://www.amazon.com/"), 10, 30,
pdfFormat, out SizeF size);
    //GcPdfGraphicsExt.DrawHtml(g, "Hello, <b>World!</b>", 72, 72, pdfFormat, out
SizeF size);

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

[Back to Top](#)

The resulting image is shown below:



For more information on rendering HTML to PDF using GcPdf, see [GcPdf sample browser](#).

Save PDF as Image

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.

You can save a PDF document as an image by using the below methods:

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 of GcPdfDocument class.
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);
```

[Back to Top](#)

Save as SVG

In addition to above mentioned common image formats, GcPdf also lets you save the PDF pages as SVG or its compressed format SVGZ. You can use [SaveAsSvg](#) and [ToSvgz](#) methods of the [GrapeCity.Documents.Pdf.Page](#) class to export an instance of pdf page to SVG file or stream(.svg) or a byte array(.svgz).

C#

```
var pdfDoc = new GcPdfDocument();
using (var fs = new FileStream("Test.pdf", FileMode.Open, FileAccess.Read,
FileShare.Read))
{
    pdfDoc.Load(fs);
    var page = pdfDoc.Pages[0];

    // Render a PDF page to the .svg file
    page.SaveAsSvg("GcPDFRenderToSVG.svg", null,
        new SaveAsImageOptions() { Zoom = 2f },
        new XmlWriterSettings() { Indent = true });

    // Render a PDF page to the byte array with compressed data in SVGZ format
```

```
    var svgzData = page.ToSvgz(new SaveAsImageOptions() { Zoom = 1f });
    File.WriteAllBytes("GcPDFRenderToSVGZ.svgz", svgzData);
}
```

Limitation

Text is always rendered as graphics [using paths](#). Hence, resulting .svg files for text pages are large and it is not possible to select or copy text on the SVG images opened in the browsers.

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 of [Page](#) class to render the PDF file pages content with or without annotations and the [DrawAnnotations](#) method of [Page](#) class 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

Apart from the above, you can also render text in PDF and save it as an image by enabling TrueType hinting instructions as explained below:

Support for TrueType Hinting Instructions

GcPdf supports enabling TrueType hinting instructions while rendering text on **GcPdfGraphics** and saving it as an image.

Hinting instructions are included in some TrueType fonts which improve their look by reusing some glyph parts in different glyphs regardless of their font size. TrueType hinting instructions, in GcPdf, supports drawing CJK characters as combinations of other smaller glyph pieces which enhances their final look.

For fonts which include TrueType glyph hinting instructions, the **EnableHinting** property of the **Font** class is set to true, for the others it is set to False. Further, to apply the hinting instructions of the font, **EnableFontHinting** property of the **SaveAsImageOptions** class must be set to true (the default value).

However, if the **EnableHinting** property is explicitly set to false, then the hinting instructions cannot be enabled.

As the default value of both the properties is true, hence the hinting instructions are supported for any TrueType font which includes them.

Disabled Hinting Instructions

The quick brown fox jumps over the lazy dog.

The quick brown fox jumps over the lazy dog.

入秋空污警報！這幾招遠離PM2.5學起來

Enabled Hinting Instructions

The quick brown fox jumps over the lazy dog.

The quick brown fox jumps over the lazy dog.

入秋空污警報！這幾招遠離PM2.5學起來

To enable TrueType hinting instructions for Chinese string:

1. Load a Chinese Font file.
2. Initialize the **GcPdfGraphics** class which represents a graphics object on a PDF page.
3. Define a Chinese string and configure **TextFormat** properties.
4. Draw the Chinese string.
5. Create an instance of **SaveAsImageOptions** class and use it to set the **EnableFontHinting** property to true .
6. Save the image.

C#

```
var font = Font.FromFile("kaiu.ttf");
GcPdfDocument doc = new GcPdfDocument();
{

    GcPdfGraphics g = doc.NewPage().Graphics;
    {

        //Draw the string with hinting instructions set to true
        string s1 = @"入秋空污警報！這幾招遠離PM2.5學起來";
        //Define text formatting attributes
        var tf1 = new TextFormat()
        {
            Font = font,
            FontSize = 20,

        };
        g.DrawString(s1, tf1, new PointF(10, 110));

        SaveAsImageOptions imgOptions = new SaveAsImageOptions();
        imgOptions.EnableFontHinting = true;

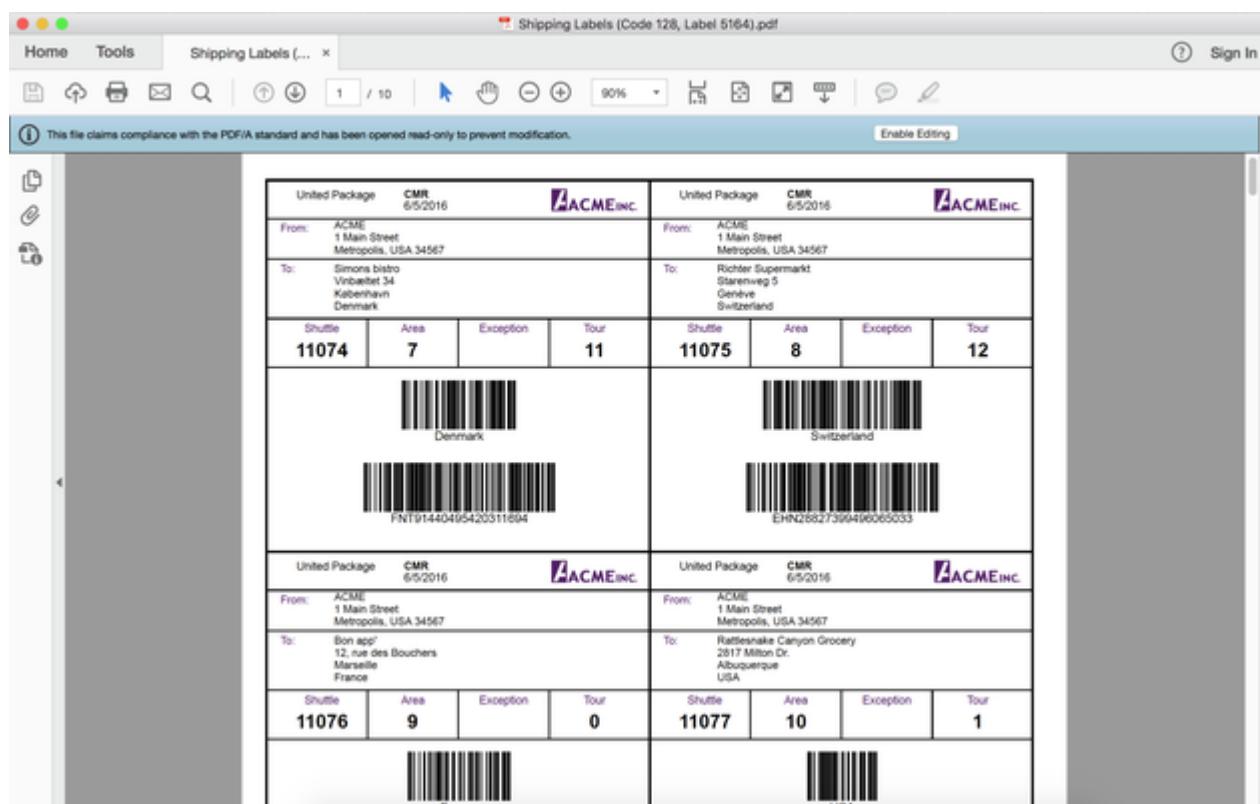
        doc.SaveAsPng("ChineseFontwithHintingInstructions1.png", null, imgOptions);
    }
}
```

[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	<p>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:</p> <ul style="list-style-type: none">● 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) <p>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.</p> <p>Multiple AIs (along with their data) can be combined into a single bar code.</p>

	<p>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.</p> <p>To insert FNC1 character, set "\n" (for C#), or "vbLf" (for VB) to Text property at runtime.</p>
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	<p>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.</p> <p>MicroPDF417 allows you to insert an FNC1 character as a field separator for variable length Application Identifiers (AIs).</p> <p>To insert FNC1 character, set "\n" (for C#), or "vbLf" (for VB) to Text property at runtime.</p>
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

	<p>variable length Application Identifiers (AIs).</p> <p>To insert FNC1 character, set “\n” (for C#), or “vbLf” (for VB) to Text property at runtime.</p>
RSSLimited	<p>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.</p> <p>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.</p>
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 = TextAlign.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 and Editor. It is a cross platform solution for viewing and editing PDF files on Windows, MAC, Linux, iOS and Android devices. 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. The Viewer is supported on following browsers:

- Firefox
- Chrome
- Opera
- IE 11/Edge
- Safari 9+
- Mobile Safari (iOS 10+)

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.

The screenshot shows the GrapeCity Documents PDF Viewer interface. On the left is a vertical toolbar with icons for file operations (New, Open, Save, Print, Copy, Paste, Find, Replace, Undo, Redo), a search icon, and a refresh icon. The main area displays a PDF page with the title "Wetlands.pdf". The page contains a block of text about wetlands and a photograph of a wetland landscape with water and green vegetation. The top navigation bar includes icons for back, forward, search, and zoom, along with a status bar showing "2 / 3" and "80%".

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.

- **Edit PDF documents**

The GcDocs PDF Viewer allows you to edit PDF documents. You can use Annotation and Form editors, add comments and share and collaborate PDF documents with other users as well.

- **Touch Support**

The GcDocs PDF Viewer supports touch events. These can be used to draw ink annotations in Annotation editor with your finger, pen or stylus, to select, move, resize or edit the annotations and form fields. The touch events are supported on iOS version 12+ and Android 9+ systems.

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.

Standard Deployment

Included with every GcPdf license purchase. This deployment includes all features of the viewer except the features that require the viewer to use SupportApi property (i.e. connect the viewer to GcPdf on the server).

Professional Deployment

Includes all the features of the Standard license in addition to features that require the viewer to use SupportApi property (i.e. connect the viewer to GcPdf on the server). The Professional license is an additional fee.

Please refer to [GcPDF Viewer License Options](#) to know more about Standard and Professional Viewer License.

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 initializing 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.

View PDF

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

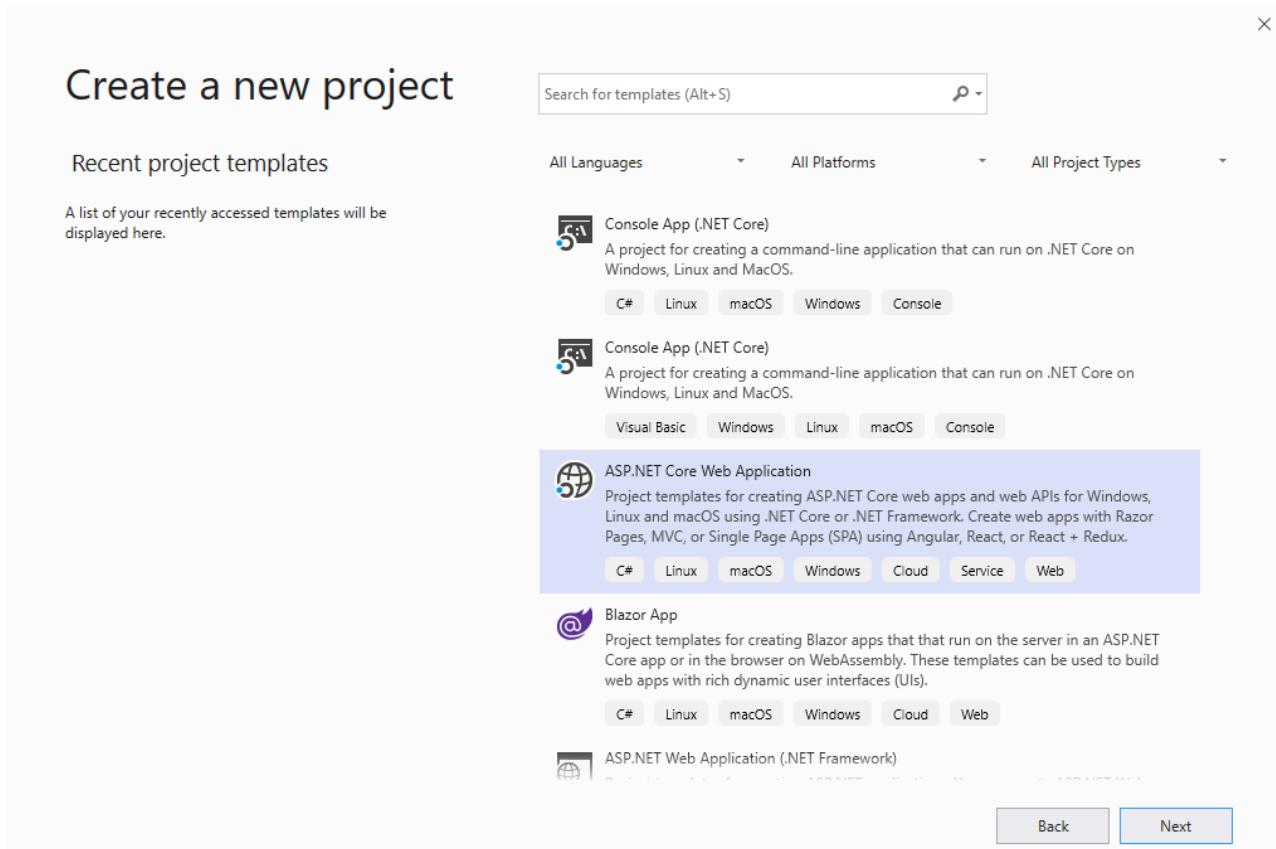
Refer the following topics to configure PDF viewer and use its features:

- [Configure PDF Viewer](#)
- [Features](#)

Configure PDF Viewer

The steps listed below describe how to create an ASP.NET Core Web Application that uses GcDocs PDF Viewer to view PDF Files.

1. Open Microsoft Visual Studio and select **Create a new project | ASP.NET Core Web Application**.



2. In the 'Create a new ASP.NET Core web application' dialog, select the following:

- o .NET Core / ASP.NET Core 3.1
- o 'Empty' - Project template

Note: Make sure that 'Configure for HTTPS' option is unchecked to avoid warnings shown by FireFox on Windows.

Create a new ASP.NET Core web application

The screenshot shows the 'Create a new ASP.NET Core web application' dialog. At the top, there are two dropdown menus: '.NET Core' and 'ASP.NET Core 3.1'. Below them is a list of project templates:

- Empty**: An empty project template for creating an ASP.NET Core application. This template does not have any content in it.
- API**: A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.
- Web Application**: A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.
- Web Application (Model-View-Controller)**: A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.
- Angular**: A project template for creating an ASP.NET Core application with Angular.
- React.js**: A project template for creating an ASP.NET Core application with React.js.

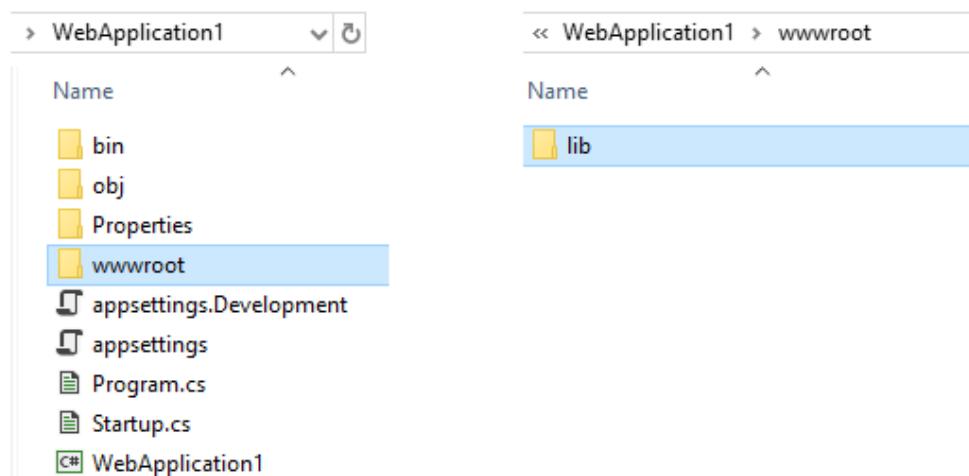
On the right side, there are sections for 'Authentication' (with 'No Authentication' selected), 'Advanced' (with options for 'Configure for HTTPS' and 'Enable Docker Support'), and a dropdown for 'Linux'. At the bottom, there are buttons for 'Back' and 'Create'.

3. Make sure that sample project builds and runs fine (shows the 'Hello World!' screen in browser). Next steps assume that the project is named as 'WebApplication1'.

4. Open the project in File Explorer and create the 'wwwroot' and 'lib' directories as shown below:

WebApplication1\wwwroot

WebApplication1\wwwroot\lib

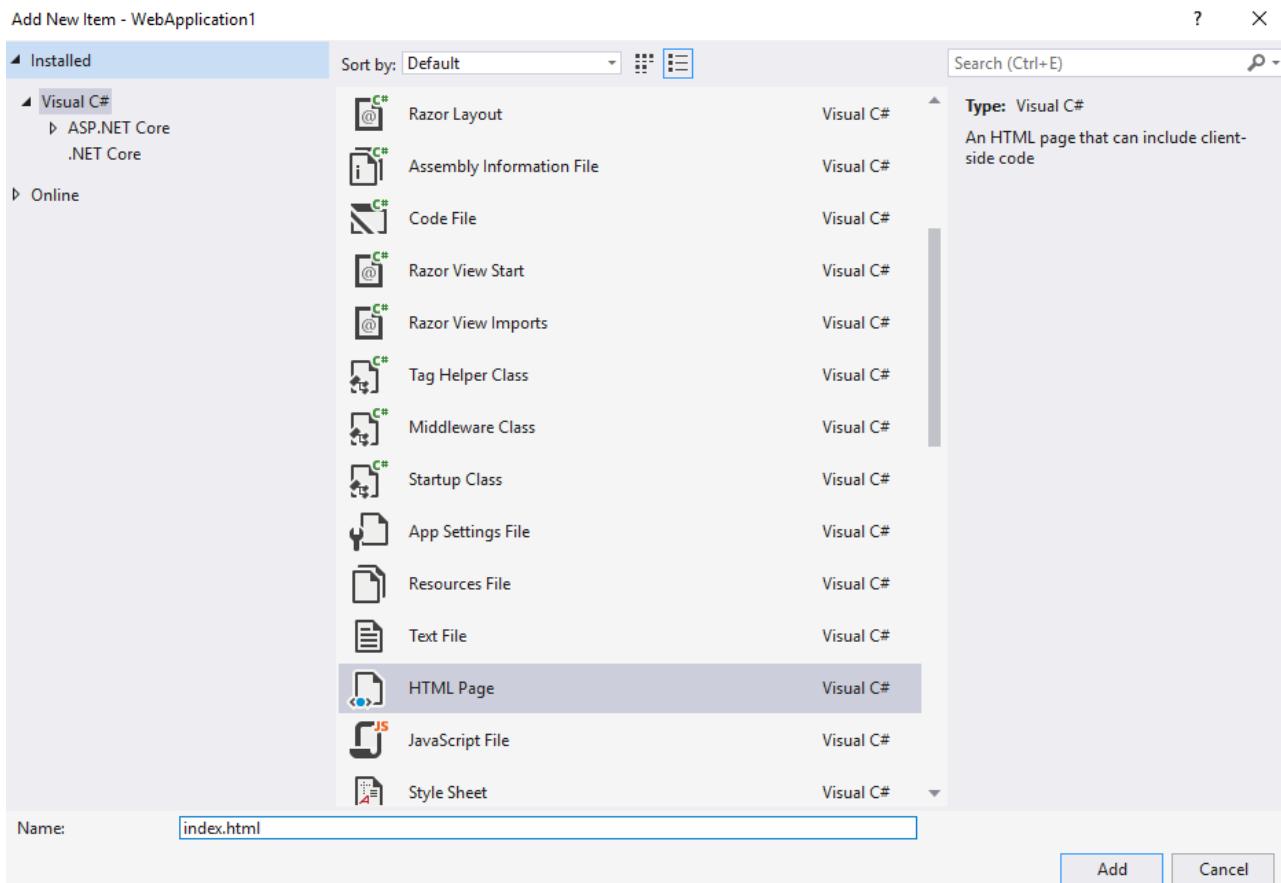


5. Run the following command to install GcDocs PDF Viewer. Make sure that the directory location in command prompt is set to *lib* folder. The GcDocs PDF Viewer will be installed in *WebApplication1\wwwroot\lib*:

```
npm install @grapecity/gcpdfviewer
```

 **Note:** The location where this command runs is important as the Viewer is placed relative to it. The above command puts the Viewer in `WebApplication1\wwwroot\lib\node_modules@grapecity\gcpdfviewer`.

6. In VS, add a new HTML page to 'wwwroot' folder and name it 'index.html'.



7. Paste the following code in the index.html file.

```
index.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="lib/node_modules/@grapecity/gcpdfviewer/gcpdfviewer.js "></script>
</body>
</html>
```

 **Note:** Besides adding GcDocs PDF Viewer to the page, the above code also loads a static PDF (Wetlands.pdf) into it on startup. To make sure it works, place the Wetlands.pdf in the wwwroot directory.

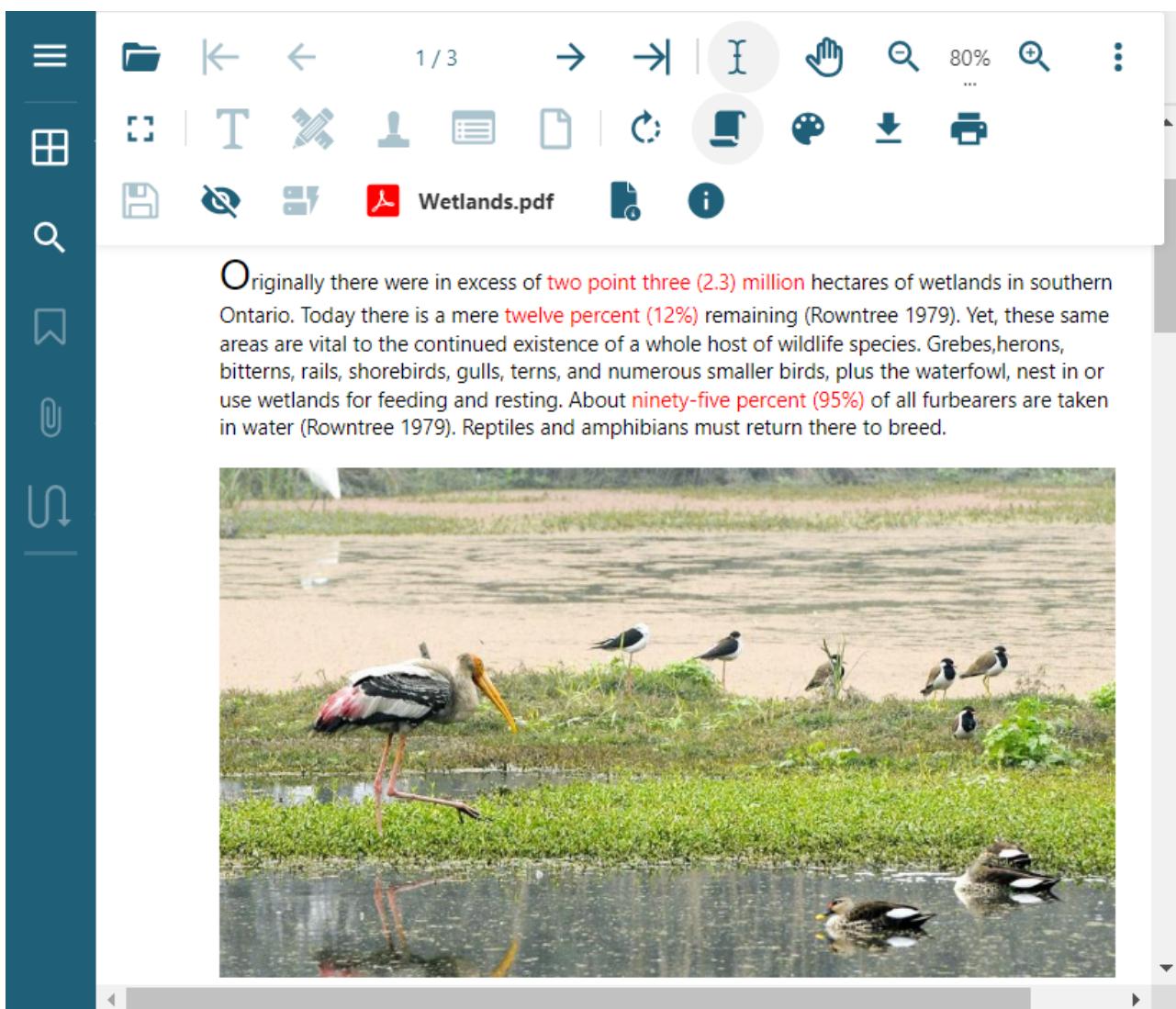
8. Modify the Startup.cs file by replacing the default 'Configure' method with below code snippet. This will open the index.html by default, when the app starts.

Startup.cs

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())

    { app.UseDeveloperExceptionPage(); }
    app.UseRouting();
    app.UseDefaultFiles();
    app.UseStaticFiles();
}
```

9. Build and run the application. A page with the GcDocs PDF Viewer (loaded with Wetlands.pdf) will show in your default browser.



For more information, refer [View PDF](#) in GcDocs PDF Viewer demos.

Features

GcDocs PDF Viewer supports standard as well as advanced PDF viewer features:

- [Toolbar and Panel Icons](#)
- [Custom Context Menu](#)
- [Advanced Search Options](#)
- [View PDF Elements](#)

Toolbar and Panel Icons

GcDocs PDF Viewer features 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 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.
Hide Annotations		Allows you to hide annotations in a PDF document.
Document Properties		Allows you to view document properties like File Name, File Size, Title etc.
About		Allows you to view the version number of PDF Viewer.

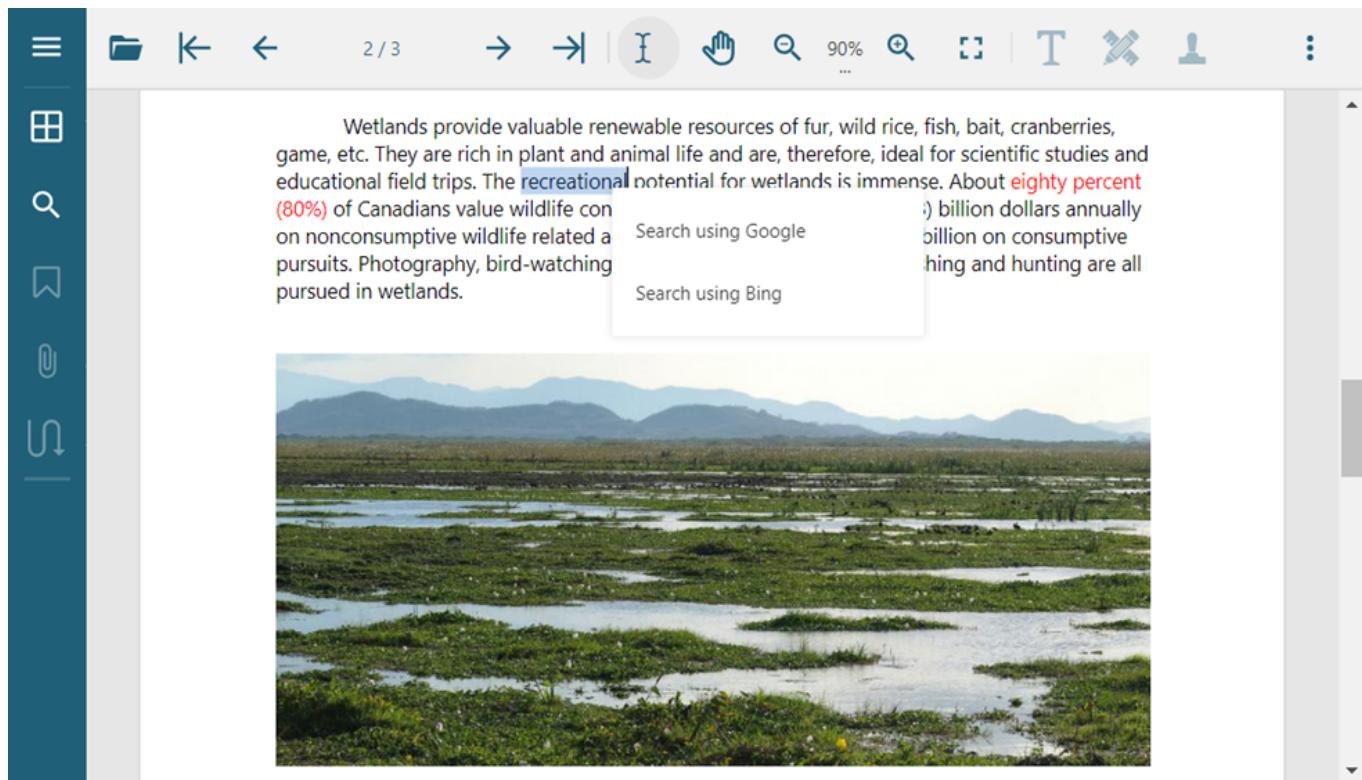
Features	Panel Icons	Description
Thumbnail navigation		Allows you to see the preview of all available pages in the PDF document.
Advanced search option		Allows you to search for text with match-case and whole-word search options.
Page-level and document-level attachments		Allows you to view the attachments 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.

Custom Context Menu

GcDocs PDF Viewer provides 'Copy' and 'Print' options in its context menu, by default. However, the context menu

options can be customized for other operations, like searching selected text by using different Web Search Engines. The below image displays custom context menu when selected text is right clicked.



To configure custom context menu in GcDocs PDF Viewer:

Index.cshtml

```
viewer.options.onBeforeOpenContextMenu = function (items, mousePosition, viewer) {
    var selectedText = viewer.getSelectedText();

    if (selectedText) {
        // Remove existent items:
        items.splice(0, items.length);
        // Add own menu items:
        items.push({
            type: 'button',
            text: 'Search using Google',
            onClick: function () {
                window.open('http://www.google.com/search?q=' +
encodeURI(selectedText), '_blank');
            }
        });

        items.push({
            type: 'button',
            text: 'Search using Bing',
            onClick: function () {
                window.open('https://www.bing.com/search?q=' +
encodeURI(selectedText), '_blank');
            }
        });
    }
}
```

```

        }
    });
}

return true;
};
}

```

You can also disable the GcDocs PDF Viewer's context menu to use browser's context menu by using below code:

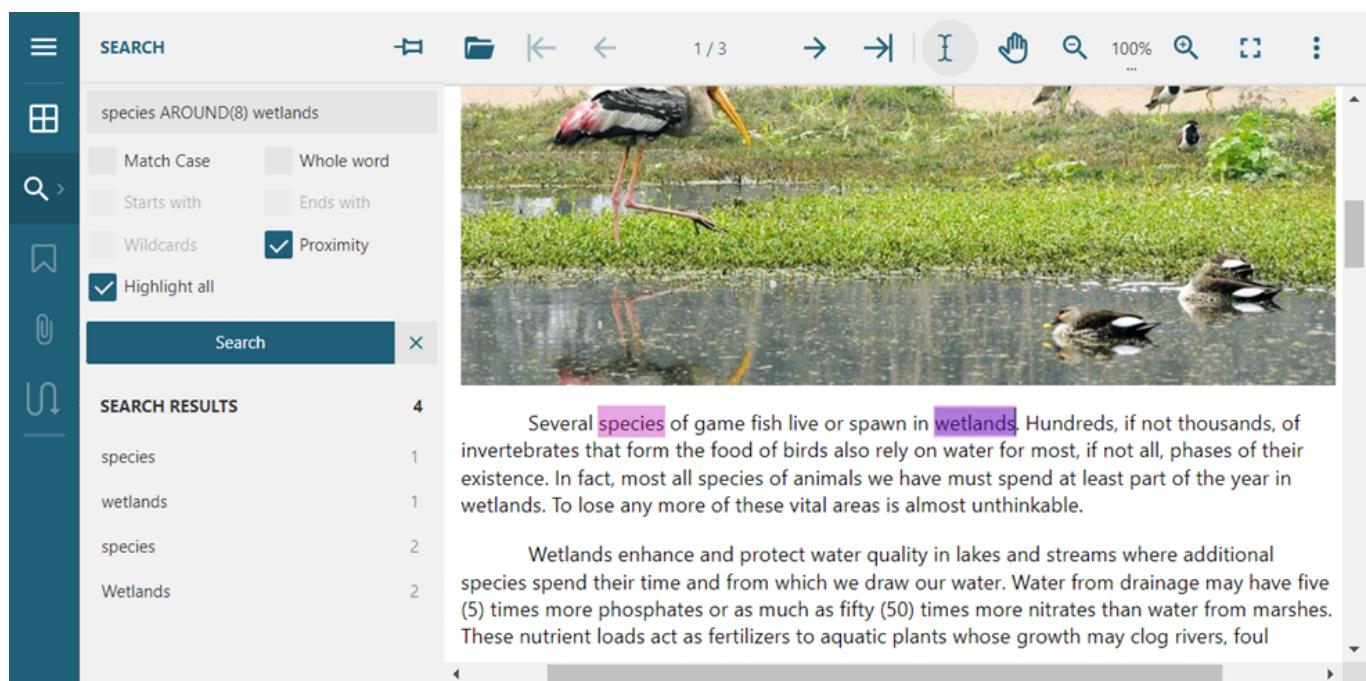
Index.cshtml

```
var viewer = new GcPdfViewer("#root", { useNativeContextMenu: true });
```

Note: The default value of **useNativeContextMenu** is false. When it is set to true, some context menu functions become unavailable (for example, actions of Editor and Reply tool).

Advanced Search Options

GcDocs PDF Viewer provides advanced search options to help you find the desired content in a PDF document. It eliminates the irrelevant options and narrows down the scope of a search query. Along with the count of total search results, it displays the page number on which the specific result is found.



The search results are highlighted all at once when 'Highlight all' option is checked in the search panel. The below code example shows how to change the default highlight colors by using the 'useCanvasForSelection' option:

Index.cshtml

```
var viewer = new GcPdfViewer('#root',
{
    useCanvasForSelection:
    {
        selectionColor: 'rgba(0, 0, 195, 0.25)',
        highlightColor: 'rgba(255, 0, 0, 0.35)'
    }
});
```

```
        inactiveHighlightColor: "rgba(180, 0, 170, 0.35)"  
    }  
});
```

Various advanced search options are provided in GcDocs PDF Viewer which are explained as below:

Match Case

This option finds all those instances in a PDF document that are written in the same case (lower or upper) as specified in the search query.

Example: If you search "test", the search results will display "**test**", "**tests**", "**testing**" but not "Test".

Whole Word

The 'Whole Word' option finds all those occurrences in a PDF document which contain the whole word as specified in the search query.

Example: If you search "demo", the search results will display "**demo**", "**Demo**" but not "demonstration", "demos".

Starts With

This option finds all the occurrences in a PDF document which starts with the characters specified in the search query.

Example: If you search "us", the search results will display "**us**", "**used**", "**Users**" but not "various", "status".

Ends With

This option finds all the occurrences in a PDF document which ends with the characters specified in the search query.

Example: If you search "at", the search results will display "**at**", "format", "treat" but not "attribute", "atom".

Wildcards

The 'Wildcard' option can be used to maximize the search results. You can type a part of a word and use any number of wildcard characters with it. GcDocs PDF Viewer supports two wildcard characters:

- **Asterisk (*)** - It can be used to specify any number of characters, anywhere in the word.
Example: If you search "te*", the search results will display "**tentative**", "**text**", "**extensive**", "**polite**"
- **Question Mark (?)** - It can be used to specify a single character zero or one time, anywhere in the word.
Example: If you search "t?e", the search results will display "**treat**", "**starter**", "**elaborate**", "**centre**"

 **Note:** Wildcard search option cannot be combined with 'Starts With', 'Ends With' and 'Whole Word' options.

Proximity

The 'Proximity' option can be used to search for two or more words that are separated by a certain number of words from each other. The operator AROUND(n) can be used to specify the maximum count of words between search terms. The search results includes only those words which are present on the same page of a PDF document.

Example: Consider the below text in a PDF document:

Several species of game fish live or spawn in wetlands. Hundreds, if not thousands, of invertebrates that form the food of birds also rely on water for most, if not all, phases of their existence. In fact, most all species of animals we have must spend at least part of the year in wetlands. To lose any more of these vital areas is almost unthinkable.

Case 1:

Search Query: species AROUND(8) wetlands

Result: **species** of game fish live or spawn in **wetlands**

Explanation: The words "species" and "wetlands" are present at a gap of 7 words from each other. In order to display this search result, the value of 'n' should be 7 or greater.

Case 2:

Search Query: species wetlands

Results: **species** of game fish live or spawn in **wetlands**

species of animals we have must spend at least part of the year in **wetlands**

Explanation: If operator AROUND(n) is not specified, the search results will include all words from query without any location constraint.

Case 3:

Search Query: species AROUND(8) wetlands AROUND(4) thousands

Result: **species** of game fish live or spawn in **wetlands**. Hundreds, if not **thousands**

Explanation: The words "species", "wetlands" and "thousands" are present at the specified gaps.

 **Note:** Proximity search option cannot be combined with 'Starts With', 'Ends With' and 'Wildcard' options.

View PDF Elements

GcDocs PDF Viewer supports viewing different PDF elements such as layers, structured content or XFA content.

View Layers

PDF documents can contain content in different layers (also known as optional content) and a particular layer can be made visible or invisible as required. GcPdf allows you to work with layers and set their properties, refer [Layers](#) for more information.

In GcDocs PDF Viewer, you can examine these layers and show or hide content associated with each layer by using the 'Layers' panel, provided in its sidebar. The viewer also saves visibility state of the layers on pressing the '**Save**' button.

Enable Layers Panel in GcDocs PDF Viewer

The Layers panel can be displayed by enabling the addLayersPanel in the viewer using below code:

```
Index.cshtml
```

```
viewer.addLayersPanel();
```

The below GIF shows a PDF document containing 'English' and 'Russian' language layers and how they can be made visible or invisible using the GcDocs PDF Viewer Layers panel.



The screenshot shows a software interface for document management. At the top, there's a toolbar with icons for file operations, search, and zoom. Below the toolbar, a main area displays a blue header for "GrapeCity Documents" with the sub-header "Fast, Efficient Document APIs for .NET 5 and Java Applications". The main content area features several sections: "High-Speed, Small Footprint, No Dependencies" with a brief description and a "Zero Dependencies" icon; "Full .NET Support for Windows, Linux, and Mac" with a brief description and a ".NET" icon; and "Comprehensive, Highly Programmable" with a brief description and a "Supports Hundreds of Features" icon. To the right of these descriptions are three small screenshots of document pages: an invoice, a spreadsheet, and another document page.

Fast, Efficient Document APIs for .NET 5 and Java Applications

Take total control of your documents with ultra-fast, low-footprint APIs for enterprise apps.

- Generate, load, edit, save XLSX spreadsheets, PDF, Images, and DOCX files using C# .NET, VB.NET, or Java
- View, edit, print, fill and submit documents in JavaScript PDF Viewer and PDF Editor
- Compatible on Windows, macOS, and Linux
- No dependencies on Excel, Word, or Acrobat
- Deploy to a variety of cloud-based services, including Azure, AWS, and AWS Lambda
- Product available individually or as a bundle

High-Speed, Small Footprint, No Dependencies
The .NET 5 Document API is designed to generate large, optimized documents fast – while remaining lightweight and extensible, giving you greater flexibility, and creativity in developing your applications.

Full .NET Support for Windows, Linux, and Mac
Develop for any .NET platform or major operating system with a single code base. Use in your apps for .NET, C#, VB.NET, .NET Framework, Mono, Xamarin.iOS, and Xamarin.Android.

Comprehensive, Highly Programmable
Do more with your Excel spreadsheets, Word documents, PDFs, and images with our feature-rich APIs. Create, load, edit, save, and convert your business documents with intuitive tools.

Zero Dependencies

.NET

Supports Hundreds of Features

View Structured Content in Tagged PDF

GcPdf allows you to create or modify tagged PDF documents. Refer [Tagged PDF](#) to know more. You can use the Structure Tree panel of GcDocs PDF Viewer to load a tagged PDF and navigate between the available structured elements such as heading, table, paragraph etc.

The below image shows the Structure Tree panel when a tagged PDF is opened in GcDocs PDF Viewer. The number in the page header indicates the total number of items in the structure tree on that page (excluding the root element).

The screenshot shows the GcPdfViewer interface. On the left, there is a dark blue sidebar titled "STRUCTURE TREE" containing a tree view of page content. The tree structure includes nodes like "Page 1", "Role Part", "Role P", "Role Span", "Content", "Role H1", and "Role P". On the right, there are several content sections with titles and descriptions. One section is titled "What is C1Olap" with a subtitle about it being a suite of .NET controls for analytical processing. Another section is titled "Introduction to Olap" with a subtitle explaining what Olap means ("online analytical processing"). Both sections include small icon bars.

The below code example shows how to enable the Structure Tree panel by using the **addStructureTreePanel** method.

Index.cshtml

```
var viewer = new GcPdfViewer(selector);
viewer.addStructureTreePanel();
viewer.open("read-tags.pdf");
```

View XFA Content

XFA stands for XML Forms Architecture and can be used to enhance the processing of web forms. Refer [XFA](#) to know more.

GcDocs PDF Viewer supports displaying PDF documents containing XFA content, by default. It also allows you to select and copy the content from XFA forms. Moreover, GcDocs PDF Viewer lets you use links, reset, submit and print the XFA form if these JavaScript actions are included in the form. To disable the rendering of XFA content in PDF, you can use the **enableXfa** option as shown in the below code example:

Index.cshtml

```
// Turn off XFA forms rendering.
var viewer = new GcPdfViewer(selector, { enableXfa: false });
```

Limitation

- XFAF (XFA Foreground) subset is not supported.
- Editing and save operations are not supported in XFA forms.

Edit PDF

GrapeCity Documents PDF Viewer allows you to edit PDF documents. You can use Annotation and Form editors, add comments and share and collaborate PDF documents with other users as well.

- [Configure PDF Editor](#)
- [Editors](#)
- [Features](#)
- [Comments Tool](#)
- [Graphical Signature Tool](#)
- [Share and Collaborate](#)

Configure PDF Editor

When connected to a server running GcPdf (using the SupportApi property, see below), GcDocs PDF Viewer allows you to edit PDF documents by using the general editing options, annotations and form editor tools. To configure **GcDocs PDF Viewer** for editing PDF documents, you need to download **SupportAPI** to connect to GrapeCity Documents for PDF (GcPdf) on server, which enables the PDF editing operations and saves PDF documents on client. The GcDocs PDF Viewer works with server-side API, that is, GcPdf via **SupportApi** property to save the modified changes and sends the PDF back to the client. There are two ways to fetch SupportAPI:

1. Using NuGet package
2. Building and using SupportApi from sources

 **Note:** SupportApi is a pre-requisite to edit PDF documents, and is available only with the [Professional Deployment License](#).

Configure GcDocs PDF Viewer for Editing

1. A PDF is loaded into the GcDocs PDF Viewer in one of the following ways:
 - Using GcPdf
 - Using **Open** button or **Ctrl-O** shortcut
 - Creating new PDF using the **New** button
2. Edits are done in the PDF by using general editing features or editing tools. At this point, all edits are stored locally by the viewer.
3. After pressing the '**Save**' button, the original PDF and edits are sent to the server via SupportApi.
4. The server applies edits to the sent file and sends the modified PDF back to the client.
5. The modified PDF can be saved locally, or opened in any PDF Reader or similar.

 **Note:** The edits or changes are NOT persisted by the server, they are applied to the PDF and the modified PDF is sent back.

Using the **GrapeCity.Documents.Pdf.ViewerSupportApi** package

The **GrapeCity.Documents.Pdf.ViewerSupportApi** package can be downloaded from [NuGet](#) and is also included in **nupkg** folder of the [GcPdf distribution zip](#) along with other packages. You need add the package in your project references.

ASP.NET Core Web Application

The steps listed below describe how to configure GcDocs PDF Viewer in an ASP.NET Core Web Application to view and edit PDF Files.

1. Open Microsoft Visual Studio and select **Create a new project | ASP.NET Core Web Application**.

The screenshot shows the Visual Studio search interface for project templates. The search bar at the top contains the text "Search for templates (Alt+S)". Below it, there are dropdown menus for "C#" (selected), "All platforms" (selected), and "Web". A "Clear all" button is also present. The results list includes:

- ASP.NET Core Web App**: A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content. It supports C#, Linux, macOS, Windows, Cloud, Service, and Web.
- Blazor Server App**: A project template for creating a Blazor server app that runs server-side inside an ASP.NET Core app and handles user interactions over a SignalR connection. This template can be used for web apps with rich dynamic user interfaces (UIs). It supports C#, Linux, macOS, Windows, Blazor, Cloud, Service, and Web.
- ASP.NET Core Web API**: A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers. It supports C#, Linux, macOS, Windows, Cloud, Service, and WebAPI.
- ASP.NET Core Empty**: An empty project template for creating an ASP.NET Core application. This template does not have any content in it. It supports C#, Linux, macOS, Windows, Cloud, Service, and Web.

2. In the **Create a new ASP.NET Core web application** dialog, select the following:
.NET Core 3.1(Long-term support)

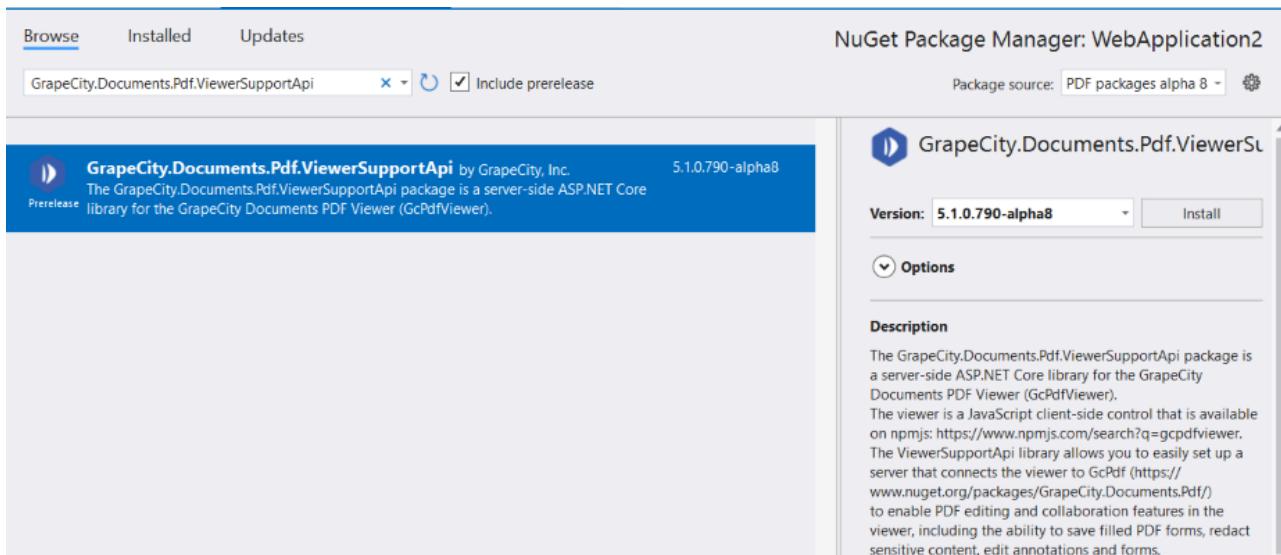
Additional information

The screenshot shows the "Create a new ASP.NET Core web application" dialog. The selected template is "ASP.NET Core Web App". The configuration options are as follows:

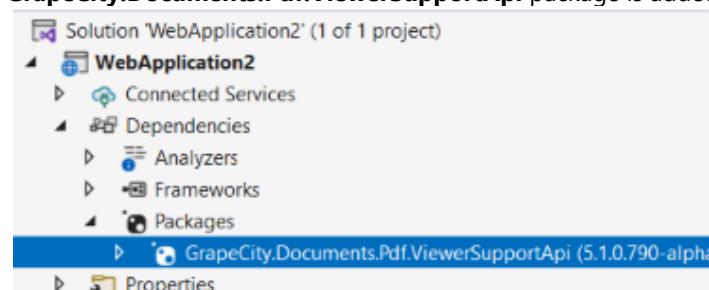
- Framework**: .NET Core 3.1 (Long-term support)
- Authentication type**: None
- Configure for HTTPS**: Unchecked
- Enable Docker**: Unchecked
- Docker OS**: Linux
- Enable Razor runtime compilation**: Unchecked

Note: Make sure that 'Configure for HTTPS' option is unchecked to avoid warnings shown by FireFox on Windows.

3. Make sure that sample project builds and runs fine (shows the 'Welcome' screen in browser). Next steps assume that the project is named as 'WebApplication2'.
4. Run the following command to install GcDocs PDF Viewer. Make sure that the directory location in command prompt is set to lib folder. The GcDocs PDF Viewer will be installed in WebApplication2\WebApplication2\wwwroot\lib:
npm install @grapeCity/gcpdfviewer
5. Right-click the project in Solution Explorer and choose Manage NuGet Packages
6. In the Package source on top right, select **nuget.org**.
7. Click **Browse** tab on top left and search for "GrapeCity.Documents.Pdf.ViewerSupportApi".
8. On the left panel, select **GrapeCity.Documents.Pdf.ViewerSupportApi** as shown in image below:



- On the right panel, click **Install** to install the **GrapeCity.Documents.Pdf.ViewerSupportApi** package and its dependencies into the project. When the installation is complete, make sure you check the **Packages** folder under **Dependencies** folder in your solution explorer and confirm whether the **GrapeCity.Documents.Pdf.ViewerSupportApi** package is added to your project dependencies.



- Modify the default content of WebApplication2\WebApplication2\Pages\Index.cshtml with the following code:

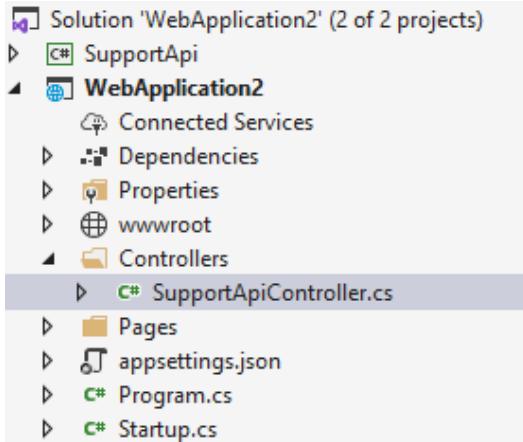
```
Index.cshtml

@page
@model IndexModel
{@ ViewData["Title"] = "Home page"; }
<style>
    .container {
        height: calc(100% - 128px);
        max-width: inherit;
    }

    #host, .pb-3 {
        height: 100%;
    }
</style>
<div id="host"></div>
<script src="~/lib/node_modules/@grapacity/gcpdfviewer/gcpdfviewer.js" asp-append-version="true"></script>
<script>
    var viewer = new GcPdfViewer("#host", { supportApi: 'api/pdf-viewer' });
    viewer.addDefaultPanels();
    viewer.addAnnotationEditorPanel();
    viewer.addFormEditorPanel();
    viewer.beforeUnloadConfirmation = true;
```

```
viewer.newDocument();  
</script>
```

11. Create a '**Controllers**' folder in **WebApplication2** project and add a class file '**SupportApiController.cs**' to it as shown below:



12. Modify **Startup.cs** by adding the following lines of code to default ConfigureServices() method:

```
services.AddMvc((opts) => { opts.EnableEndpointRouting = false; });  
services.AddRouting();
```

and following line of code to Configure() method:

```
app.UseMvcWithDefaultRoute();
```

The final startup.cs will look like below:

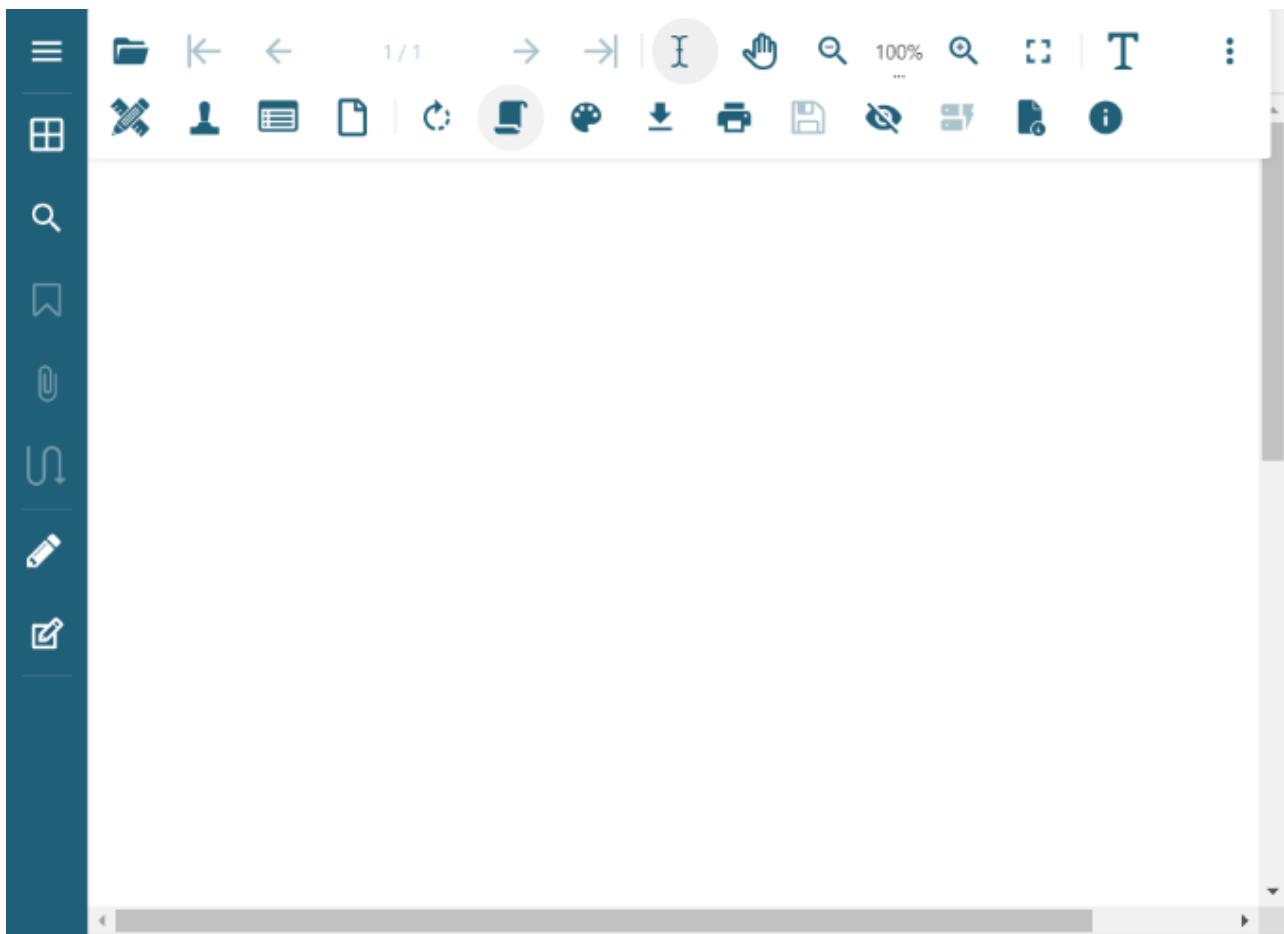
```
C#  
  
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Threading.Tasks;  
using Microsoft.AspNetCore.Builder;  
using Microsoft.AspNetCore.Hosting;  
using Microsoft.Extensions.Configuration;  
using Microsoft.Extensions.DependencyInjection;  
using Microsoft.Extensions.Hosting;  
  
namespace WebApplication2  
{  
    public class Startup  
    {  
        public Startup(IConfiguration configuration)  
        {  
            Configuration = configuration;  
        }  
  
        public IConfiguration Configuration { get; }  
  
        // This method gets called by the runtime. Use this method to add services to  
        // the container.  
        public void ConfigureServices(IServiceCollection services)  
        {  
            services.AddRazorPages();  
        }  
    }  
}
```

```
// Enable routing:  
services.AddMvc((opts) => { opts.EnableEndpointRouting = false; });  
services.AddRouting();  
}  
  
// This method gets called by the runtime. Use this method to configure the  
HTTP request pipeline.  
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)  
{  
    if (env.IsDevelopment())  
    {  
        app.UseDeveloperExceptionPage();  
    }  
    else  
    {  
        app.UseExceptionHandler("/Error");  
    }  
    app.UseStaticFiles();  
    app.UseRouting();  
    app.UseAuthorization();  
    app.UseEndpoints(endpoints =>  
    {  
        endpoints.MapRazorPages();  
    });  
    // Enable routing:  
    app.UseMvcWithDefaultRoute();  
}  
}  
}
```

13. Replace the code in '**SupportApiController.cs**' with the code snippet:

```
C#  
  
using GrapeCity.Documents.Pdf.ViewerSupportApi.Controllers;  
using Microsoft.AspNetCore.Mvc;  
namespace WebApplication2  
{  
    [Route("api/pdf-viewer")]  
    [ApiController]  
    public class SupportApiController : GcPdfViewerController  
    {  
    }  
}
```

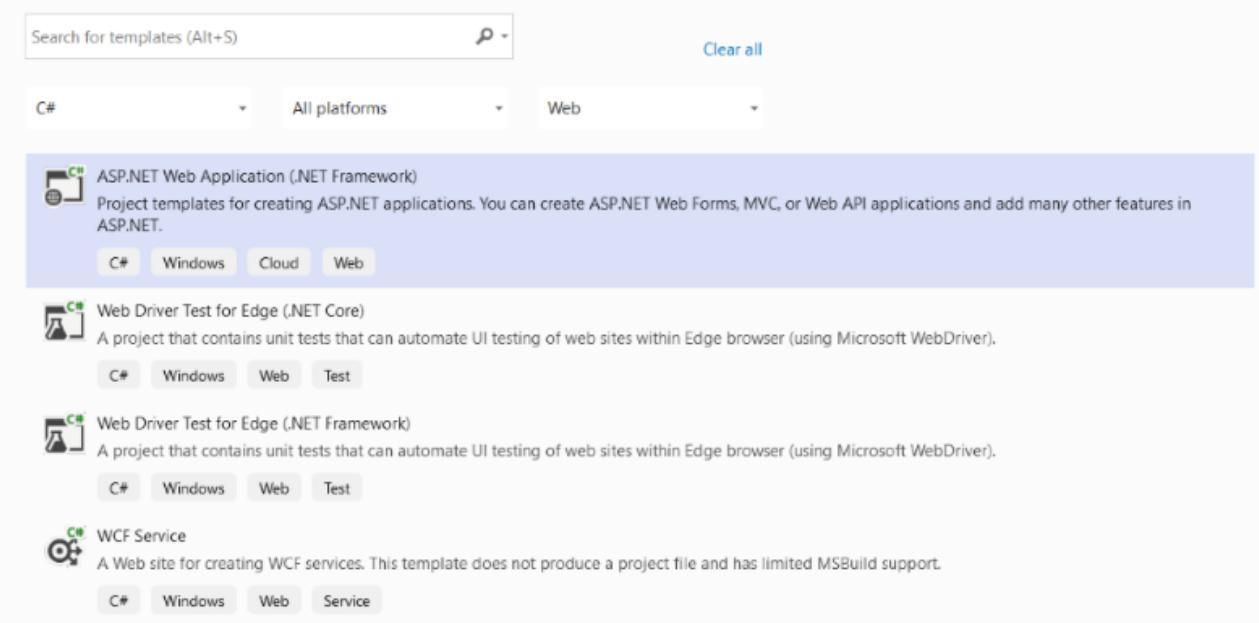
14. Build and run the application to view GcDocs PDF Viewer in your browser which contains the annotation and form editor tools to edit PDF documents.



ASP.NET WebForms Application

The steps listed below describe how to configure GcDocs PDF Viewer in an ASP.NET WebForms Application to view and edit PDF Files.

1. Open Microsoft Visual Studio and select **Create a new project | ASP.NET Web Application (.NET Framework)**.



2. In '**Configure your new project**' dialog, name the project and select '**.NET Framework 4.8**' framework.

Configure your new project

ASP.NET Web Application (.NET Framework) C# Windows Cloud Web

Project name

WebApplication1_WebForms

Location

D:\Samples



Solution name ⓘ

WebApplication1_WebForms

Place solution and project in the same directory

Framework

.NET Framework 4.8

Back

Create

3. In the '**Create a new ASP.NET web application**' dialog, select '**Web Forms**'.



Note: Make sure that '**Configure for HTTPS**' option is unchecked to avoid warnings shown by FireFox on Windows.

Create a new ASP.NET Web Application

Empty
An empty project template for creating ASP.NET applications. This template does not have any content in it.

Web Forms
A project template for creating ASP.NET Web Forms applications. ASP.NET Web Forms lets you build dynamic websites using a familiar drag-and-drop, event-driven model. A design surface and hundreds of controls and components let you rapidly build sophisticated, powerful UI-driven sites with data access.

MVC
A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.

Web API
A project template for creating RESTful HTTP services that can reach a broad range of clients including browsers and mobile devices.

Single Page Application
A project template for creating rich client side JavaScript driven HTML5 applications using ASP.NET Web API. Single Page Applications provide a rich user experience which includes client-side interactions using HTML5, CSS3, and JavaScript.

Authentication
None

Add folders & core references
 Web Forms
 MVC
 Web API

Advanced
 Configure for HTTPS
 Docker support
(Requires Docker Desktop)
 Also create a project for unit tests
WebApplication1_WebForms.Tests

Back **Create**

- Run the following command to install GcDocs PDF Viewer. Make sure that the directory location in command prompt is set to Scripts folder. The GcDocs PDF Viewer will be installed in `WebApplication1_WebForms\WebApplication1_WebForms\Scripts`:
`npm install @grapecity/gcpdfviewer`
- Replace the code in `<asp:Content>..</asp:Content>` tag in Default.aspx with below code:

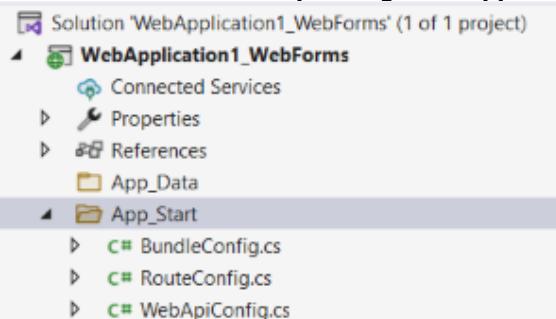
```
Default.aspx
<%@ Page Title="Home Page" Language="C#" MasterPageFile="~/Site.Master"
AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="WebApplication1_WebForms._Default" %>

<asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
    <style>
        html, body, body > form, .body-content {
            height: 100%;
            width: 100%;
        }

        #host {
            padding: 60px 20px 20px 10px;
        }
    </style>
    <div id="host"></div>
    <script src="Scripts/node_modules/@grapecity/gcpdfviewer/gcpdfviewer.js">
    </script>
    <script>
        var viewer = new GcPdfViewer("#host", { supportApi: 'SupportApi' });
        viewer.addDefaultPanels();
        viewer.addAnnotationEditorPanel();
        viewer.addFormEditorPanel();
    </script>
```

```
        viewer.beforeUnloadConfirmation = true;
        viewer.newDocument();
    </script>
</asp:Content>
```

6. In WebApplication1_WebForms project, Right click **Manage Nuget Packages** and add below packages from nuget.org:
 - o *GrapeCity.Documents.Pdf.ViewerSupportApi*
 - o *Microsoft.AspNet.WebApi.WebHost*
7. Add a new class file '**WebApiConfig.cs**' in **App_Start** folder and add the following code to it:



WebApiConfig.cs

```
using System.Web.Http;

namespace WebApplication1_WebForms.App_Start
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            // Web API configuration and services
            // Web API routes
            config.MapHttpAttributeRoutes();
            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "{controller}/{action}/{id}/{id2}/{id3}",
                defaults: new { id = RouteParameter.Optional, id2 =
RouteParameter.Optional, id3 = RouteParameter.Optional }
            );
        }
    }
}
```

8. Add the following lines of code to WebApplication1_WebForms\WebApplication1_WebForms\Global.asax.cs file. The final **Global.asax.cs** will look like below:

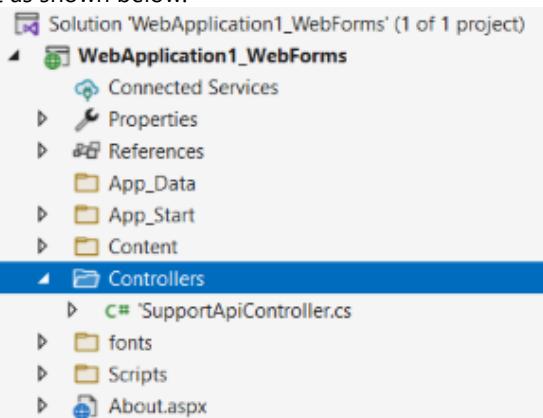
```
using System.Web.Http;
GlobalConfiguration.Configure(WebApiConfig.Register);
```

Global.asax.cs

```
using System;
using System.Web;
using System.Web.Http; //added line
using System.Web.Optimization;
using System.Web.Routing;
using WebApplication1_WebForms.App_Start;
```

```
namespace WebApplication1_WebForms
{
    public class Global : HttpApplication
    {
        void Application_Start(object sender, EventArgs e)
        {
            // Code that runs on application startup
            RouteConfig.RegisterRoutes(RouteTable.Routes);
            BundleConfig.RegisterBundles(BundleTable.Bundles);
            GlobalConfiguration.Configure(WebApiConfig.Register); //added line
        }
    }
}
```

9. Create a '**Controllers**' folder in WebApplication1_WebForms project and add a class file '**SupportApiController.cs**' to it as shown below:



10. Replace the code in 'SupportApiController.cs' with below code:

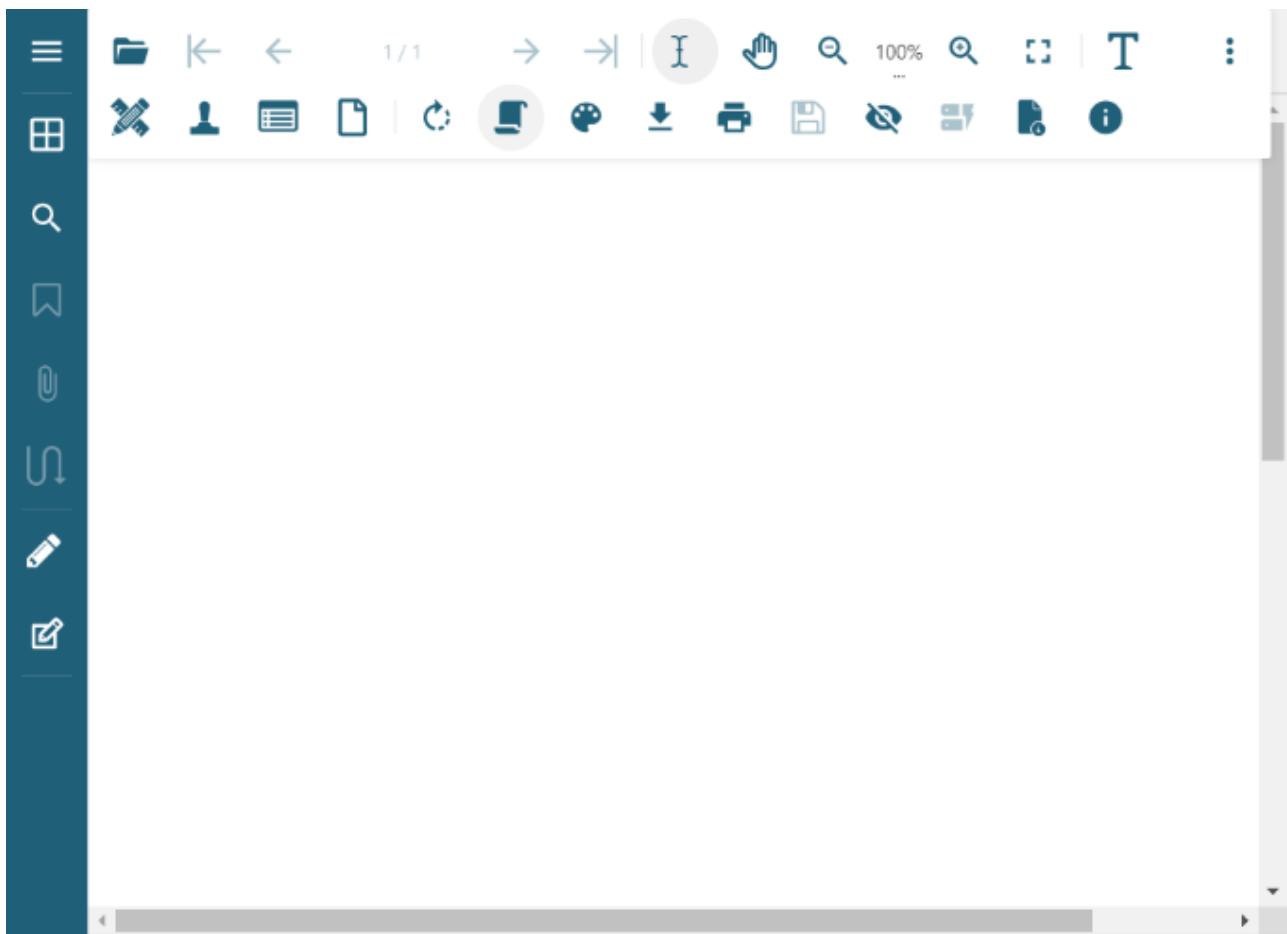
```
SupportApiController.cs
```

```
using GrapeCity.Documents.Pdf.ViewerSupportApi.Controllers;

namespace WebApplication1_WebForms.Controllers
{
    public class SupportApiController : GcPdfViewerController
    {

    }
}
```

11. Build and run the application to view GcDocs PDF Viewer with enabled editing options in your browser. You can open any PDF document and modify it.



Building and using SupportApi from sources

The SupportApi controller can be used by downloading and unzipping the [GcPDF distribution file](#). It contains a GcPdfViewerWeb folder which further contains:

- SupportApi - An ASP.NET Core web library which implements the GcDocs PDF Viewer's Support API. It can be used as it is (without any changes) in all projects. SupportAPI files are also provided as NuGet package as explained above.
- SupportApiDemo - An ASP.NET Core web application, an app that demonstrates the use of SupportApi.
- SupportApiDemo.sln - A solution file that can be used to build and run SupportApiDemo.

To use the SupportAPI from sources, instead of including GrapeCity.Documents.Pdf.ViewerSupportApi NuGet package as explained in the steps **above**, you need to copy and include SupportApi project to the application solution and then add its reference to your main project. Note that, name of the SupportApi projects is different for ASP.NET Core and ASP.NET WebForms:

- SupportApi project for ASP.NET Core: **SupportApi.csproj**
- SupportApi project for ASP.NET WebForms: **SupportApi-WebForms.csproj**

For more information, refer [Edit PDF](#) in GcDocs PDF Viewer demos.

Editors

GrapeCity Documents PDF Viewer provides the below editors which can be used to edit PDF documents:

- [Annotation Editor](#)
- [Form Editor](#)

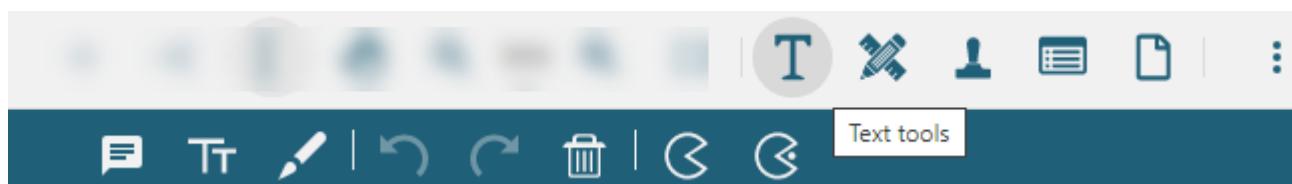
Annotation Editor

The Annotation editor in GcDocs PDF Viewer allows you to edit and review PDF documents. You can add or remove different types of annotations, comment or reply on comments and set or modify annotation properties like text, color, position, border, style etc. Along with the Annotation editor's toolbar, you can access all the annotations and various other options through the the main toolbar. It provides editing tools which allow you to quickly perform edit operations without the need of switching into full editing mode.

The editing tools displayed in the below screenshots provide the annotation options along with 'Undo, Redo and Delete' and Redact options in the quick editing toolbars.

Text Tools

On clicking the Text tools button, the quick editing toolbar appears which displays various related text annotation tools like sticky note, free text, ink annotation etc.



Draw Tools

On clicking the Draw tools button, the quick editing toolbar appears which displays various draw annotation tools like square, circle, line annotation etc.



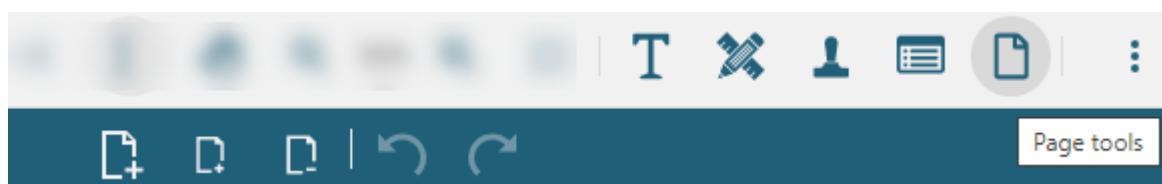
Attachments and Stamps

On clicking the Attachments and stamps button, the quick editing editor toolbar appears which displays various attachment and stamp tools like stamp annotation, file attachment , signature tool etc. Stamp annotation also supports rotation through **Rotate** property or the rotation handle attached to the annotation. For more information, see "[Stamp Rotation](#)".



Page Tools

On clicking the Page tools button, the quick editing editor toolbar appears which displays various page related options like adding a new blank document, inserting a blank page, deleting current page and perform undo or redo operations.



The Form tools button  in the above toolbar provides various form field buttons. To know more, refer [Form](#)

Editor.

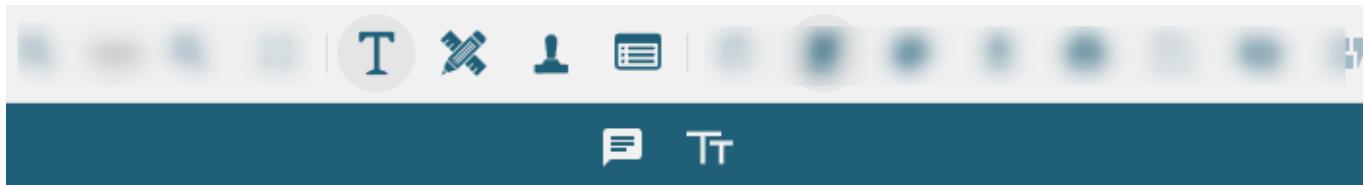
 **Note:** The editing tools (explained above) are automatically enabled in the main toolbar when `SupportApi` is configured in the project (which allows editing operations in a PDF document).

You can also configure which tools should be displayed in the second toolbar by using the `secondToolbarLayout` property as shown below:

Index.cshtml

```
const secondToolbarLayout = viewer.secondToolbarLayout;
secondToolbarLayout["text-tools"] = ['edit-text', 'edit-free-text'];
```

The output of above code will look like below:

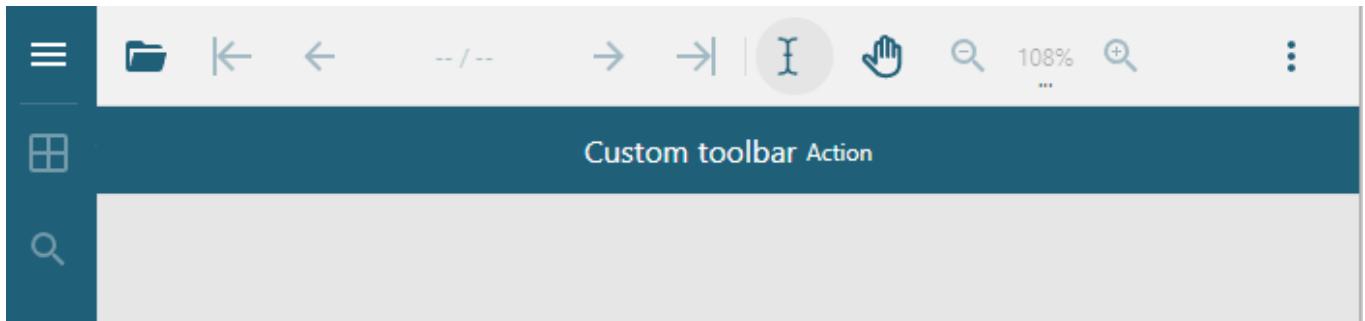


You can also customize the second toolbar by using the `render` handler for `secondToolbar` option:

Index.cshtml

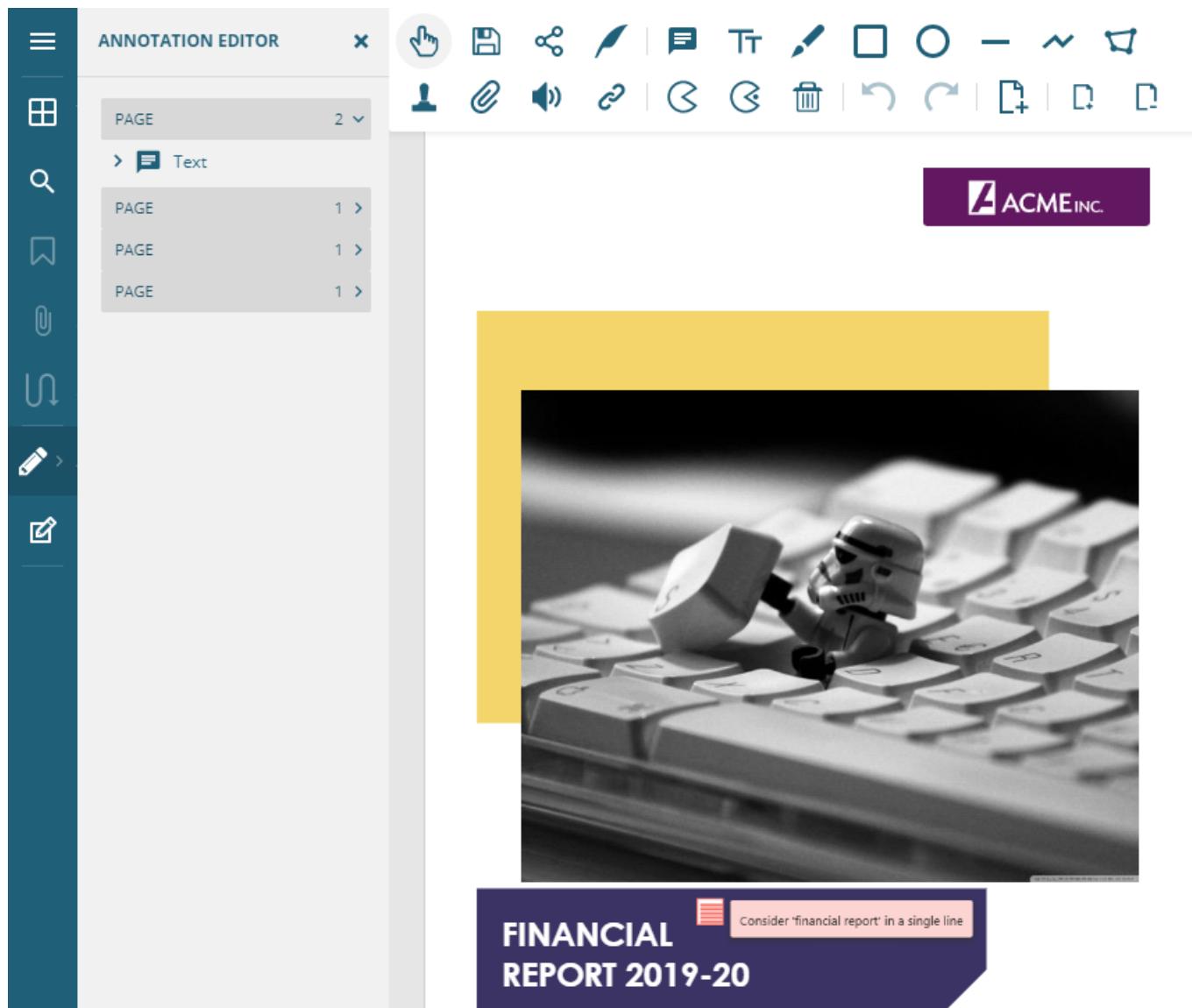
```
var React = viewer.getType("React");
// Create custom toolbar controls:
var toolbarControls = [React.createElement("label", { style: { color: "white" } },
"Custom toolbar"),
React.createElement("button", { className: "gc-btn gc-btn--accent", onClick: () => {
alert("Execute action.");
}, title: "Action title" }, "Action")];
// Register custom second toolbar for key "custom-toolbar-key":
viewer.options.secondToolbar = {
    render: function (toolbarKey) {
        if (toolbarKey === "custom-toolbar-key")
            return toolbarControls;
        return null;
    }
};
// Show custom second toolbar:
viewer.showSecondToolbar("custom-toolbar-key");
```

The output of above code will look like below:



Alternatively, you can access all the available annotations through the Annotation editor's toolbar which opens on

clicking the Annotation editor button in the side panel. The below image shows Annotation editor and its toolbar in GcDocs PDF Viewer with a PDF document containing a Text annotation.



The different toolbar buttons are described as below. To know more about different annotations, refer [Annotation Types](#).

Name	Toolbar Icons	Description
Select		Select an annotation added on PDF
Signature Tool		Adds graphical signatures on the PDF
Sticky Note		Adds text or sticky notes on the PDF
Free Text Annotation		Adds a note that is always visible on the PDF

Ink Annotation		Draws free-hand scribble on the PDF
Square Annotation		Adds a rectangle shape on PDF
Circle Annotation		Adds a circle shape on the PDF
Line Annotation		Adds a straight line on the PDF
PolyLine Annotation		Adds closed or open shapes of multiple edges on the PDF
Polygon Annotation		Adds a polygon on the PDF
Stamp Annotation		Adds image on the PDF
File Attachment Annotation		Attaches a file to the document, which will be embedded in the PDF
Sound Annotation		Adds sound (.au, .aiff, or .wav format) imported from a file or recorded from the computer's microphone
Link Annotation		Adds link on the PDF
Delete Annotation Button		Deletes the annotation
Redact Annotation		Marks region on PDF document to be redacted
Apply Redact Annotation		Applies redact to all regions marked for redact

Note: All the above mentioned annotations are supported in GcDocs PDF Viewer. The below annotations are explained in detail in following topics:

- [Redact Annotation](#)
- [Stamp Annotation](#)
- [Link Annotation](#)

Apart from the different types of annotations described above, GcDocs PDF Viewer also provides some general editing features while working with PDF documents. They are explained as below:

Toolbar Icons	Description
	Undo changes

	Redo changes
	Saves the modified document on client
	Creates a new blank document
	Inserts a blank page
	Deletes current page

 **Note:** You can view the original PDF document at any point of time by using the 'Hide Annotations' button on the main toolbar.

You can also insert a blank page in a PDF document and set its size by using the **newPage** method. Alternatively, you can only set the size of an existing page using **setPageSize** method as shown below:

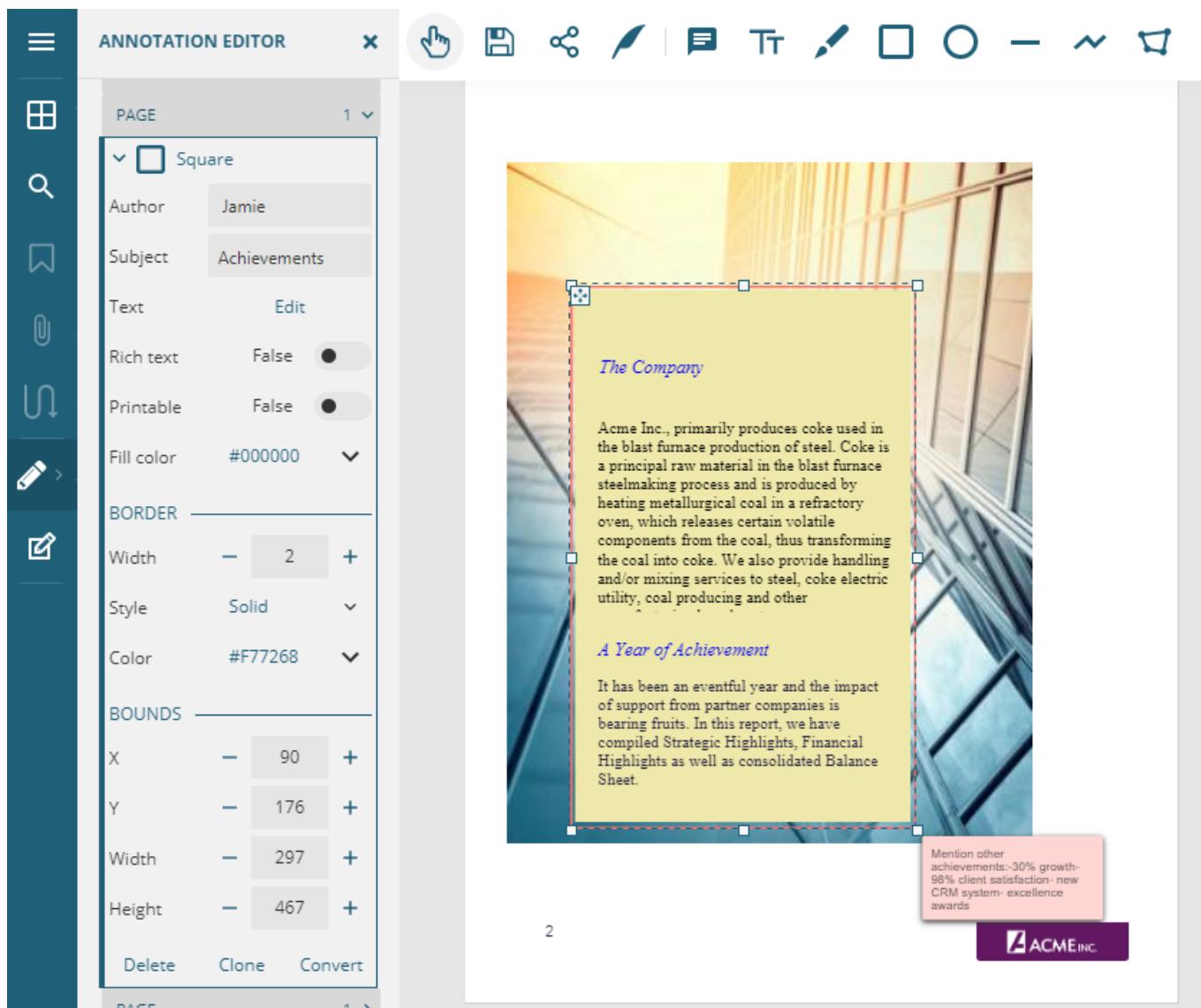
Index.cshtml

```
viewer.open("AcmeFinancialReport.pdf");
viewer.onAfterOpen.register(()=>
{
    //Add new page at second position and set its size
    viewer.newPage({height:400, pageIndex:1, width:300});
    //Set page size for the first page
    viewer.setPageSize(0, { width: 300, height: 500 });
});
```

Property Panel of Annotation Editor

When you click the Annotation editor icon in the left vertical panel, the Property panel of the Annotation editor becomes visible. The Property panel displays the list of all the annotations page-wise in your document.

It also allows you to set or modify properties of any annotation in the document like its text, color, border location etc. For eg. The image below shows the properties of a square annotation in the Property panel.



Enable Annotation Editor in GcDocs PDF Viewer

The Annotation editor is displayed by default in GcDocs PDF Viewer, by enabling the AnnotationEditorPanel in the viewer using code:

Index.cshtml

```
<script>
    var viewer = new GcPdfViewer("#host", { supportApi: 'SupportApi/GcPdfViewer' });
    viewer.addDefaultPanels();
    viewer.addAnnotationEditorPanel();
    viewer.beforeUnloadConfirmation = true;
    viewer.open("Home/GetPdf");
</script>
```

To customize the annotation options in GcDocs PDF Viewer, add the following lines of code in the class file where you load the PDF in the Viewer:

C#

```
public static GcPdfViewerSupportApiDemo.Models.PdfViewerOptions PdfViewerOptions
{
    get => new GcPdfViewerSupportApiDemo.Models.PdfViewerOptions(
        GcPdfViewerSupportApiDemo.Models.PdfViewerOptions.Options.AnnotationEditorPanel |
        GcPdfViewerSupportApiDemo.Models.PdfViewerOptions.Options.ActivateAnnotationEditor,
        annotationEditorTools: new string[] { "edit-select", "$split", "edit-text",
        "edit-free-text", "$split", "edit-erase", "$split", "edit-redact", "edit-redact-
        apply", "$split", "edit-undo", "edit-redo", "save" });
}
```

Review PDF document using Annotation editor

Follow the below steps to review a PDF document using the annotation editor in GcDocs PDF Viewer:

1. Configure the [GcDocs PDF Viewer for editing](#) PDF documents.
2. Run the application and open the PDF you want to review using the Open button.
3. Open the Annotation Editor using the second last icon on the left vertical toolbar.
4. Choose any annotation from the different annotations available in the toolbar at the top.
5. Use that annotation to mark a correction in your document.
6. Set annotation properties from the Property panel like color, border etc.
7. Add a Text annotation and type the final review text.
8. Close the Annotation editor and go back to View mode.

The annotations are successfully added annotations to the document. You can also view the list of all the annotations in the property panel of Annotation Editor.

The screenshot shows the GrapeCity Documents PDF Viewer interface. On the left, there is an 'ANNOTATION EDITOR' panel for a 'Circle' annotation. The properties shown are: Author (Anonymous), Subject (<empty>), Text (Edit), Rich text (False), Printable (False), and Fill color (#000000). Below these are sections for BORDER (Width: 1, Style: Solid, Color: #000000) and BOUNDS (X: 177, Y: 146, Width: 107, Height: 19). At the bottom of the editor are buttons for Delete, Clone, and Convert. The main area displays a document page titled 'Documents for PDF, .NET Edition' with a page number of 122. The page content includes a heading 'GrapeCity Documents PDF Viewer' and a paragraph about wetlands. A photograph of a wetland landscape is also visible.

For more information, refer [Annotation Editor](#) in GcDocs PDF Viewer demos.

Redact Annotation

The annotation editor allows you to add Redact annotation. A redact annotation marks an area under which all the existing content should be removed from PDF document. Redact annotation, unlike other annotations, is applied in two phases:



Redact (erase) a region: It highlights the area under which the content needs to be redacted or removed.

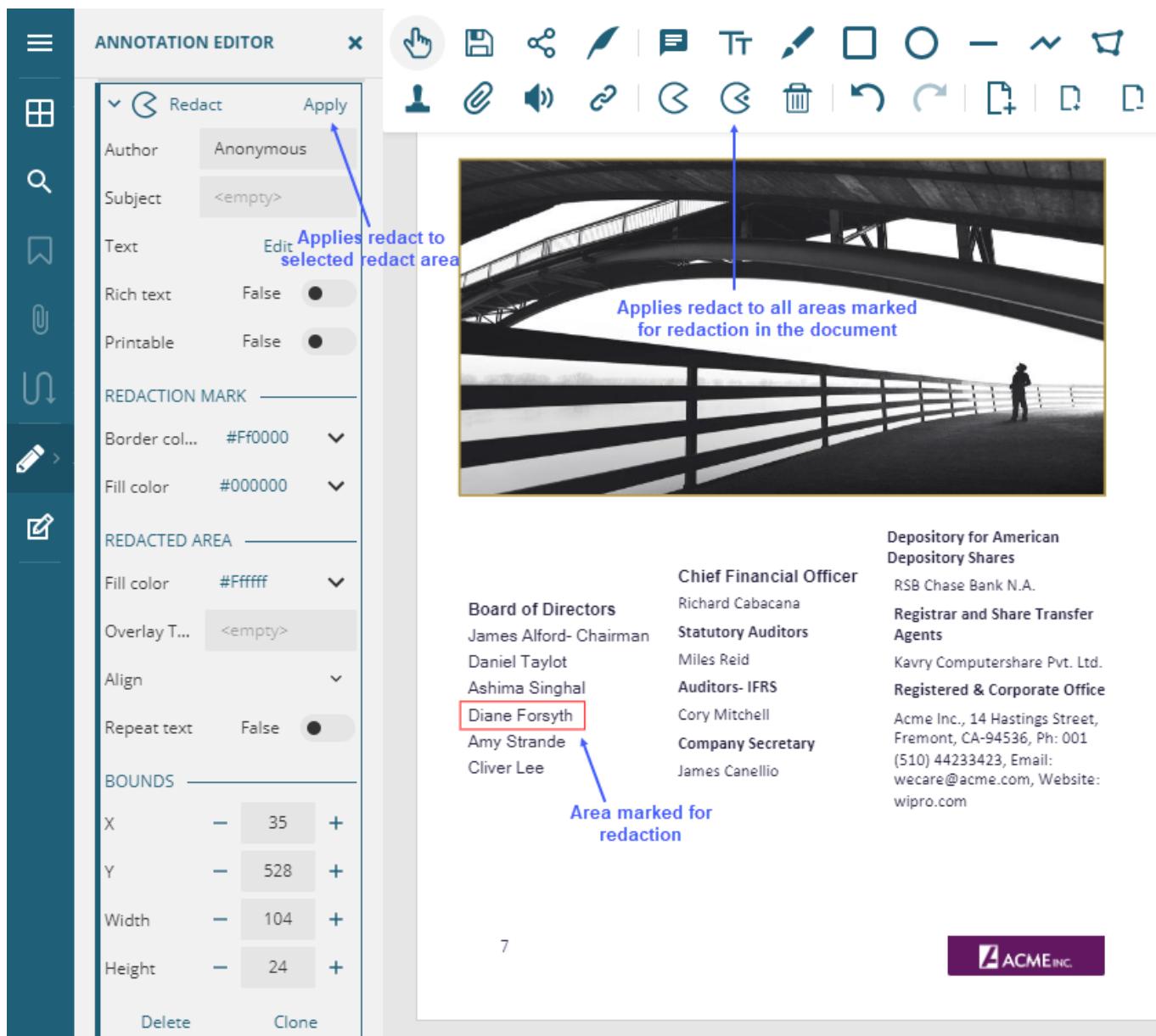


Apply all Redacts: It applies redaction, that is, removes all the content which is marked for redaction.



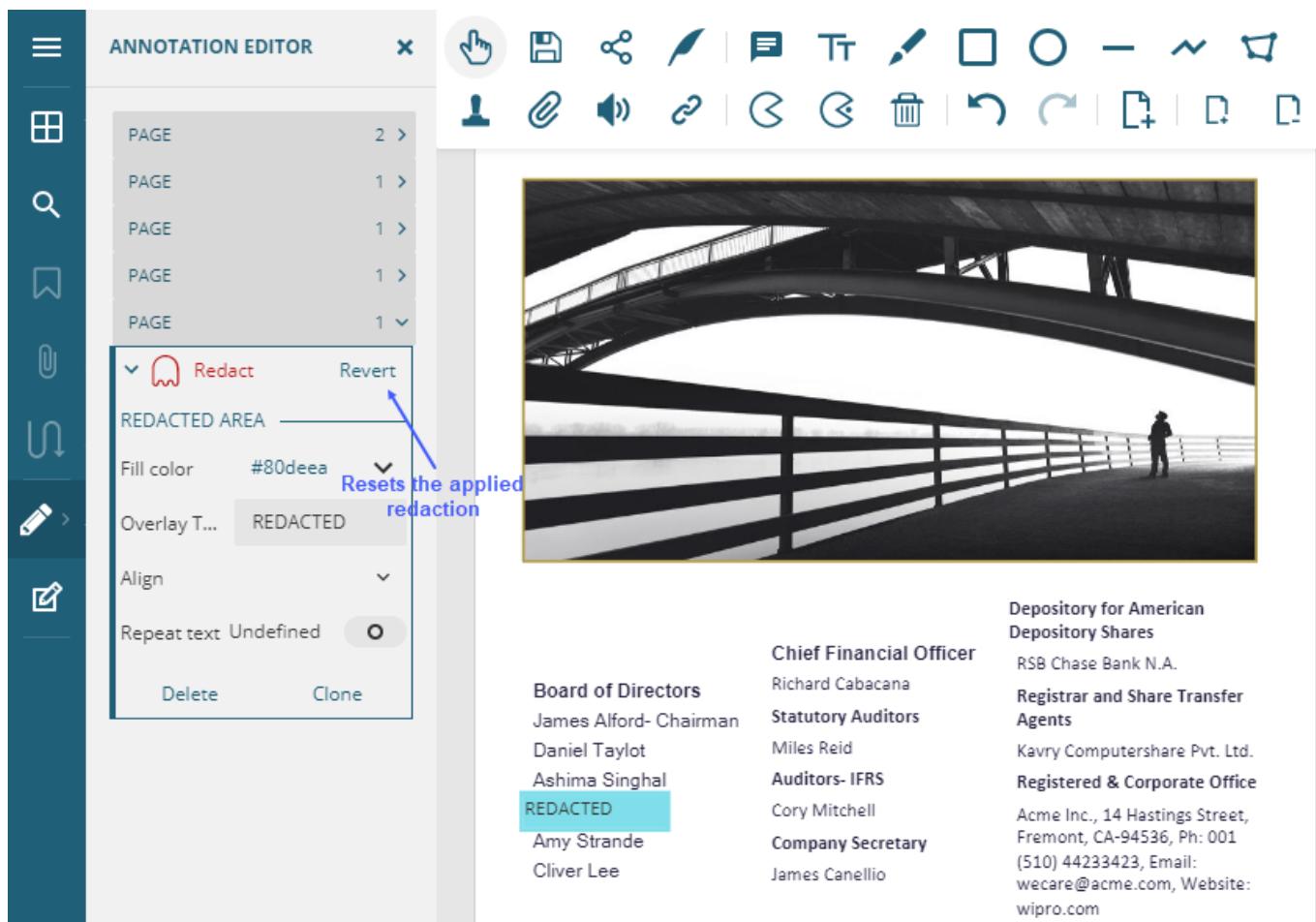
Note: The Redact annotation options can also be accessed through quick editing toolbar. For more information, refer [Annotation Editor](#).

The below image shows the area marked for redaction in a PDF document. The properties in 'Redaction Mark' can be defined to specify how the highlighted area marked for redaction should look before the redaction has been applied.



After the redaction is applied, the content marked for redaction is removed. However, you can revert the redaction and view the original content by selecting the 'Reset redact' button from the Properties panel.

You can also define the 'Overlay text' or 'Fill Color' in the 'Redacted area' settings whose text or Fill color will appear in place of the redacted content.



When a PDF document with applied redact annotations is saved, the annotations and the content does not exist in the document anymore. They are replaced with the 'Redacted Area' properties. Also, as the content is removed, the redacted area cannot be used to copy or paste the content under redaction into other documents.

Stamp Annotation

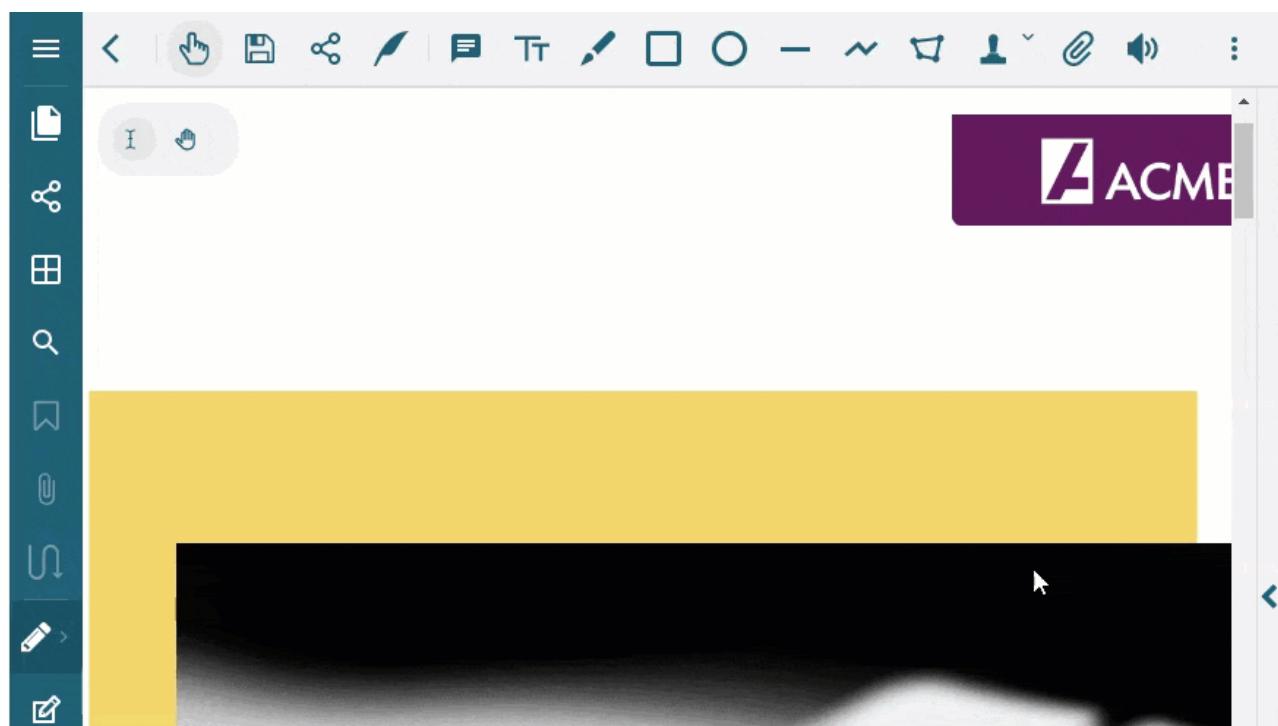
The Annotation Editor provides Stamp annotation in its toolbar which allows you to add predefined stamps and custom images in PDF documents. The below image shows the Stamp annotation button:



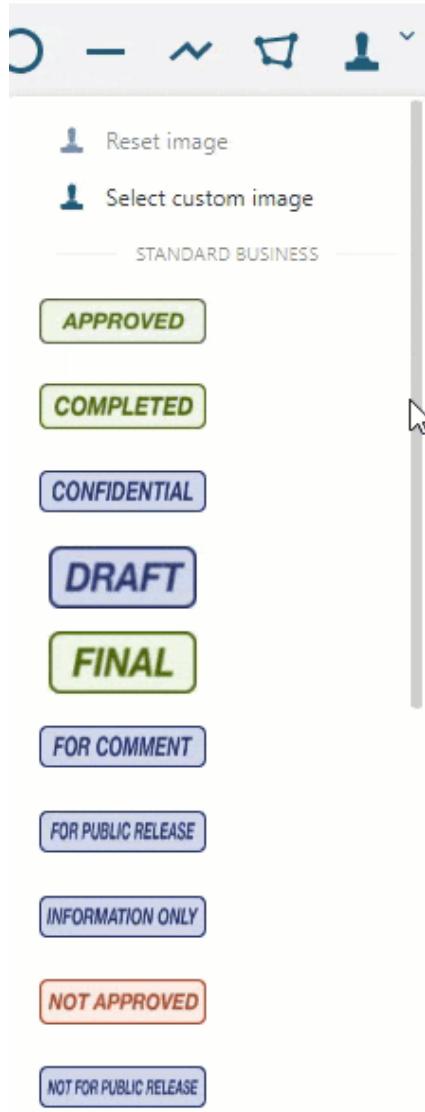
Note: The Stamp annotation option can also be accessed through quick editing toolbar. For more information, refer [Annotation Editor](#).

Add Predefined Stamps

The below GIF demonstrates how to add a predefined stamp in a PDF document by using the Stamp annotation's dropdown button. As can be observed, when a predefined stamp is added, the stamp button shows a preview of the selected stamp image. However, you can choose 'Reset Image' option to revert back to the original stamp button.



The below image shows all the predefined stamps available in GcDocs PDF Viewer:



You can also specify your own set of predefined stamps on the client side by using the **stampCategories** option as shown below:

Index.cshtml

```
var viewer = new GcPdfViewer("#root", {
    stamp: {
        stampCategories: [
            {
                name: 'Nature', stampImageUrls: ['ferns.jpg',
                    'ducky.png']
            },
            {
                name: 'Nature 2', stampImageUrls: ['fiord.jpg', 'sky.jpg',
                    'fire.jpg']
            }
        ]
    }
});
```

The output of the above code will look like below where the specified stamps will be available in the Stamp annotation's dropdown button:



Reset image

Select custom image

NATURE



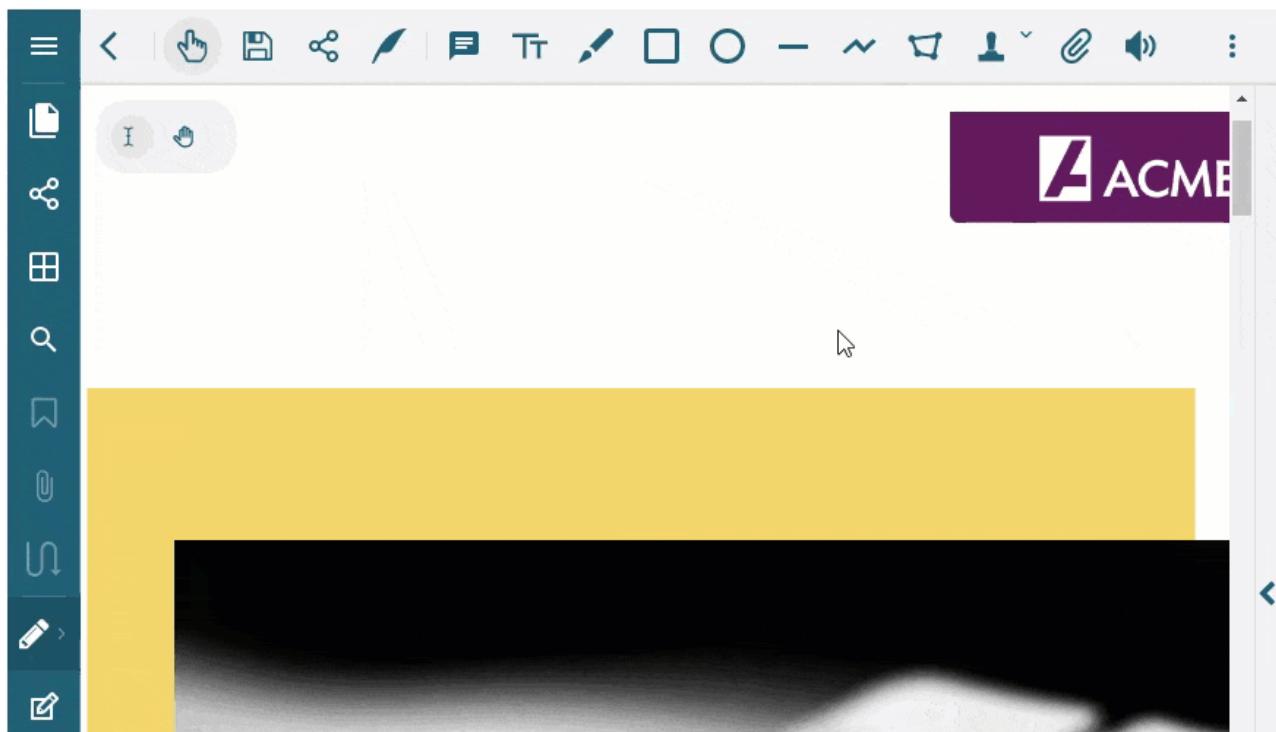
NATURE 2



Similarly, you can also specify your own set of predefined stamps on the server side. To know more, refer [Custom Predefined Stamps on server side](#).

Add Custom Images

The below GIF demonstrates two different ways to add images from your system in a PDF document, either by using the Stamp Annotation button or 'Select custom image' option from the dropdown:

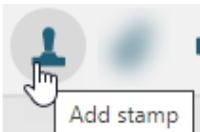


In case you want to add only custom images in a PDF document and no predefined stamps, you can remove the dropdown for adding predefined stamps (visible by default).

Index.cshtml

```
var viewer = new GcPdfViewer("#root", {
    stamp: {
        stampCategories: false
    }
});
```

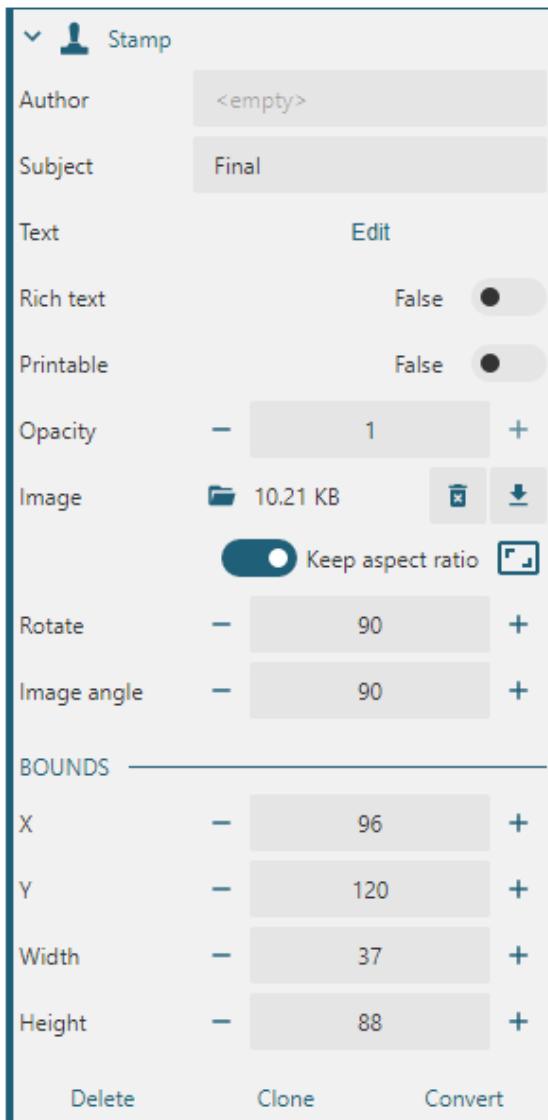
The above code will remove the dropdown of Stamp annotation button:



Property Panel of Stamp Annotation

Once a custom image or a predefined stamp is added in a document, the stamp annotation is added in the property panel as well. You can rotate the annotation, change the size and coordinates of image, download or remove the image by using property panel options.

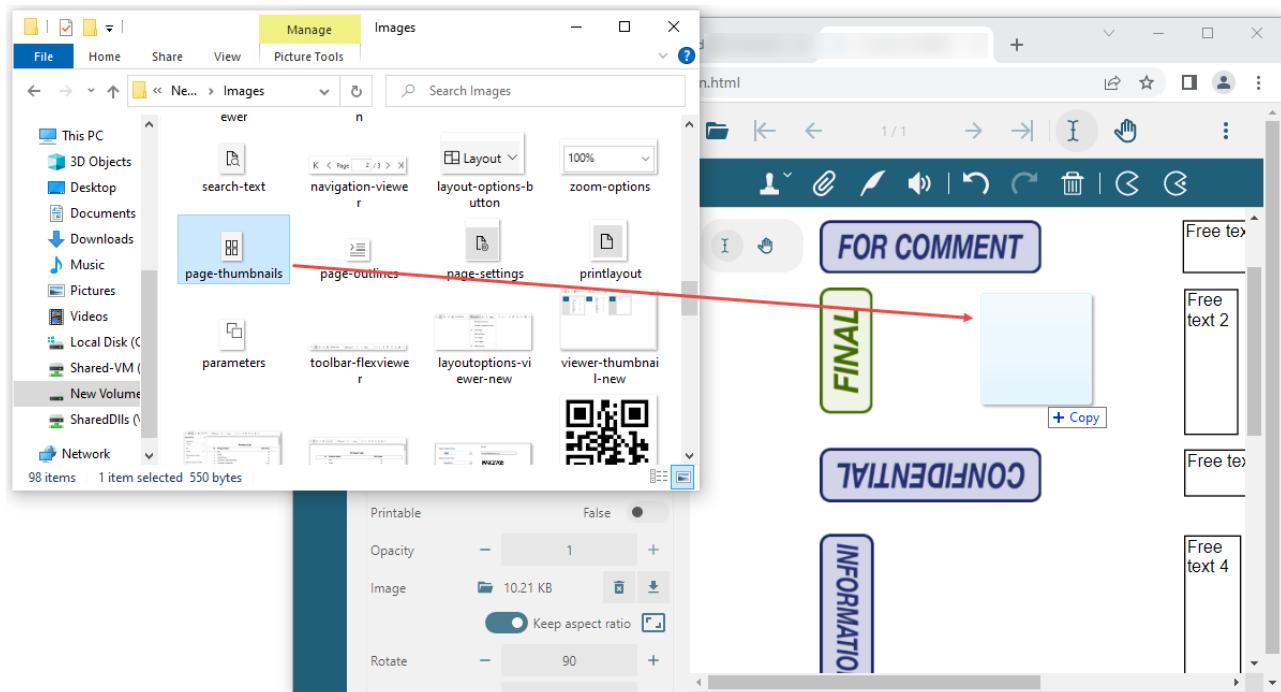
The "Keep aspect ratio" toggle button is 'on' by default and preserves the aspect ratio while resizing. You can hold down the Shift key to toggle the "Keep aspect ratio" button temporarily.



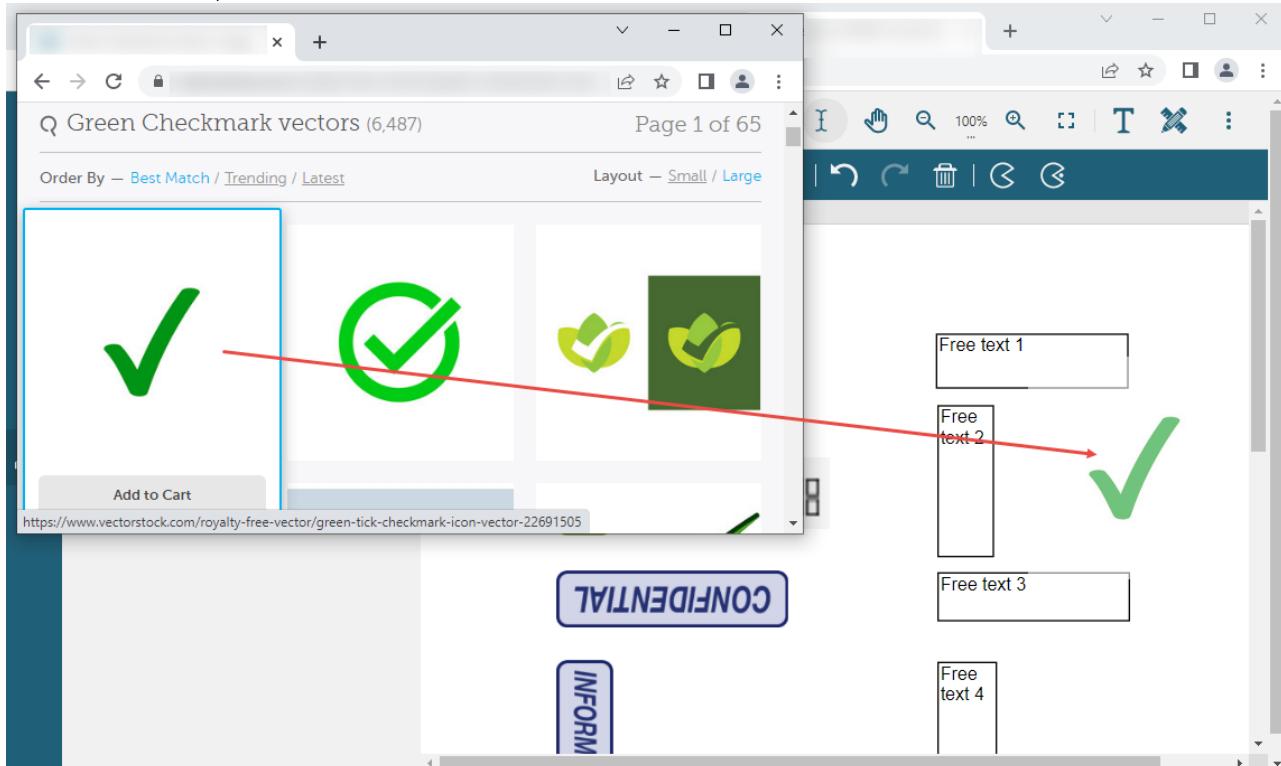
Add Custom Images using Drag and Drop Operation

A custom image can also be added by using drag and drop operation:

1. From local system, as shown below:



- From a remote URL, as shown below:

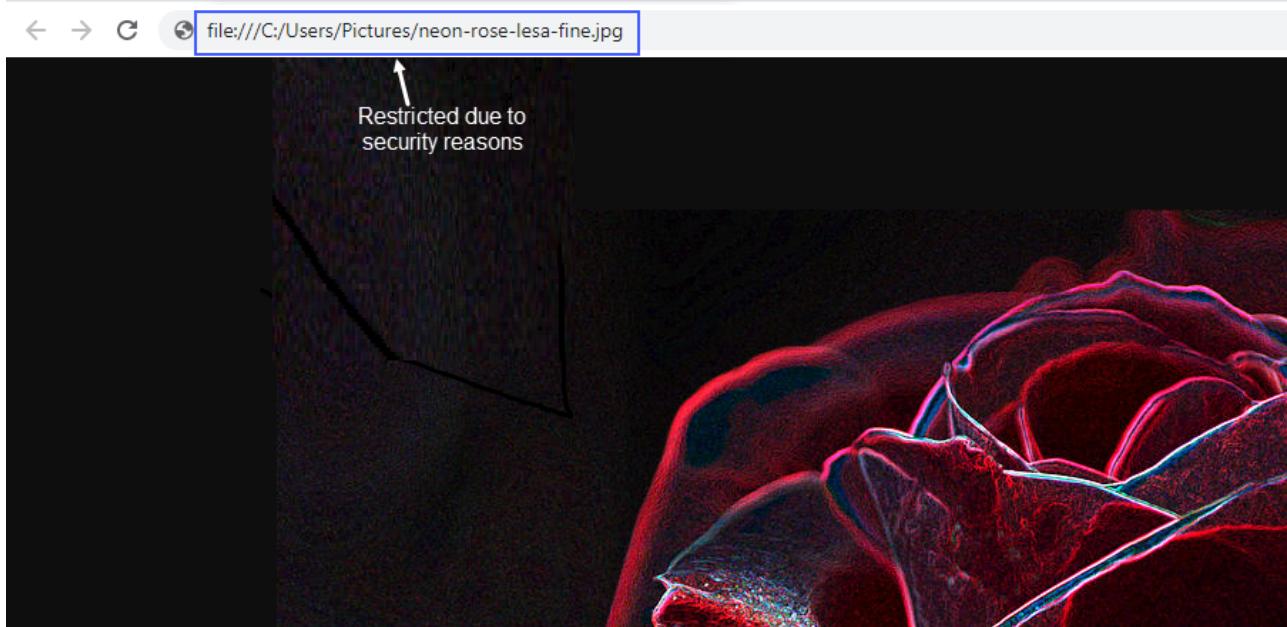


 **Note:** While performing drag and drop operation, the supported image format depends on the browser type. However, all common image formats are supported, to learn more about image formats supported by different browsers, please check [here](#).

For more information, refer [Stamp Annotations](#) in GcDocs PDF Viewer demos.

Limitation

1. A local image opened in browser by using local file system URL is not supported for drag and drop operation due to browser's security reasons.



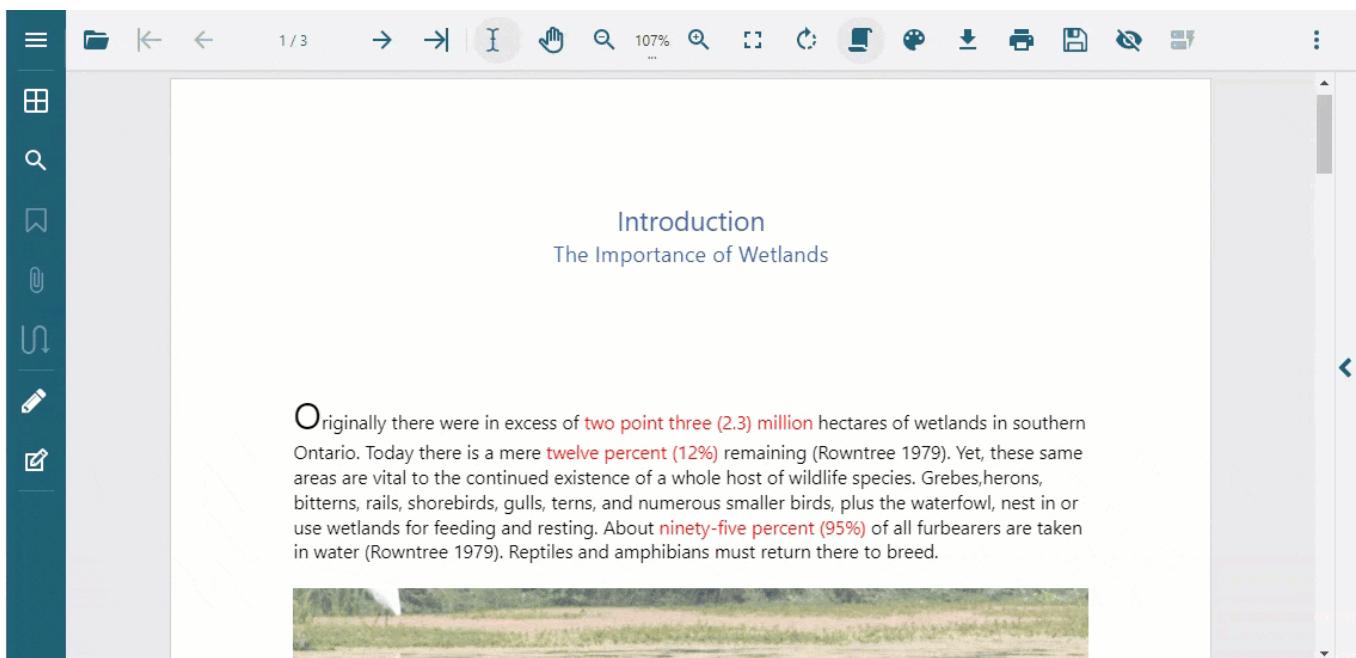
2. Drag and drop operation from a remote URL can be blocked by the host server's cross-domain policy.

Link Annotation

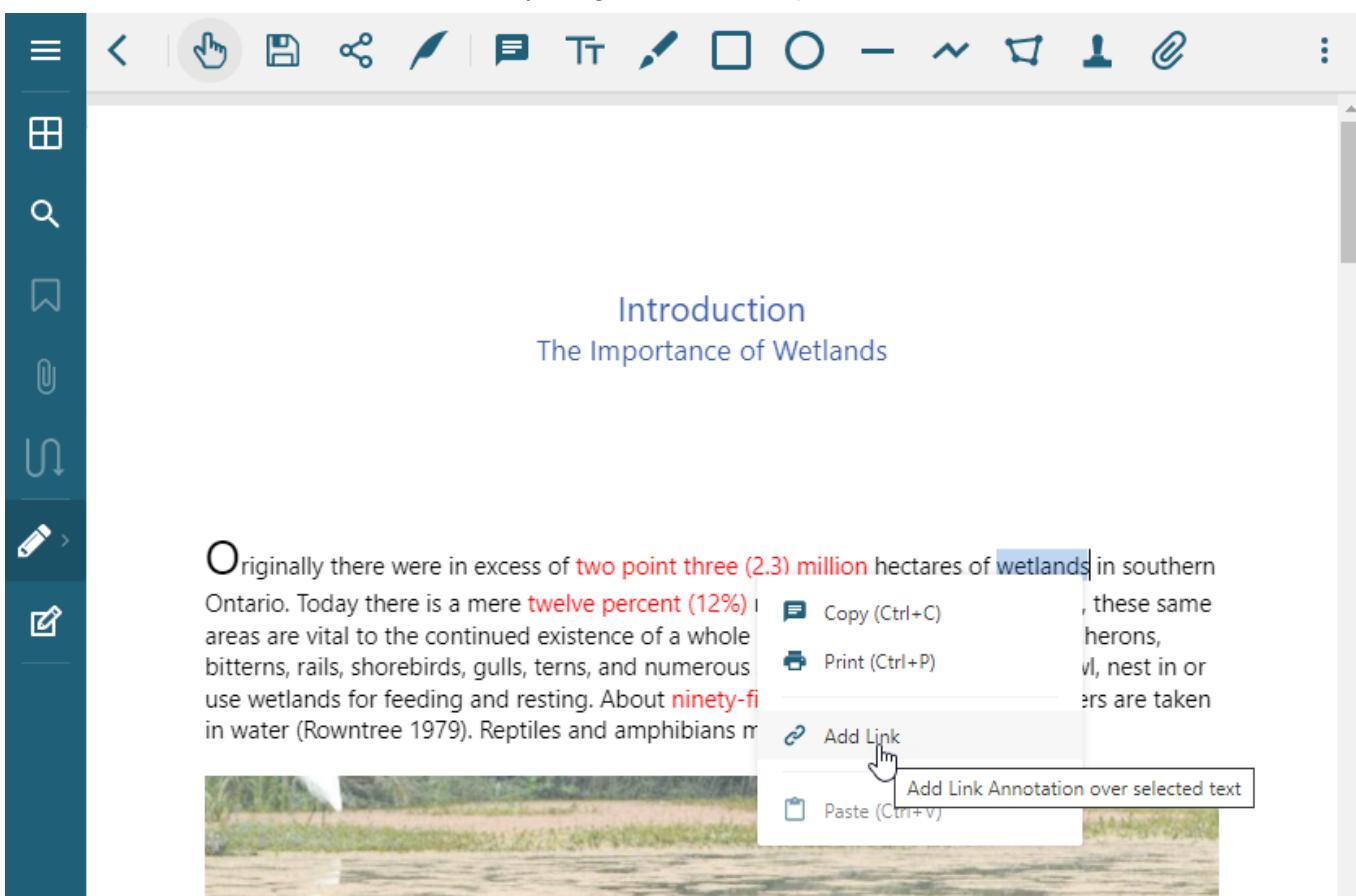
The Annotation Editor provides Link annotation in its toolbar which allows you to add links in a PDF document. The links can be added to any area of a PDF document like text, image or over any other existing annotation or form fields. The below image shows the Link annotation button:



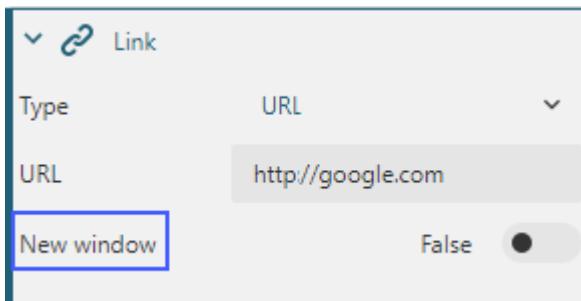
The below GIF demonstrates how to add a link in a PDF document by using Link Annotation. When you hover over the linked area, the link is displayed as a tooltip:



You can also add a link over a selected text, by using context menu option 'Add Link' as shown below:



By default, all external links are opened in the same browser window unless the 'New window' option in the property panel is set to true.



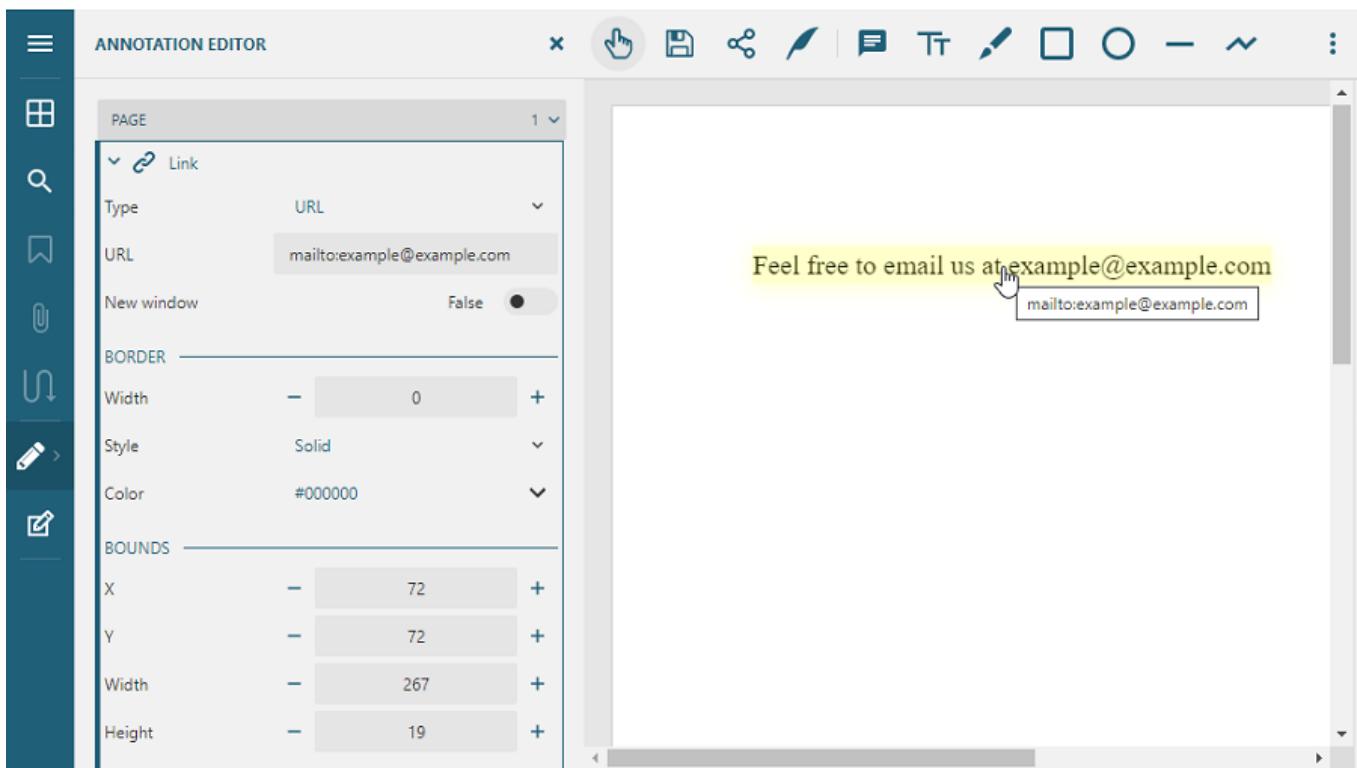
Types of Link Annotations

GcDocs PDF Viewer allows you to add following types of links in PDF documents by selecting from the 'Type' dropdown in properties panel.

URL: Creates hyperlink to web pages, email addresses or anything a URL can address. The below image shows adding a URL as a link to search 'Wetlands':

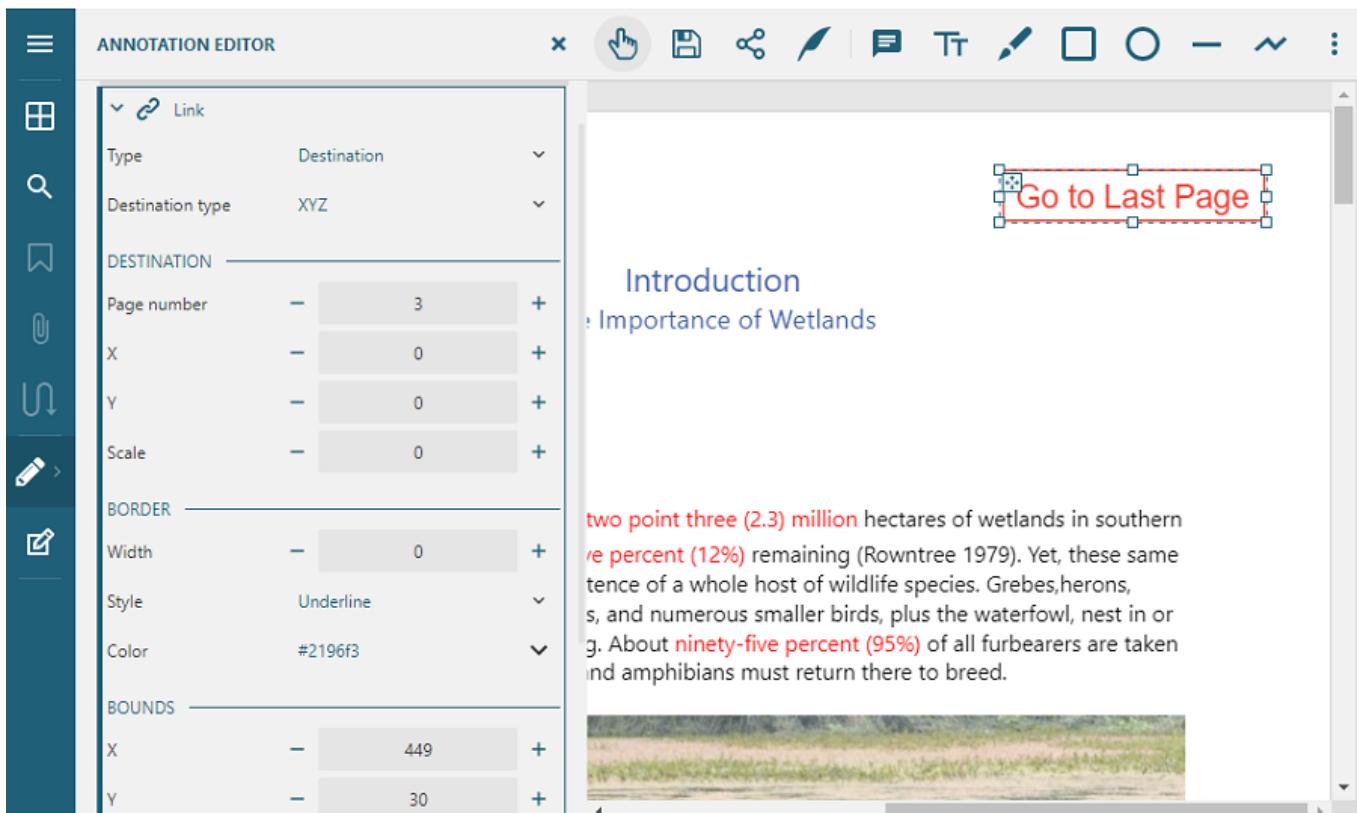
The screenshot shows the 'Annotation Editor' interface in GcDocs PDF Viewer. On the left is a toolbar with various annotation tools. The main area shows a page with text and a small image of birds. A red box highlights a portion of the text. On the right, the 'Annotation Editor' panel is open, showing the 'Link' properties. The 'Type' is set to 'URL' and the 'URL' field contains 'https://www.bing.com/search?q=Wetland'. The 'New window' checkbox is checked. Below the properties are sections for 'BORDER' and 'BOUNDS', with numerical values for width, height, X, and Y coordinates. The background of the editor shows a portion of the document page with text and an image of birds.

The below image shows adding a URL as a link to open email:

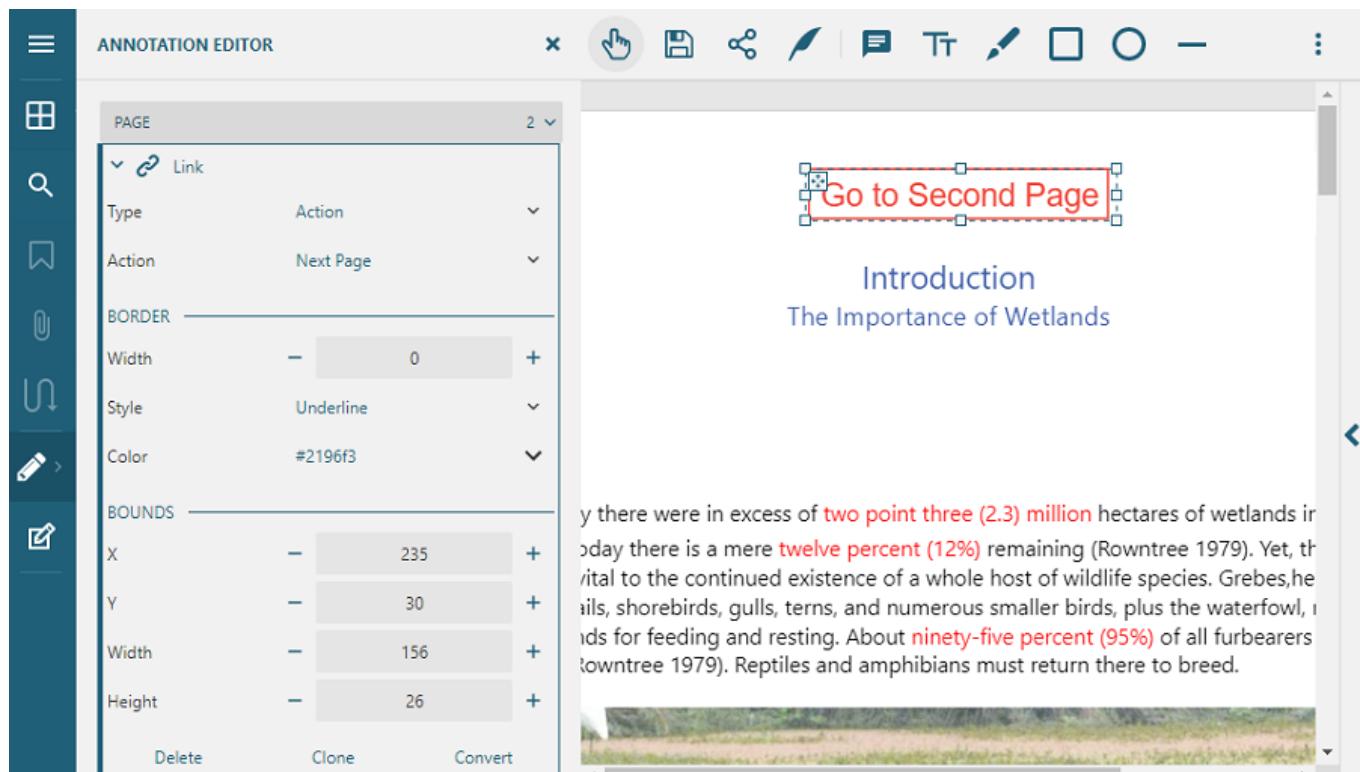


Destination: Specifies the explicit destination in a PDF document to be linked. It can point to any area or page of the PDF document with various options like fit to page, magnified content, specific positioning of vertical or horizontal coordinates etc. The 'Destination type' provides all the 8 options to set an explicit destination as described in the section 12.3.2.2 of [PDF specification 1.7](#).

The below image shows how to specify a link to open third page of the PDF document.



Action: Defines navigation actions to be performed such as First page, Last page, Next page, Previous page, Go Back and Go Forward. The below image shows a link specifying the action to navigate to next page.

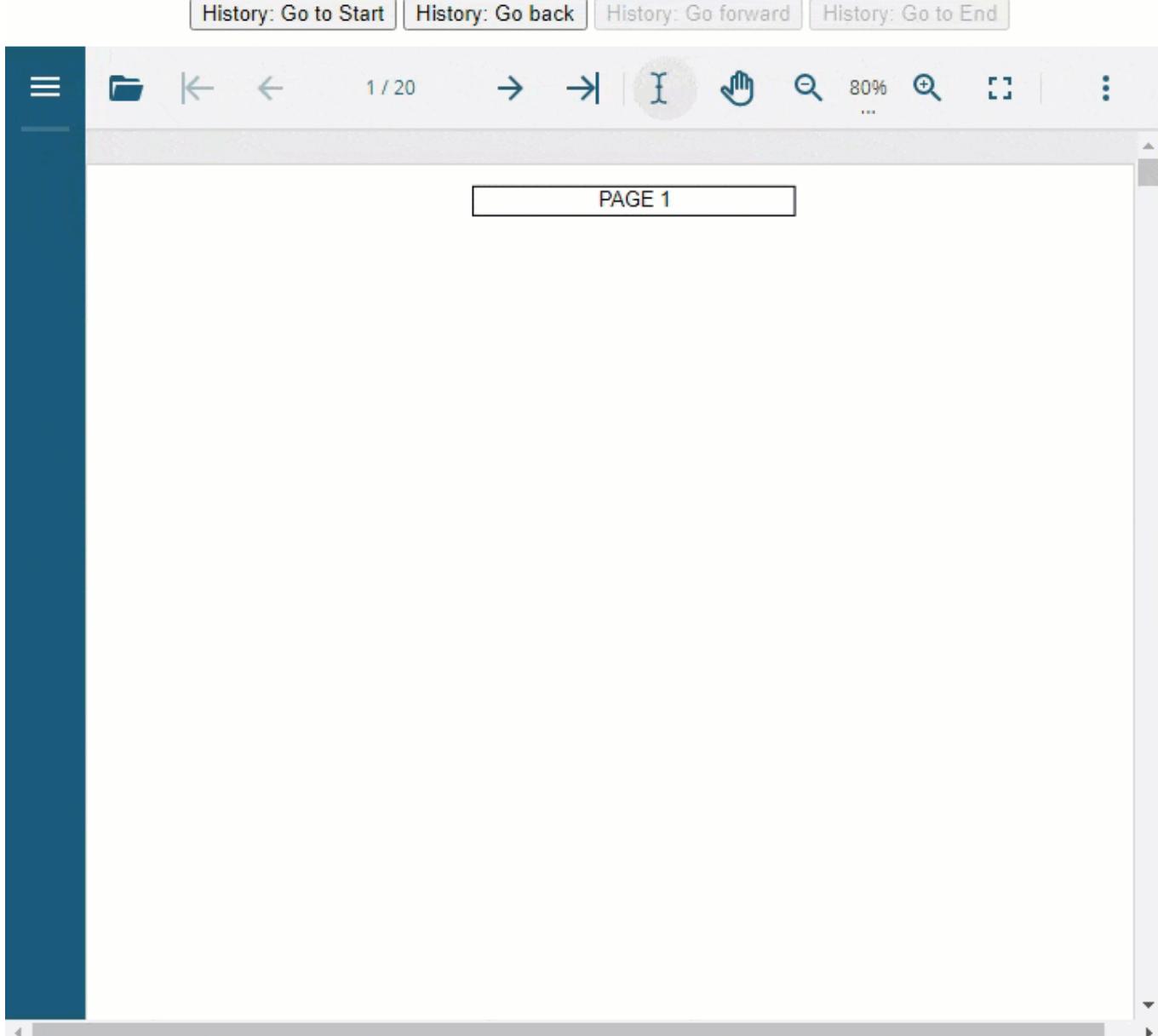


The Go Back and Go Forward actions allow you to navigate the view history of a PDF document similar to a browser's go back and go forward actions. GcDocs PdfViewer provides the below methods to navigate through the document history:

Index.cshtml

```
// Move back in the document history  
viewer.goBack();  
// Move forward in the document history  
viewer.goForward();
```

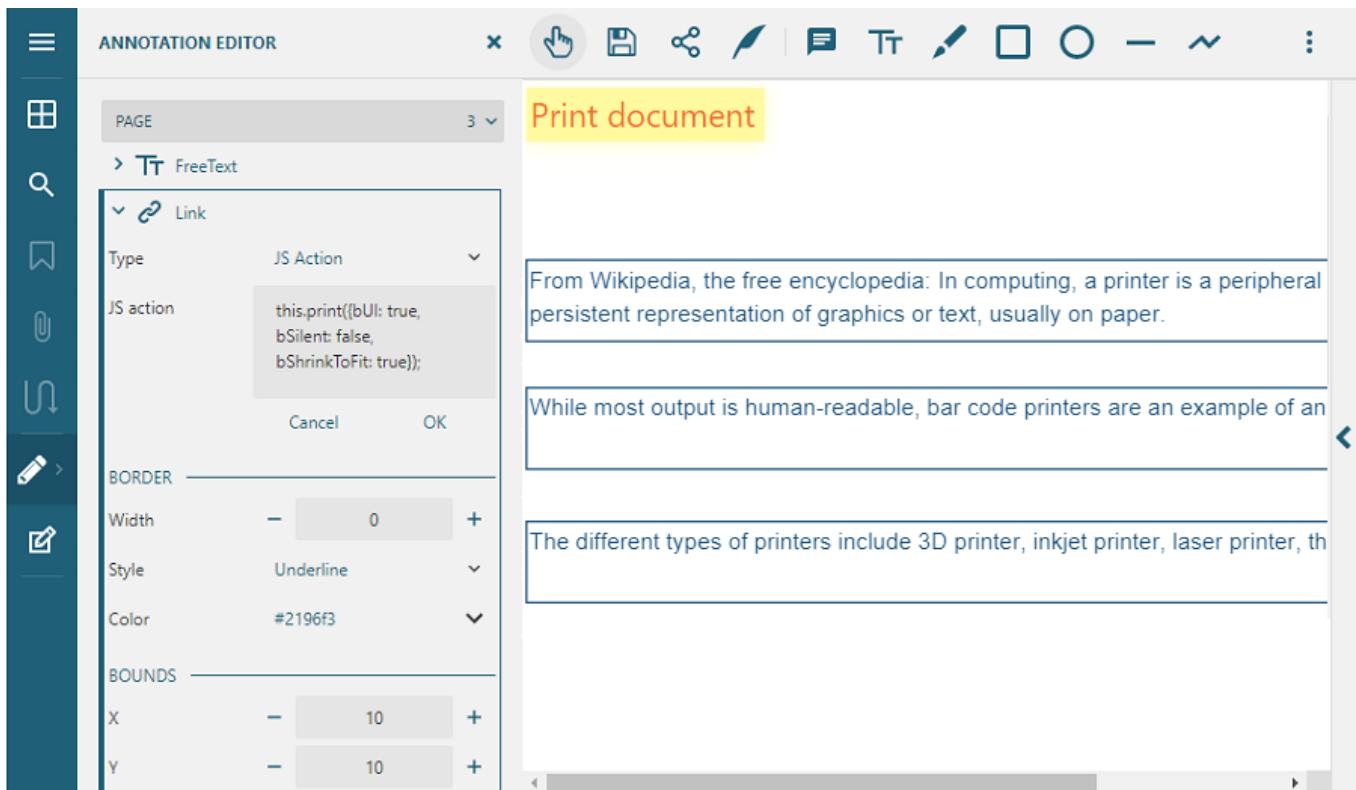
The below GIF shows the functionality of Go to Start, Go Back, Go Forward and Go to End actions using the Link annotation:



GcDocs PDF Viewer uses the browser's History API to remember the position changes while navigating through the PDF document. You can also use the standard web browser's History API. For more information, refer [here](#).

 **Note:** In order to create a history point while scrolling through a PDF document, you need to stay on a page for more than 1 second as the current view position is not saved to history immediately.

JavaScript Action: Specifies a JavaScript action which is performed depending on the script, like various interactive form fields in the document may update their values or change their visual appearances. The below image shows a specified script to print the PDF document.



Create Link Annotation using Code

To add a URL as a link in PDF document, use the following code:

Index.cshtml

```
//add link annotation
var linkAnnotation = { annotationType: 2, linkType: 'url', url: 'http://google.com',
rect: [0, 0, 200, 40] };
viewer.addAnnotation(0, linkAnnotation);
var viewer = new GcPdfViewer('#root', { externalLinkTarget: 'blank' });
```

The **externalLinkTarget** option specifies where to open the linked document. The possible values are:

- blank: Opens the linked document in a new window or tab
- self: Opens the linked document in the same frame as it was clicked (default value)
- parent: Opens the linked document in the parent frame
- top: Opens the linked document in the full body of the window

To open all links in a new window (or browser tab) by default, set the **externalLinkTarget** option to 'blank' by using following code:

Index.cshtml

```
var viewer = new GcPdfViewer("#host", {
    externalLinkTarget: 'blank',
    supportApi: 'api/pdf-viewer'
});
```

For more information, refer [Link Annotations](#) in GcDocs PDF Viewer demos.

Form Editor

The Form Editor in GcDocs PDF Viewer allows you to edit, design, fill and submit PDF forms. You can create new forms or modify existing ones by adding or removing different types of form fields and setting or modifying form field's properties like Name, ReadOnly, Max Length, Value, Bounds, Border etc.

Along with the Form editor's toolbar, you can access all the form fields and various other options directly through the main toolbar. It provides Form tools button which displays various form field buttons like text field, password field, radio button, checkbox etc. As can be observed, the 'Undo, Redo and Delete' and Redact options are also available in the quick editing toolbar.

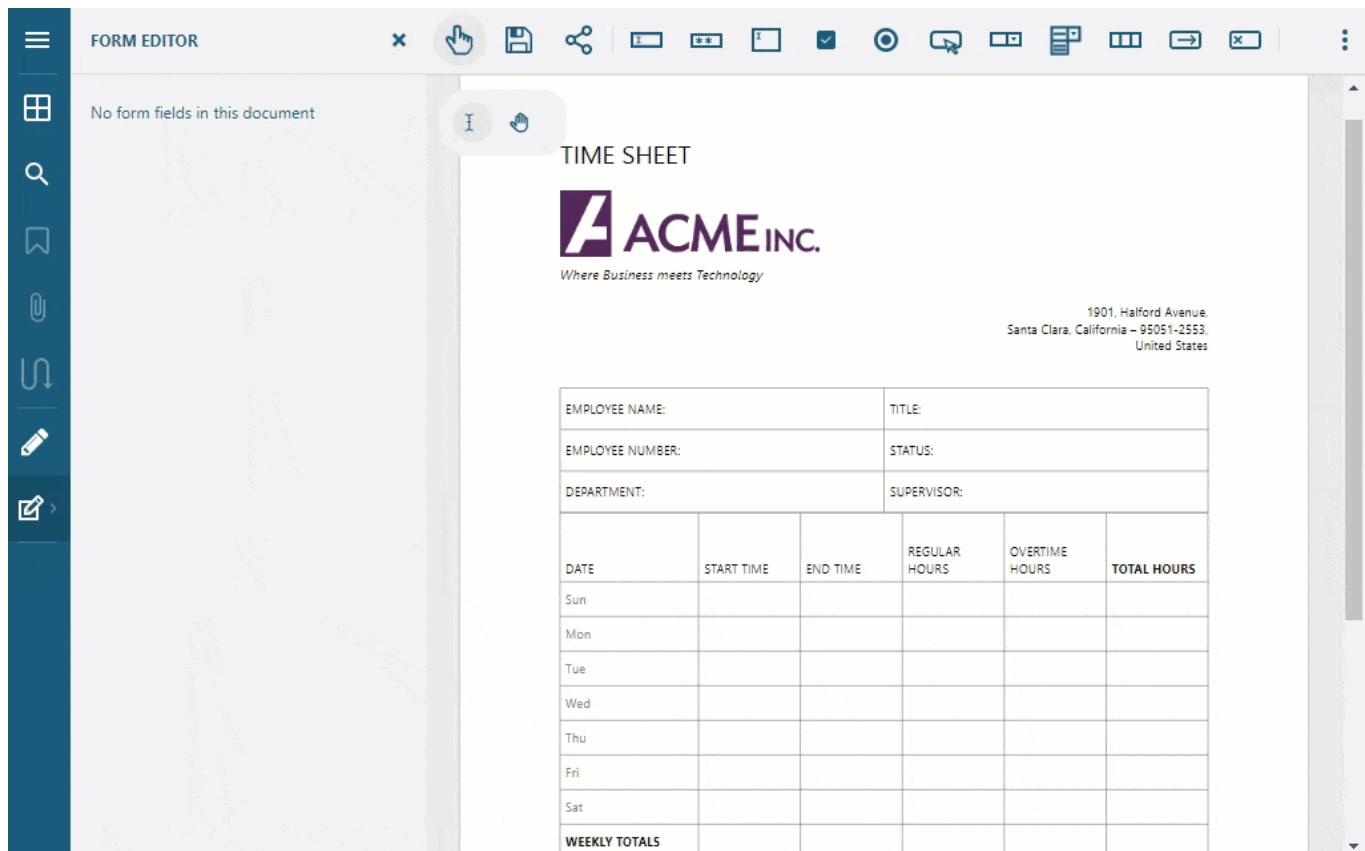


You can also configure which tools should be displayed in the quick editing toolbar of Form tools by using the

secondToolbarLayout property. Also, you can use Page tools to add a new blank document, insert a blank page, delete current page and perform undo or redo operations. To know more, refer [Annotation Editor](#).

Note: The editing tools (explained above) are automatically enabled in the main toolbar when [SupportApi](#) is configured in the project (which allows editing operations in a PDF document).

Alternatively, you can access all the form fields through the Form editor's toolbar which opens on clicking the Form editor button in the side panel. The below GIF shows a PDF form in GcDocs PDF Viewer in which a text field is added using the Form Editor:



The different toolbar buttons in the Form Editor are described as below:

Name	Toolbar Icons	Description
Select		Selects a form field added on PDF
Text		Adds a text field on PDF
Password		Adds a password field on PDF
Text Area		Adds a text area field to add long text on PDF
Checkbox		Adds a check box on PDF
RadioButton		Adds a radio button on PDF
PushButton		Adds a push button on PDF
Combobox		Adds a combo box on PDF
Listbox		Adds a list box on PDF
Comb-Text Field		Adds a comb-text field to add text in equally spaced positions on PDF
Submit Form Button		Adds a submit form button on PDF
Reset Form Button		Adds a reset form button on PDF
Delete Form Button		Deletes the form field
Checkbox and Radio button Appearance		

Apart from the different types of form fields described above, GcDocs PDF Viewer also provides some general editing features while working with PDF documents. They are explained as below:

Toolbar Icons	Description
	Undo changes
	Redo changes

	Saves the modified document on client
	Creates a new blank document
	Inserts a blank page
	Deletes current page

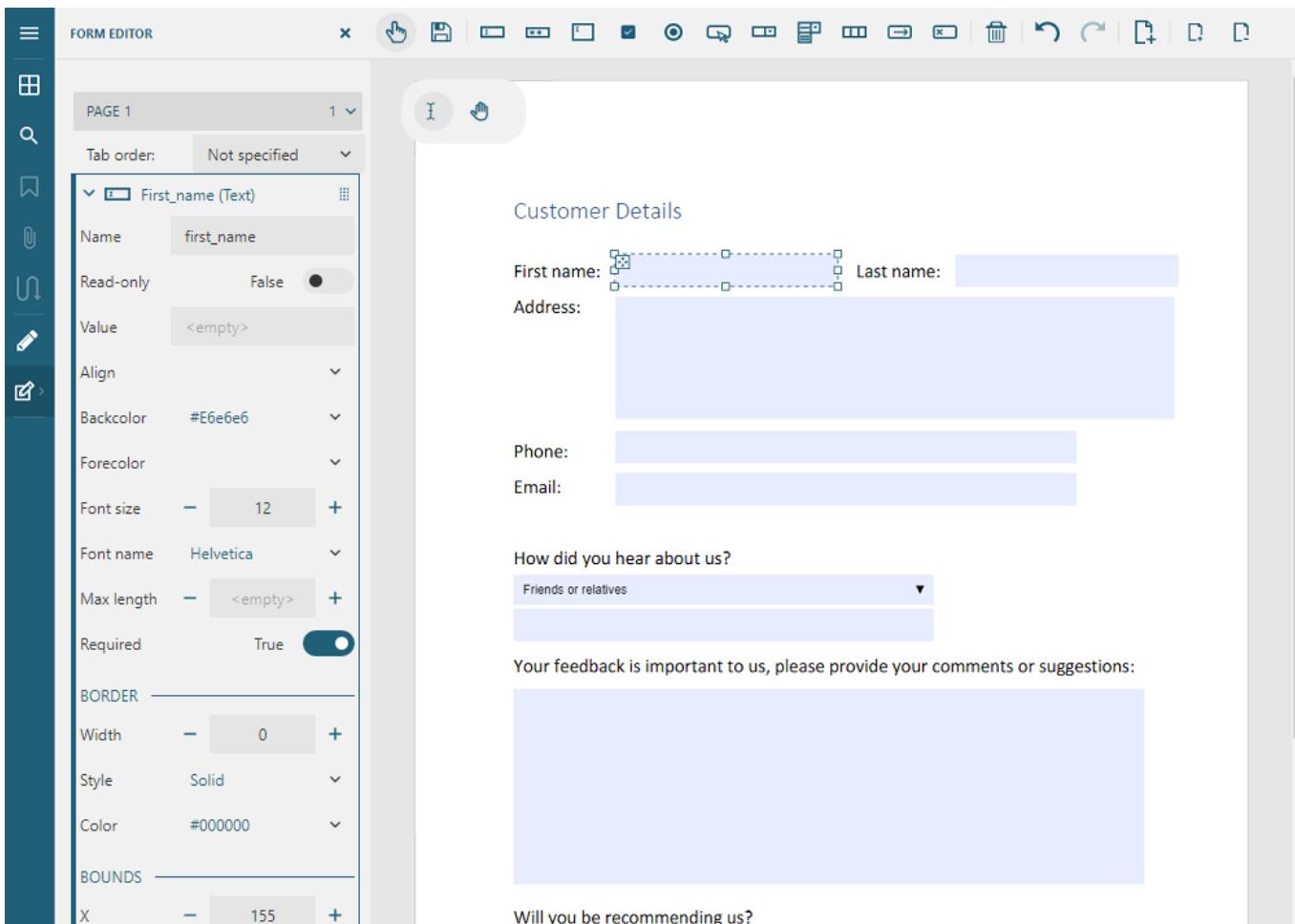
Note: You can view, print or download the original PDF document at any point of time by using the 'Hide Annotations' button on the main toolbar.

You can also insert a blank page in a PDF document and set its size by using the `newPage` method. Alternatively, you can only set the size of an existing page using `setPageSize` method. To know more, refer [Annotation Editor](#).

Property Panel of Form Editor

When you click the Form Editor icon in the left vertical panel, the Property panel of the Form Editor becomes visible. The Property panel displays the list of all the form fields page-wise in your document.

It also allows you to set or modify properties of any form field in the document like its text, border, location etc. For eg. The image below shows the properties of a text field in the Property panel.



Enable Form Editor in GcDocs PDF Viewer

The Form Editor is displayed by default in GcDocs PDF Viewer, by enabling the FormEditorPanel in the viewer using code:

Index.cshtml

```
<script>
    var viewer = new GcPdfViewer("#host", { supportApi: 'SupportApi/GcPdfViewer' });
    viewer.addDefaultPanels();
    viewer.addFormEditorPanel();
    viewer.beforeUnloadConfirmation = true;
    viewer.open("Home/GetPdf");
</script>
```

To customize the form fields in GcDocs PDF Viewer, add the following lines of code in the class file where you load the PDF in the Viewer:

C#

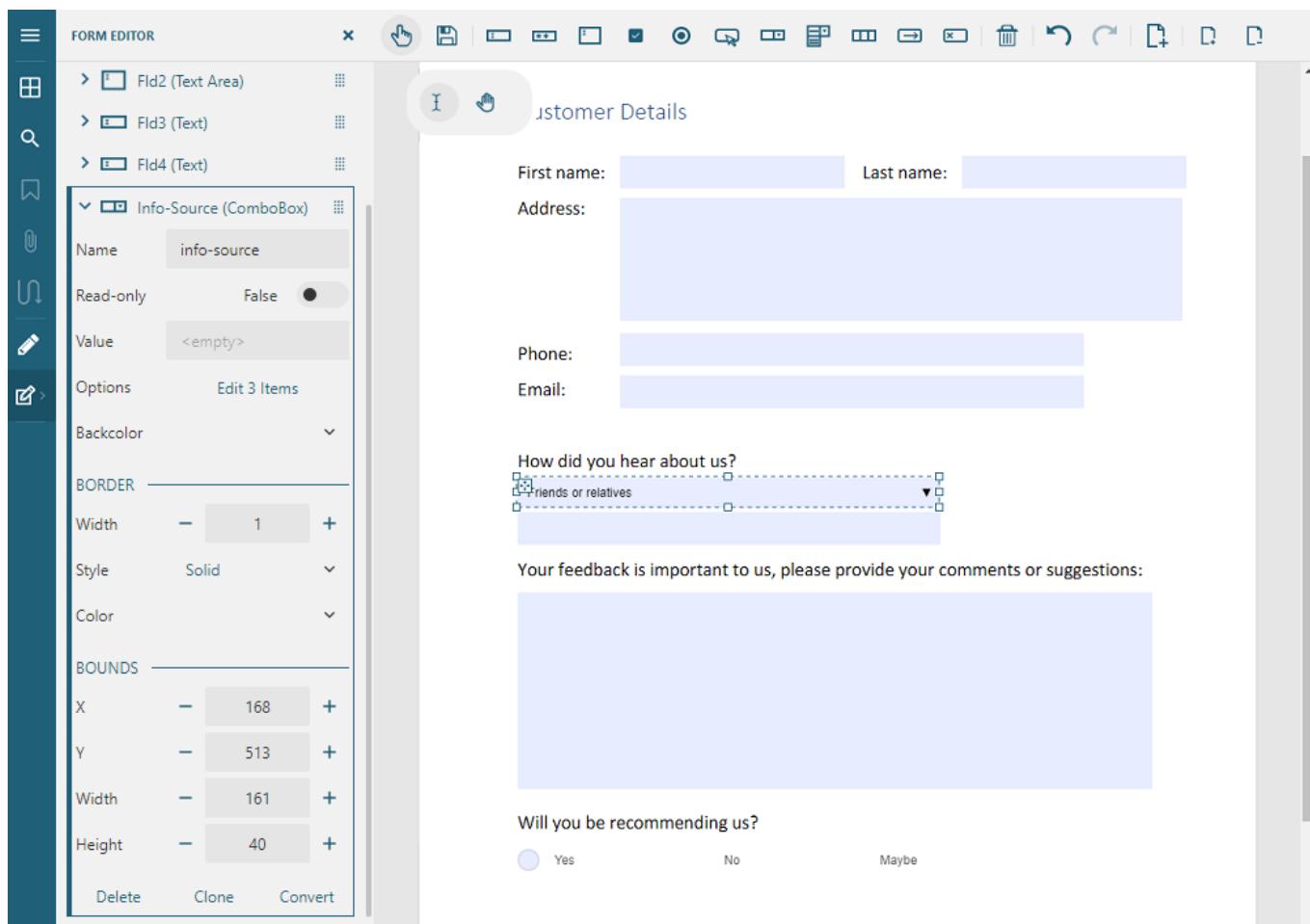
```
public static GcPdfViewerSupportApiDemo.Models.PdfViewerOptions PdfViewerOptions
{
    get => new GcPdfViewerSupportApiDemo.Models.PdfViewerOptions(
        GcPdfViewerSupportApiDemo.Models.PdfViewerOptions.Options.FormEditorPanel |
        GcPdfViewerSupportApiDemo.Models.PdfViewerOptions.Options.ActivateFormEditor,
        formEditorTools: new string[] { "edit-select-field", "$split", "edit-widget-tx-field", "edit-widget-tx-text-area", "$split", "edit-erase-field", "$split", "edit-undo", "edit-redo", "save" });
}
```

Create a PDF form using Form Editor

Follow the below steps to create a 'Customer Detail' PDF form using the Form Editor in GcDocs PDF Viewer:

1. Configure the [GcDocs PDF Viewer for editing](#) PDF documents and run the application.
2. Load a PDF document containing static text corresponding to which you want to add form fields.
3. Open the Form Editor using the last icon on the left vertical panel.
4. Add text fields, text area and radio buttons to your form by using form editor tools.
5. Set properties of form fields from the Property panel like location, border etc.
6. Add a combo box which displays various drop down options when clicked (as shown below).
7. Close the Form Editor and go back to View mode.

A Customer Detail PDF form is created successfully. You can view the list of all the form fields in the property panel of Form Editor.



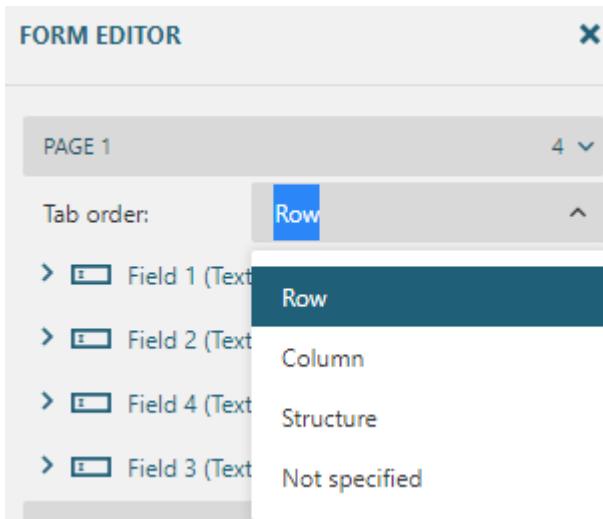
Tab Order of Form Fields

The tab order of form fields can be viewed (if already set in a PDF document) or set in the property panel of Form Editor. It helps to navigate through the form fields using the tab key. This is specially useful when filling long forms. GcDocs PDF Viewer allows you to set any of the tab order settings specified in the [PDF specification 1.7](#), namely:

- Row
- Column
- Structure
- Not Specified

In [view mode](#), only the 'Row', 'Column' and 'Not Specified' (follows the order of annotations in the page) tab orders are supported.

 **Note:** 'Structure' is not supported in view mode, but can be set in the editor (to be used by other viewers, for example Adobe Acrobat Reader).



You can also set the tab order in a PDF document using GcPdf API. To know more, refer [Forms](#).

💡 Other Resources

Using a PDF Form Designer for Web	A blog post on how to design new PDF forms or edit the existing ones and use various form editor features.
How to Create a PDF Form Using GcPdfViewer	A detailed blog on how to use the GcDocs PDF Viewer to develop a Health Intake form for an online yoga class.
Form Editor	The online sample browser demonstrating the AcroForm editing features of GcDocs PDF Viewer.
Sample Forms	The online sample browser demonstrating the filling, editing, saving or printing various types of PDF forms like Tax forms, E-commerce, HR, Membership, Events etc.

Orientation of Form Field

When a form field is placed on a rotated PDF page, it is added in its original position and does not adjust to the new orientation by itself. To enable rotation of form fields placed on a PDF page, GcDocs PDF Viewer provides **Orientation** property which lets you set the field orientation in multiples of 90 degrees.

Below sample code shows how to set orientation of the form fields in a PDF document.

```
index.cshtml
// Change orientation to 180 degrees
fld1.orientation = 180;
```

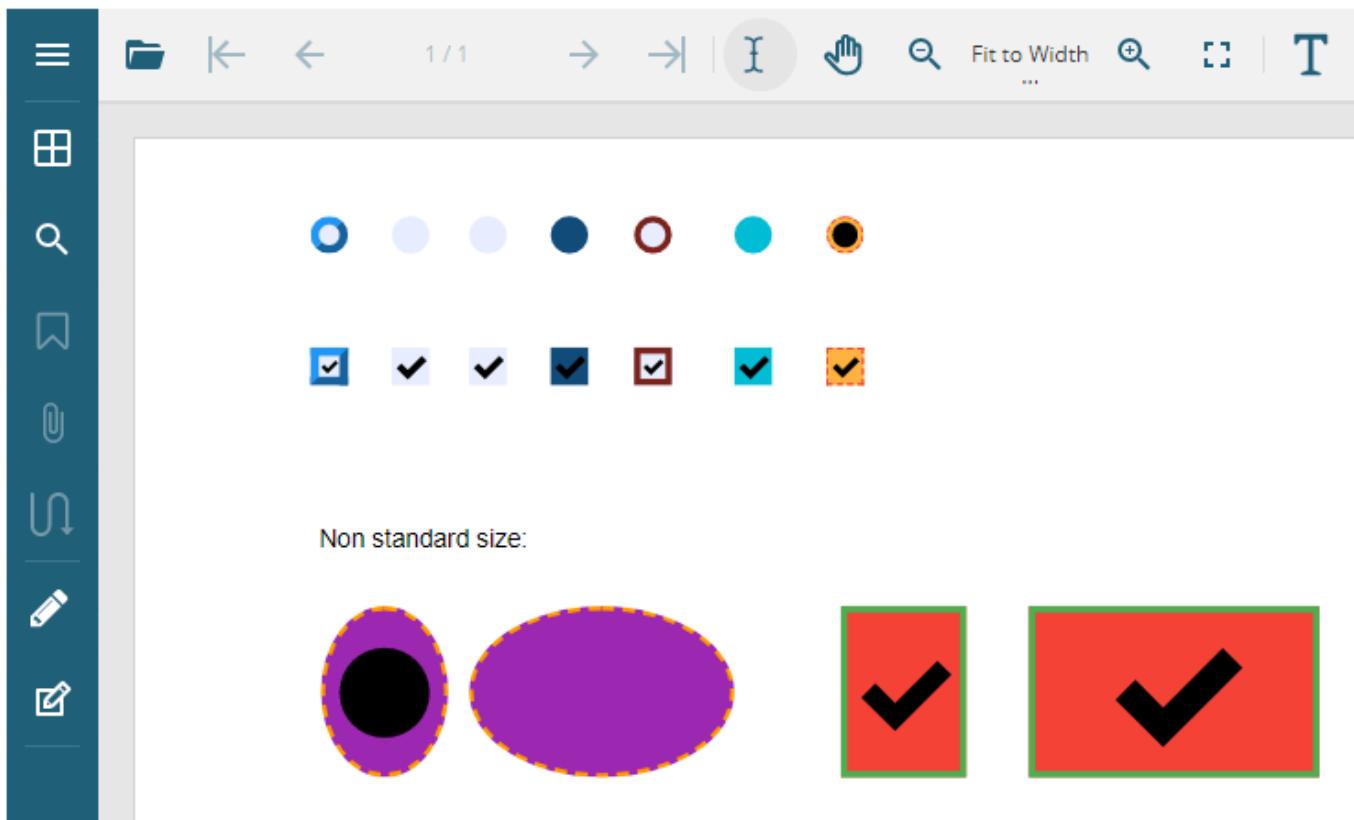
Checkbox and Radio button Appearance

GcDocs PDF Viewer allows you to render the customized appearance of radio and checkbox buttons. It provides three appearance rendering types which can be configured by using the **fieldsAppearance** option.

Custom

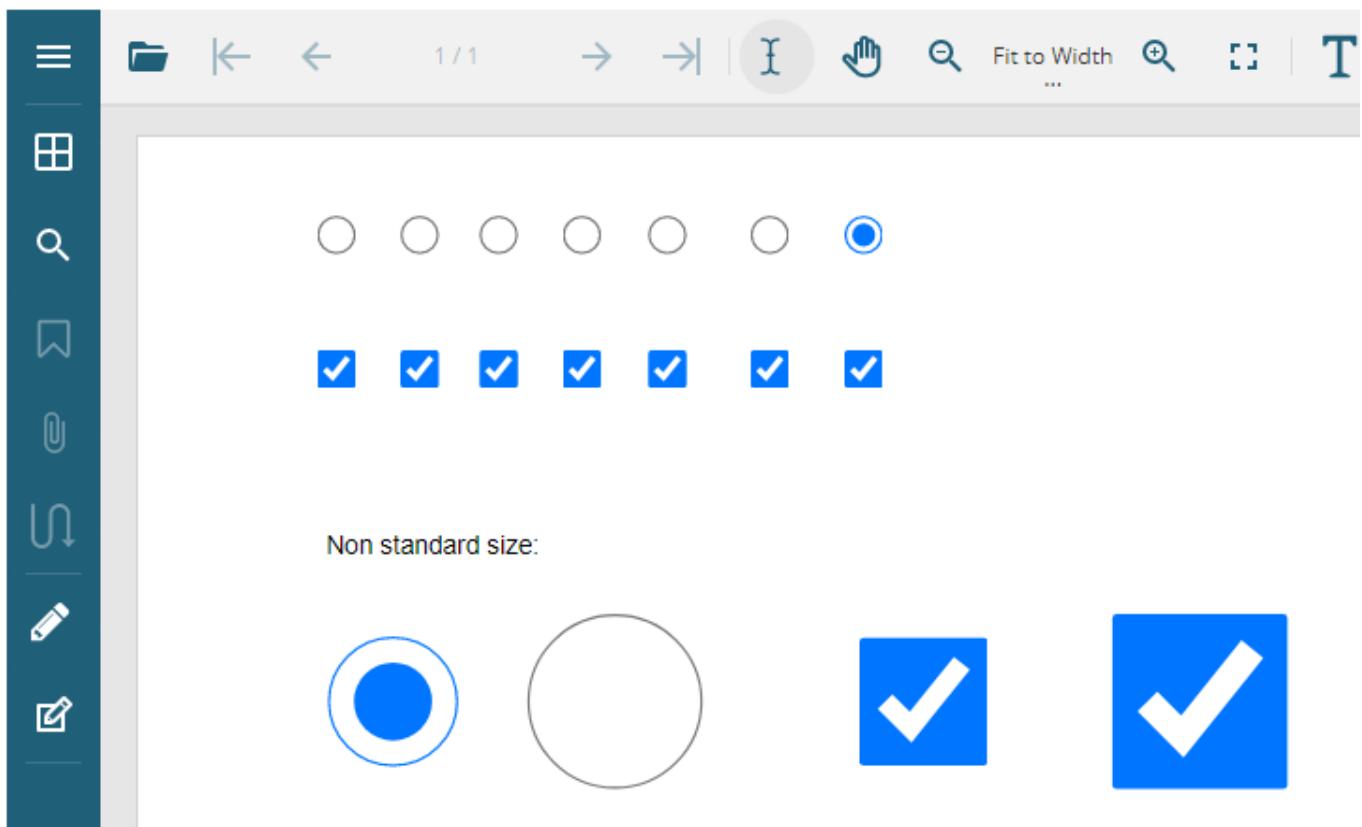
This is the default appearance type and is close to the styles defined in the PDF document. As can be observed in the

below image, it supports background color and border styles as well:



Web

This is the standard form field appearance using platform-native styling. The styles depend on the OS or browser being used. Refer [this](#) for more details. The below image shows the 'Web' appearance of radio and checkbox buttons in a PDF document:

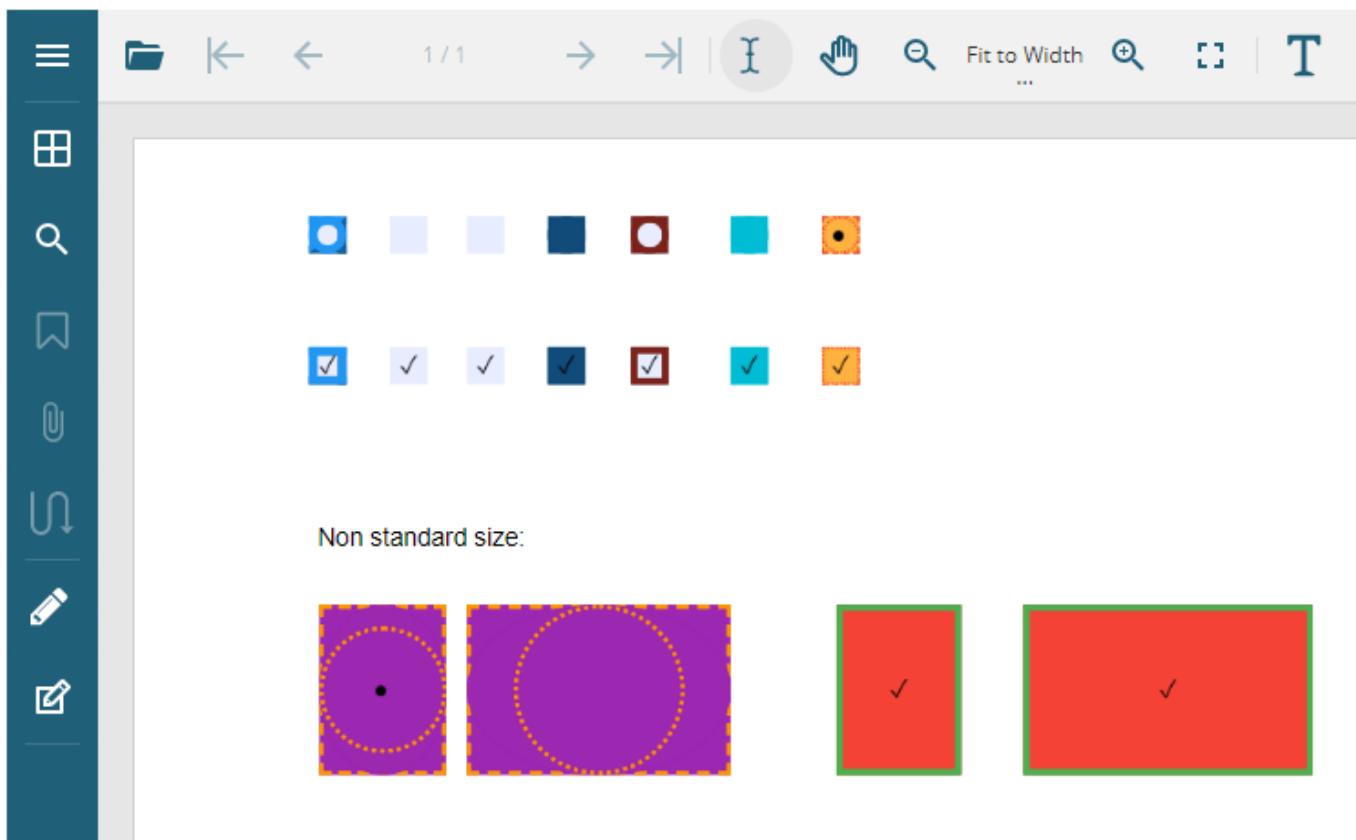


Index.cshtml

```
var viewer = new GcPdfViewer("#root", { fieldsAppearance: { radioButton: "Web",  
checkboxButton: "Web" } });
```

Predefined

This appearance type renders the radio and checkbox buttons exactly as defined in the PDF. The Predefined appearance streams from the PDF document when available, otherwise the custom appearance is used. The below image shows the 'Predefined' appearance of radio and checkbox buttons in a PDF document:

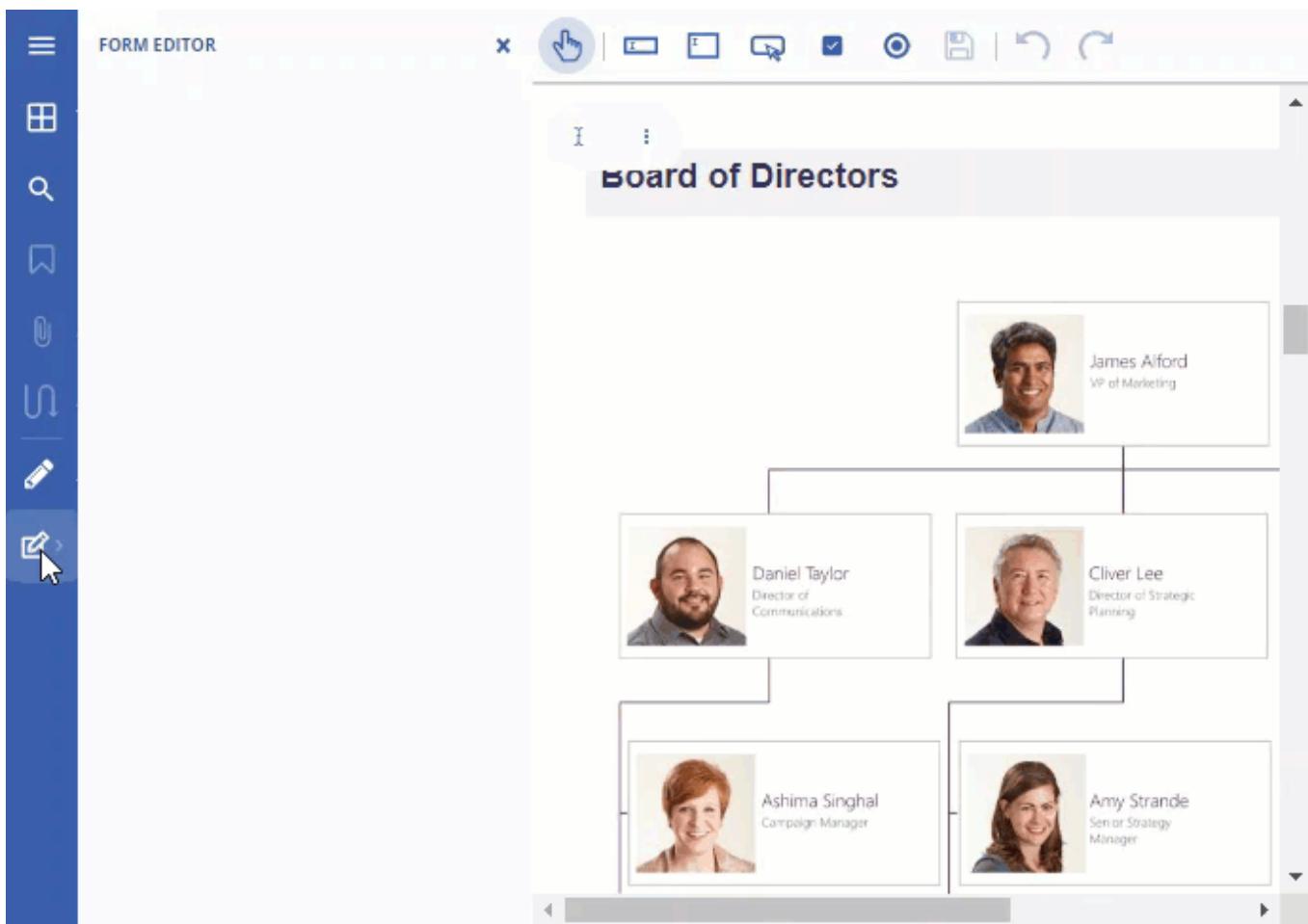


Index.cshtml

```
var viewer = new GcPdfViewer("#root", { fieldsAppearance: { radioButton: "Predefined", checkBoxButton: "Predefined" } });
```

Sticky Buttons

GcDocs PDF Viewer lets you add the **stickyBehavior** setting to the **toolbarLayout** property to set sticky behavior on button keys of the annotation or form editor, so that you can select the annotation or form field from the toolbar and draw it on PDF multiple times, without going back to the toolbar and selecting again.



The following table provides a list of supported annotation and form fields:

Supported Annotation Editor Keys			
edit-sign-tool	edit-text	edit-free-text	edit-ink
edit-square	edit-circle	edit-line	edit-polyline
edit-polygon	edit-stamp	edit-file-attachment	edit-sound
edit-link	edit-redact		
Supported Form Editor Keys			
edit-widget-tx-field	edit-widget-tx-password	edit-widget-tx-text-area	edit-widget-btn-checkbox
edit-widget-btn-radio	edit-widget-btn-push	edit-widget-ch-combo	edit-widget-ch-list-box
edit-widget-tx-comb	edit-widget-btn-submit	edit-widget-btn-reset	

The code snippet below shows how to set sticky buttons in the GcDocs PDF Viewer.

index.cshtml

```
// Set the sticky behavior for drawing annotations and form widgets:
viewer.toolbarLayout.stickyBehavior = ['edit-ink', 'edit-square', 'edit-circle',
'edit-line', 'edit-polyline', 'edit-polygon', "edit-redact",
"edit-widget-tx-field", "edit-widget-tx-text-area", "edit-widget-btn-push", "edit-
widget-btn-checkbox", "edit-widget-btn-radio"];
```

```
viewer.applyToolbarLayout();
```

Features

The Annotation and Form editor in GcDocs PDF Viewer provides various other useful features while editing PDF documents:

- Align
- Copy or Paste
- Convert to Content
- Default Settings

Align

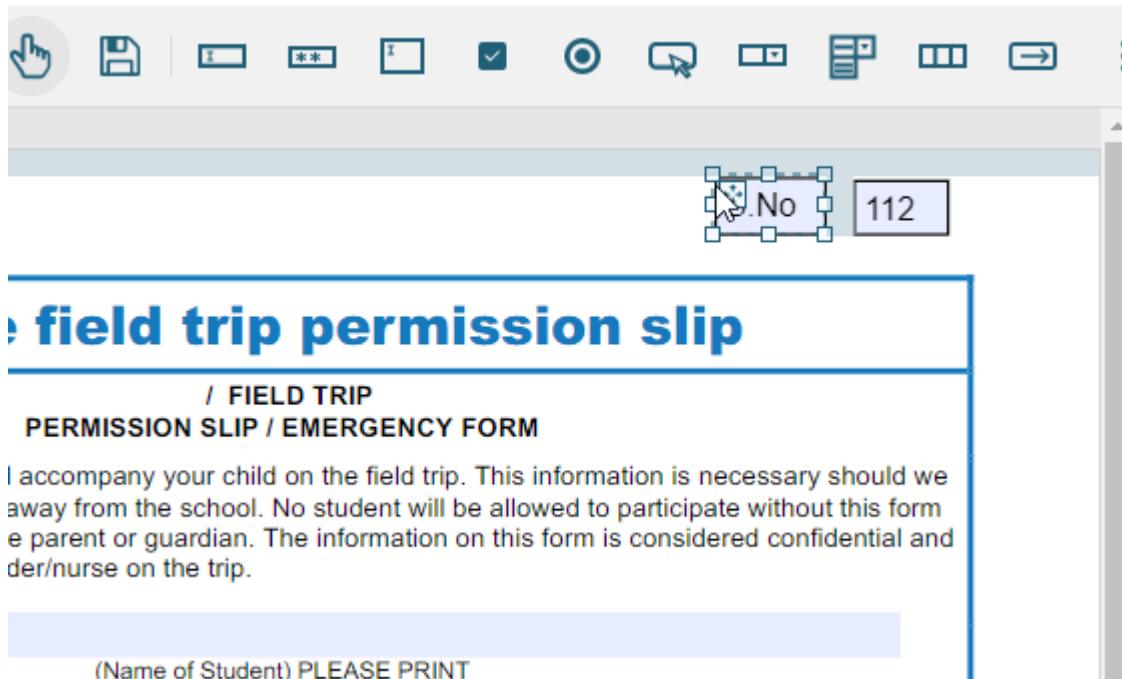
GcDocs PDF Viewer allows you to align annotations and form fields with each other. The snap alignment feature, which is enabled by default, can be used to align fields to the left, top, right, center or bottom edge.

For example, consider a PDF document containing an annotation or a form field and a new one is also added. While moving or resizing the new field, a dashed line will appear when it approaches the existing field's edge or center point. It indicates the alignment of new field with respect to existing field.



Margin

Apart from aligning the fields, GcDocs PDF Viewer allows you to define margins by using snap margin feature. These margins are the extra spaces before or after the edge of a field or page. The default margin between two elements or page edges is 10 points.



Using Keyboard Shortcuts

You can also fine tune the location and size of fields annotation and form fields by using keyboard shortcuts:

- To change location of selected field:
 - By 1 point - *Arrow keys*
 - By 10 points - *Ctrl+Arrow*
- To increase or decrease a field's size:
 - By 1 point - *Shift+Arrow*
 - By 10 points - *Ctrl+Shift+Arrow*

 **Note:** Alt key temporarily disables the Snap alignment feature during resize or move action.

Using Code

You can disable or customize snap alignment feature by using **snapAlignment** option in API.

The full specification for **snapAlignment** option is:

Index.cshtml

```
snapAlignment: true | false |
{
    tolerance: number | { horizontal: number | false, vertical: number | false },
    margin: false | true | number | { horizontal: number | boolean, vertical:
number | boolean },
    center: false | true | { horizontal: boolean, vertical: boolean },
}
```

The description of above settings:

- **tolerance** - Distance between the edges of two objects within which the object that is being moved or resized snaps to the other object.
- **margin** - Distance from the target object or page edge to which the edge of the object being moved or resized snaps.
- **center** - Allows you to snap objects to centers of other objects (in addition to edges).

By default, snap tolerance is 5 points, snap margin is 10 points, snap to center is true.

The tolerance value of snap alignment feature can be set by using below code:

Index.cshtml

```
var viewer = new GcPdfViewer("#root", { snapAlignment: { tolerance: 25 }, supportApi:
'api/pdf-viewer' });
```

The tolerance value of vertical and horizontal alignment can be set separately by using below code:

Index.cshtml

```
var viewer = new GcPdfViewer("#root", { snapAlignment: { tolerance: { vertical: 10,
horizontal: 50 } }, supportApi: 'api/pdf-viewer' });
```

The snap alignment feature to the center of an element can be disabled by using below code:

Index.cshtml

```
var viewer = new GcPdfViewer("#root", { snapAlignment: { center: false }, supportApi:
'api/pdf-viewer' });
```

The snap alignment feature to the center of an element for vertical alignment can be enabled by using below code:

Index.cshtml

```
var viewer = new GcPdfViewer("#root", { snapAlignment: { center: { vertical: true, horizontal: false, } }, supportApi: 'api/pdf-viewer' });
```

The snap alignment feature can be disabled by using below code:

Index.cshtml

```
var viewer = new GcPdfViewer("#root", { snapAlignment: false, supportApi: 'api/pdf-viewer' });
```

The horizontal snap margin can be disabled and vertical snap margin value can be set by using below code:

Index.cshtml

```
var viewer = new GcPdfViewer("#root", { snapAlignment: { margin: { vertical: 50, horizontal: false } }, supportApi: 'api/pdf-viewer' } );
```

The horizontal alignment can be disabled by using below code:

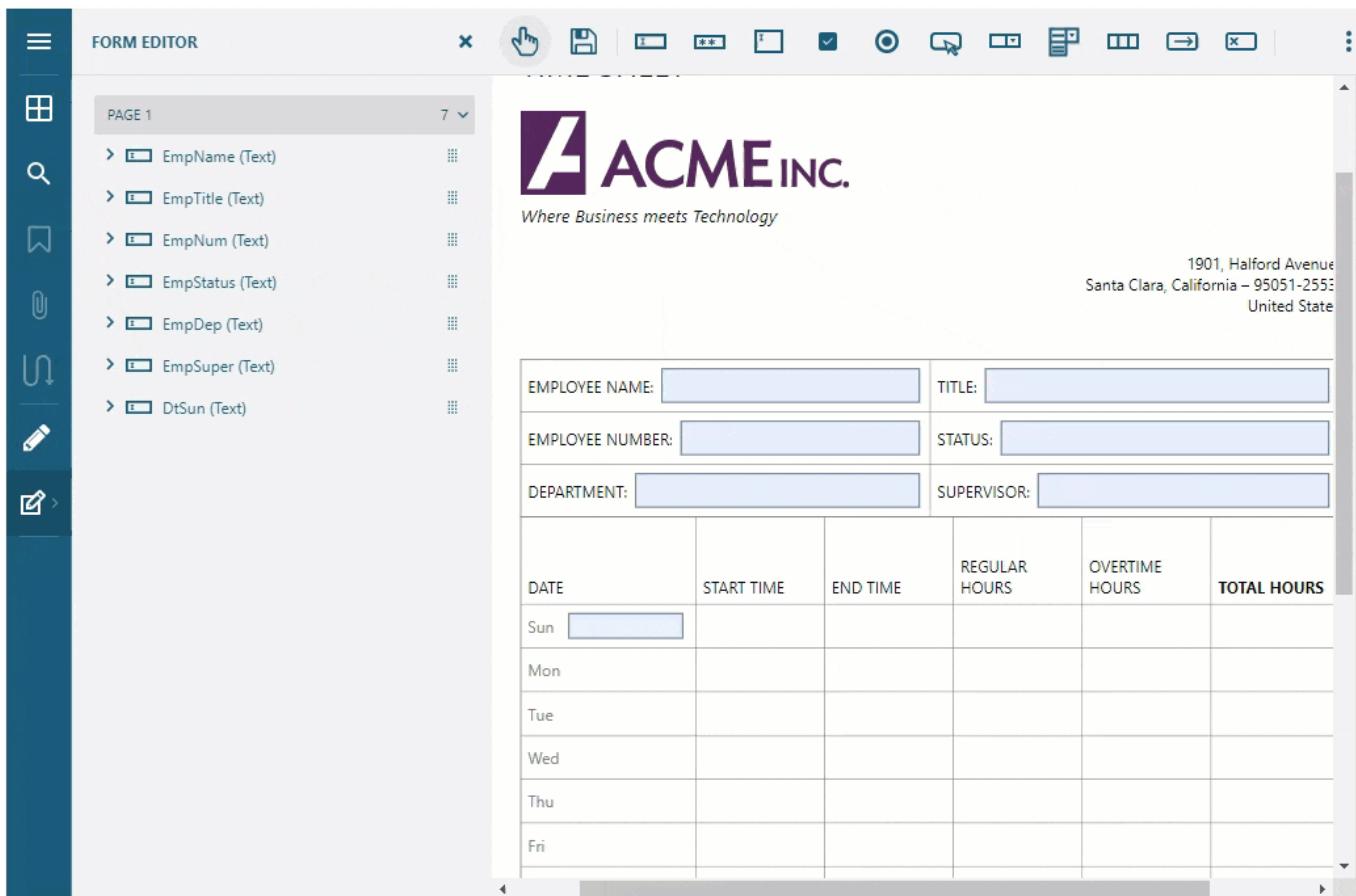
Index.cshtml

```
var viewer = new GcPdfViewer("#root", { snapAlignment: { tolerance: { horizontal: false } }, supportApi: 'api/pdf-viewer' } );
```

Copy and Paste

The Annotation and Form Editor in GcDocs PDF Viewer allows you to copy and paste existing annotations or form fields respectively. The 'Clone' button in the property panel creates an exact copy of the annotation or form field at the same location on the same page. The position and other properties of a cloned field can be changed in the property panel.

For example, while creating a PDF form, you can clone a form field and change its 'X' property to align it horizontally or 'Y' property to align it vertically with the original field.



Note: The cloned annotation or form field has a new id and field name.

Using Keyboard Shortcuts

The copy and paste operation can also be performed by using keyboard shortcuts:

- Copy: Ctrl+C
- Paste: Ctrl+V

The annotations and form fields can be pasted on the same page, different page or even in another PDF document. When keyboard shortcut (Ctrl+V) is used for the paste action, the cloned field is always placed in the center of a page. You can also use the context menu options by right clicking the field. Once cut or copied, the field can be pasted by right clicking and using 'Paste' option from context menu.

EMPLOYEE NAME:			TITLE:		
EMPLOYEE NUMBER:			STATUS:		
DEPARTMENT:			SUPERVISOR:		
DATE	START TIME	END TIME	REGULAR HOURS	OVERTIME HOURS	TOTAL HOURS
Sun	<input type="text"/>				
Mon	<input type="text"/>			Cut (Ctrl+X)	
Tue				Copy (Ctrl+C)	
Wed				Delete (DEL)	
Thu					
Fri					
Sat					

Using Code

To add a cloned annotation or form field to a PDF document using code:

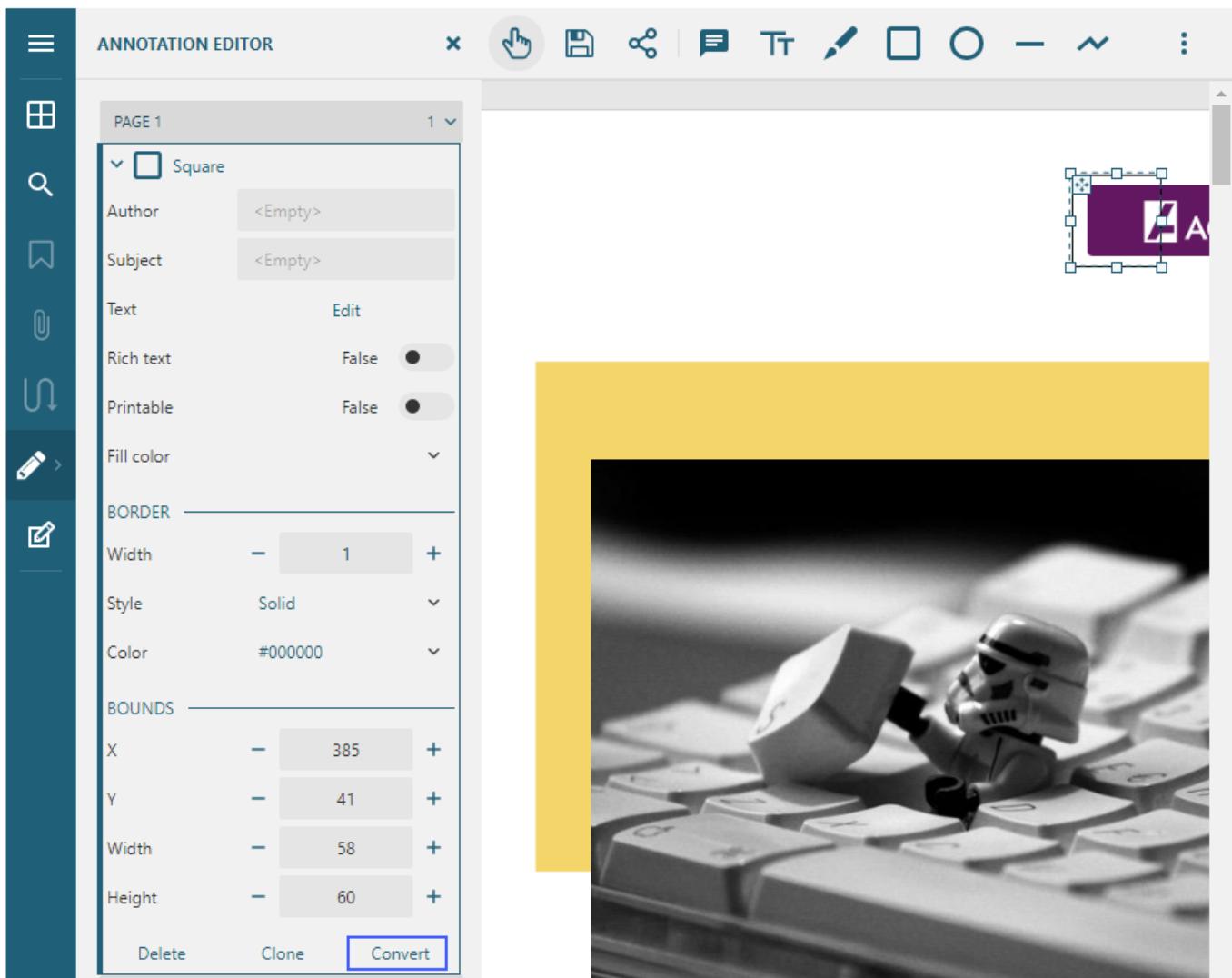
Index.cshtml

```
//Find field widget with name field1
const resultData = await viewer.findAnnotations("field1", { findField: 'fieldName' });
//Clone field
const clonedField = viewer.cloneAnnotation(resultData[0].annotation);
//Change field name property
clonedField.fieldName = "field1Copy";
//Add cloned field to the second page
viewer.addAnnotation(1, clonedField);
```

Convert to Content

The Annotation and Form Editor allows you to convert annotations and form fields to content elements. Once converted, the annotations and form fields are rendered as content in PDF document and cannot be edited. On saving the PDF document, the elements marked for conversion are removed and appear as a part of PDF content.

The below image shows 'Convert' button in Annotation Editor's properties panel along with the square annotation.



After conversion, the 'Convert' button is changed to 'Revert' button as shown in the below image. You can use the 'Revert' button to undo the conversion and make the annotation and form field visible and editable again.



Default Settings

The values for annotation and form field properties can be set by using the properties panel of respective editors. However, you can also set the default value of any annotation or form field property by using the **editorDefaults**.

For example, the below sample code sets default values of square annotation properties like border width and interior (fill) color:

```
Index.cshtml
var viewer = new GcPdfViewer("#root", {
    editorDefaults: {
        squareAnnotation: {
            borderStyle: { width: 5, style: 1 },
            color: '#000000',
            interiorColor: '#ff0000',
        }
    },
    supportApi: "api/pdf-viewer"
});
```

The default values, as set above, will be used every time a square annotation is added to a PDF document as shown below:

BALANCE SHEET

Total Assets	
Cash in Bank	\$45,000.00
Inventory	\$45,000.00
Prepaid Expenses	\$600.00
Other	\$10,000.00
Total	\$1,00,600.00

Fixed Assets

Machinery & Equipment	
Machinery & Equipment	\$56,200.00
Furniture & Fixtures	\$32,400.00
Leasehold Improvements	\$6,300.00
Real Estate / Buildings	\$62,50,000.00

Opacity

You can set the opacity property for all annotations. The valid value for the property can be within the range of 0.0 (fully transparent) to 1.0 (fully opaque). An annotation will appear as completely opaque if no opacity value is defined.

For example, the below sample code sets default values of square annotation properties along with opacity:

Index.cshtml

```
var viewer = new GcPdfViewer("#root", {
    editorDefaults: {
        squareAnnotation: {
            borderStyle: { width: 5, style: 1 },
            color: '#000000',
            opacity: 0.3,
            interiorColor: '#ff0000',
        }
    },
    supportApi: "api/pdf-viewer"
});
```

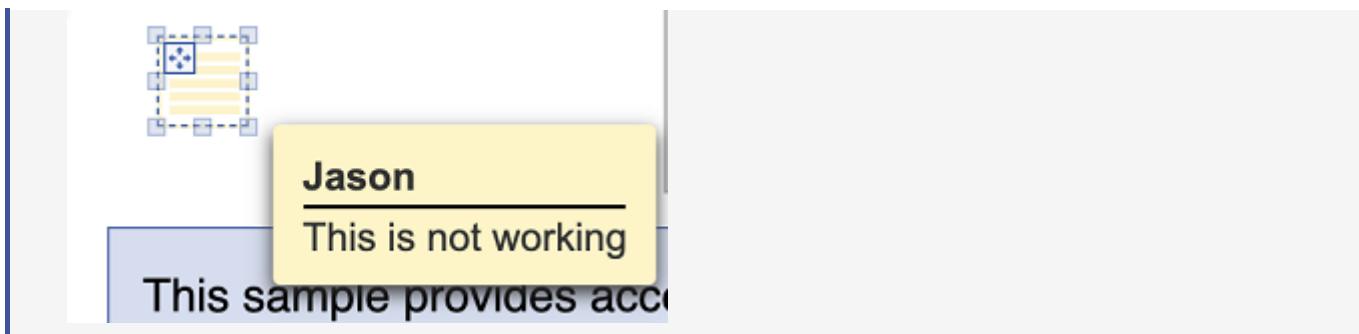
The default values of opacity and other properties will be used every time a square annotation is added to a PDF document as shown below:

The screenshot shows the Annotation Editor interface. On the left, there's a sidebar with icons for file operations like Open, Save, Print, and a search bar. The main area has a toolbar with various tools. A panel on the left displays properties for a selected 'Square' annotation on 'PAGE 4'. The properties include Author, Subject, Text, Rich text, Printable, Opacity (set to 0.3), Fill color (#Ff0000), Border width (5), Style (Solid), and Color (#000000). To the right, a 'SALES REPORT' is visible, featuring two tables. The first table shows Sales data with rows for Product A, Product B, Freight Income, and Total Sales (\$11,862.10, \$17,597.01). The second table shows Cost Of Sales data with rows for Product A, Product B, COS Adjustments, Total Cost of Sales (\$4,211.12), and Gross Profit (\$7,650.98, \$11,116.89). A red square annotation is placed over the Total Sales row in the first table.

Sales	Current Period	Year To Date
Product A	\$4,978.88	\$8,237.39
Product B	\$6,641.22	\$8,892.82
Frieght Income	\$242	\$466.80
	0	
Total Sales	\$11,862.10	\$17,597.01

Cost Of Sales	Current Period	Year To Date
Product A	\$1,728.00	\$3,024.00
Product B	\$2,328.00	\$3,301.00
COS Adjustments	\$152.12	\$155.12
Total Cost of Sales	\$4,211.12	\$6,480.12
Gross Profit	\$7,650.98	\$11,116.89

Note: The opacity value applies to all visible markup annotations and not the popup annotations as shown below:



Limitation

The opacity property is only supported by annotations and not by form fields.

Default Color for Sticky Notes

You can set the default color of a sticky note by using the **editorDefaults** option:

```
Index.cshtml
var viewer = new GcPdfViewer("#root", {
    editorDefaults: {
        stickyNote: { color: "#ff0000" }
    },
    supportApi: 'api/pdf-viewer'
});
```

The sticky note will be displayed in red color, as set above, in the PDF document as shown below:

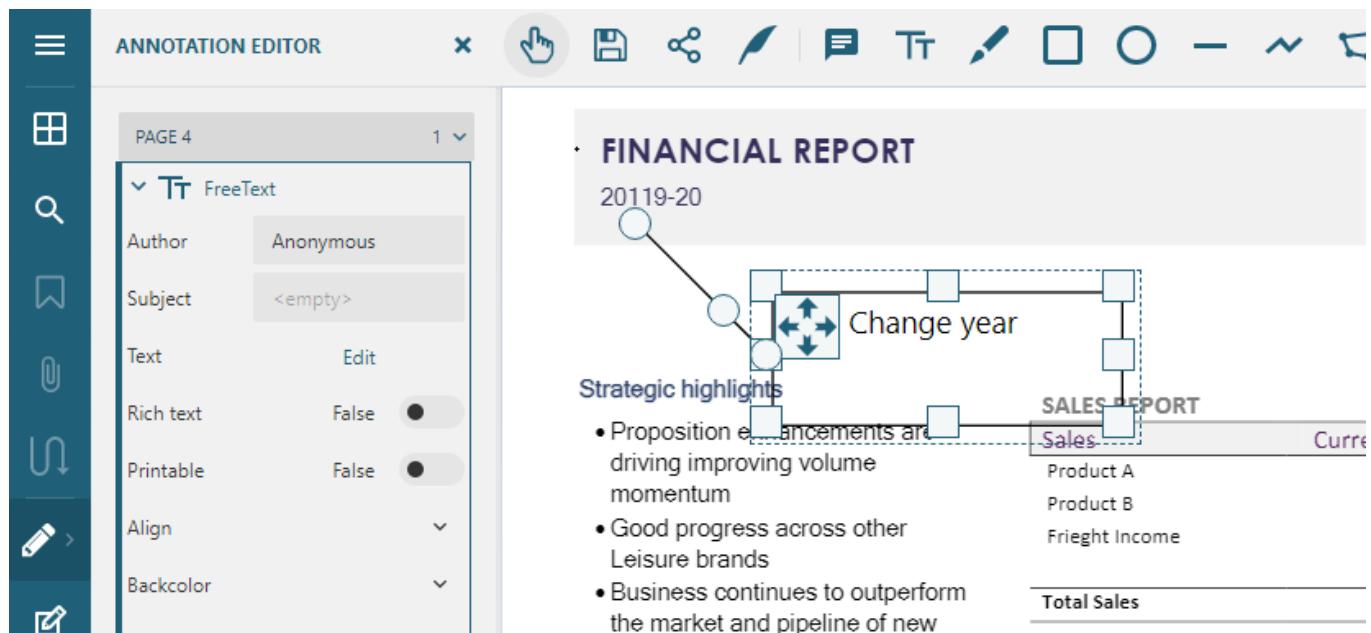
CASH FLOW REPORT	FY2019	FY2018	Variance
Operating Activities, Cash Flows Provided By or Used In			
Depreciation	\$6,111	\$7,111	-\$1,000
Adjustments To Net Income	-\$31,400	\$24,41,694	-\$55,894
Changes In Accounts Receivables	\$6,90,401	\$14,28,910	-\$7,38,509
Changes In Liabilities	\$60,07,011	-\$2,10,421	\$9,17,432
Changes In Inventories			
Changes In Other Operating Activities			
	Anonymous	Add details for FY2020	

Handle Sizes

You can set the default values for resize, move or dot handle sizes by using **editorDefaults** options. The predefined value for resizeHandleSize and moveHandleSize setting is 8 and 14 pixels respectively. The below example code sets the default values for different handles:

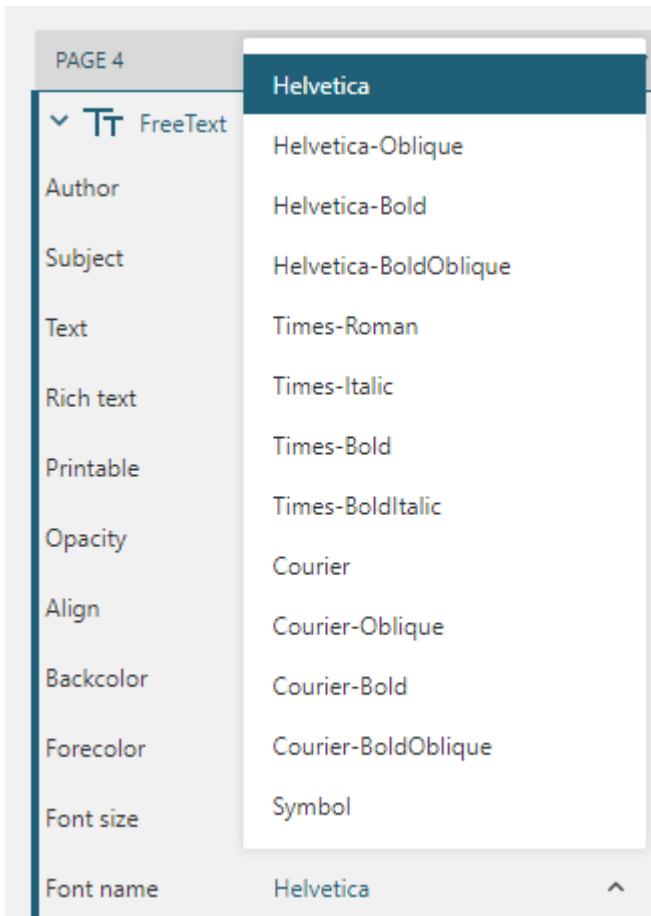
```
Index.cshtml
<script>
    var viewer = new GcPdfViewer("#host",
        {   editorDefaults: {
            resizeHandleSize: 20,
            moveHandleSize: 40,
```

```
        dotHandleSize: 20
    },
    supportApi: 'api/pdf-viewer'
}
);
viewer.addDefaultPanels();
viewer.addAnnotationEditorPanel();
viewer.addFormEditorPanel();
</script>
```



Font Family

The following fonts appear for all the text fields and FreeText annotation in GcDocs PDF Viewer, by default.



However, you can specify the default font names for the text fields and FreeText annotation by setting the **fontName** property in **editorDefaults** option.

The below example code changes the array of default font names and sets 'Courier' as default font for FreeText annotation:

Index.cshtml

```
var viewer = new GcPdfViewer("#host", {
    editorDefaults: {
        fontNames: [{ value: 'Arial', name: 'Arial' }, { value: 'Verdana', name: 'Verdana' }],
        freeTextAnnotation: { fontName: 'Courier' }
    },
    supportApi: "api/pdf-viewer"
});
```

The output of above code will look like below for a FreeText annotation:



If 'freeTextAnnotation' setting is skipped in the above code, the 'Helvetica' font will appear as the default font instead

of 'Courier' font.

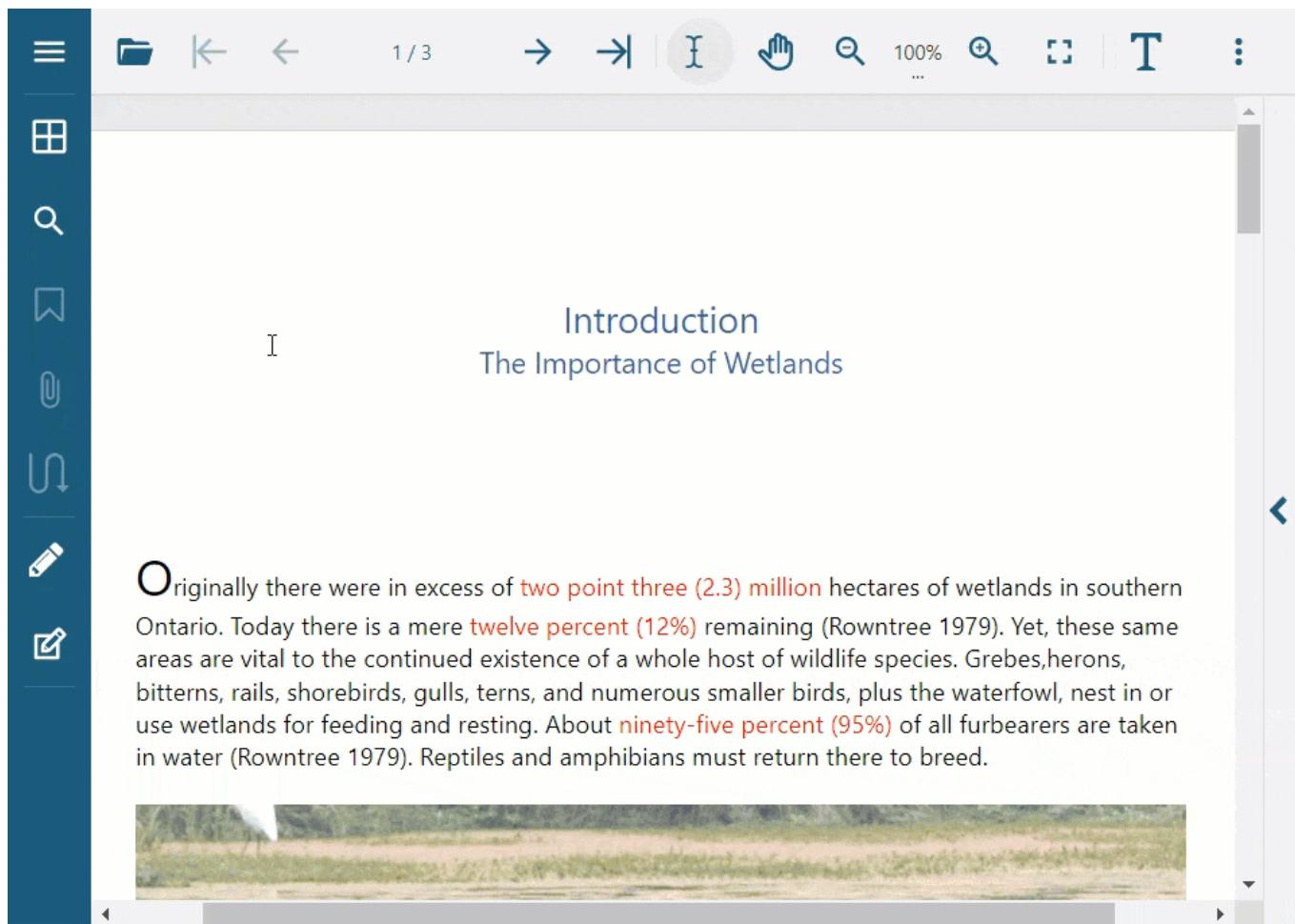
 **Note:** When a custom font is specified by using the fontNames setting, that font should be available in the web browser too.

For more information, refer [Editor Defaults](#) in GcDocs PDF Viewer demos.

Comments Tool

GcDocs PDF Viewer provides users with the ability to add text annotations and their replies as comments in PDF documents. These comments are very useful to hold some discussion, ask questions or add important information. The Comments tool can be used to add replies to text annotations in comments panel, add sticky notes to the document, view all comments in comments panel, edit or delete comments and assign their review status.

Once a text annotation is added to a document, it can be viewed in the comments panel to add a reply to it as shown below:



Once the comments panel is visible, users can see all the text annotations and their replies. To view the comments panel, choose either of the following:

- Right click on text annotation and select 'Show Comment Panel' from its context menu
- Click on the arrow at the extreme right of Viewer



Besides the comments panel, these replies are also visible below the original text annotation in the PDF document.

Enable Comments Tool

The Comments tool is hidden by default. However, it can be enabled by using **addReplyTool** method as shown below:

```
Index.cshtml
```

```
viewer.addReplyTool();
```

The Comments tool can also be enabled in expanded state which displays the comments panel, by default. The below code can be used for the same:

```
Index.cshtml
```

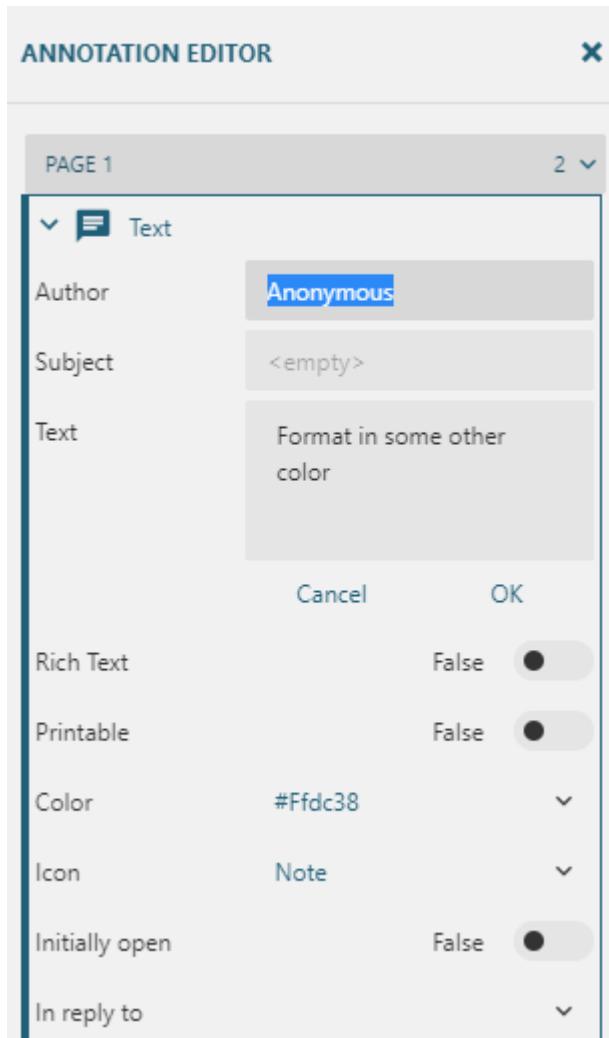
```
viewer.addReplyTool("expanded");
```

 **Note:** To use Comments tool, **SupportApi** should be configured (as it allows editing a PDF document). The tool will work in read-only mode if SupportApi is not configured. The read-only mode is particularly useful to view all comments in comments panel.

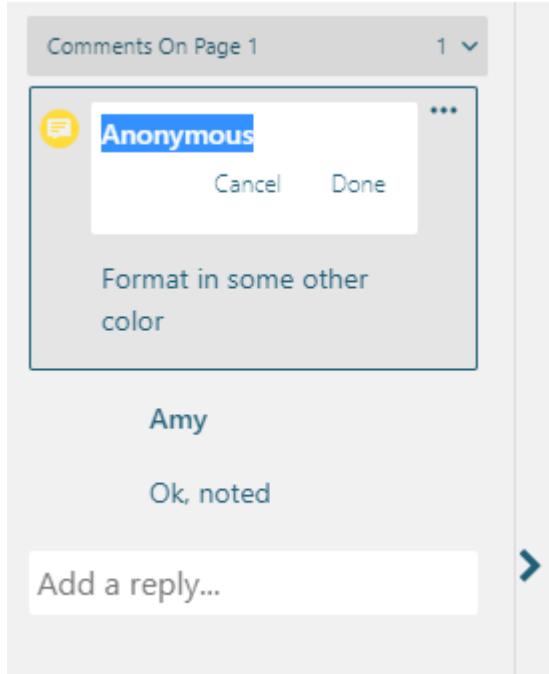
Set Author Name for Text Annotations

While replying to a text annotation, the author name is displayed as 'Anonymous' by default. However, the desired author name can be set in following ways:

- Set author name in properties panel of Annotation editor. Once set, it is stored on the client and is used for subsequent replies.



- Set author name directly in Comments tool by clicking on author name's label to enable built-in text editor.



- Set author name using **userName** option in API as shown below:

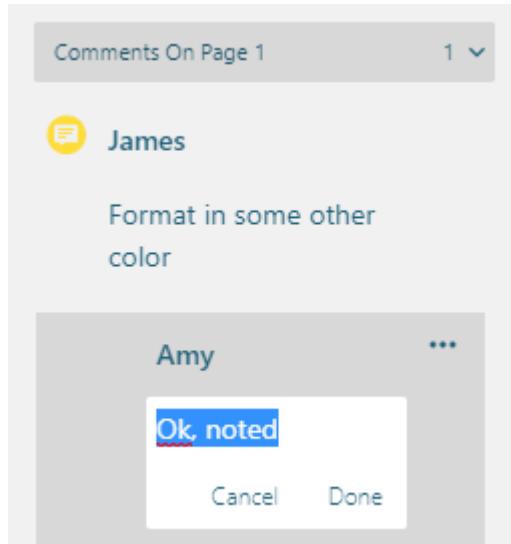
```
Index.cshtml
```

```
var viewer = new GcPdfViewer("#root", { userName: 'Jaime Smith', supportApi: 'api/pdf-viewer' });
```

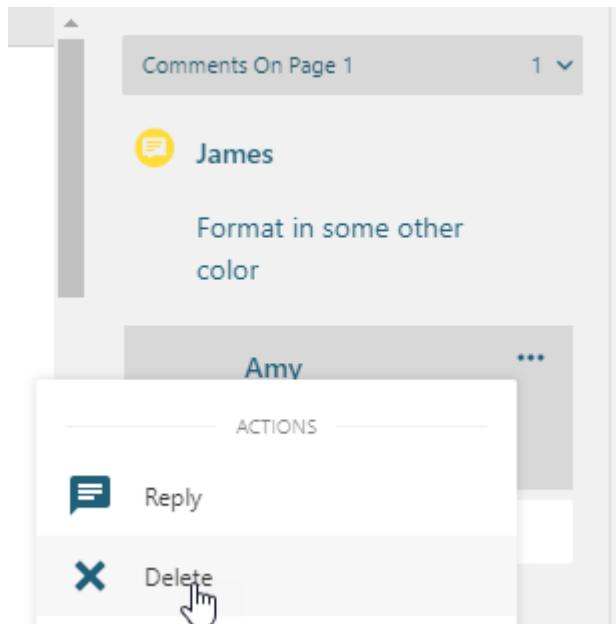
Note: If author's name is set in both API and annotation editor, the author name defined in API is given priority.

Modify or Delete Text Annotations

A text annotation or reply can be edited by clicking on it in the comments panel as shown below:



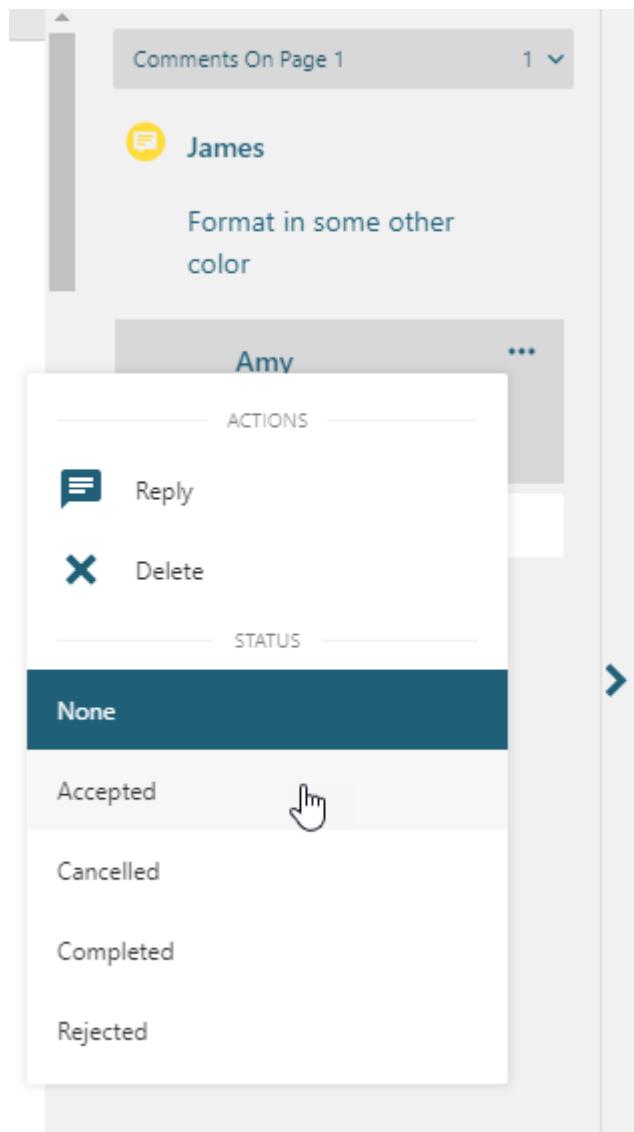
A text annotation or reply can be deleted by clicking on **Actions | Delete**



Note: To delete the parent annotation and retain its replies, use the properties panel of Annotation Editor on the left sidebar. Otherwise, all its replies are removed as well.

Add Review Status to Text Annotations

You can also add review status to a text annotation in the comments panel by clicking on **Actions | Status**:



The status is added as an icon to the comment and displays assignee's name when hovered upon. The review status is also visible in the PDF document below the original comment. A user can assign only 1 status to a text annotation but multiple users can assign status to a text annotation.

You can also assign status for a text annotation programmatically, by setting following properties: 'title' 'state', 'stateModel', 'referenceType', 'referenceAnnotationId'. The below example code shows how to add a reply to text annotation with id '6R' and assign 'Completed' status:

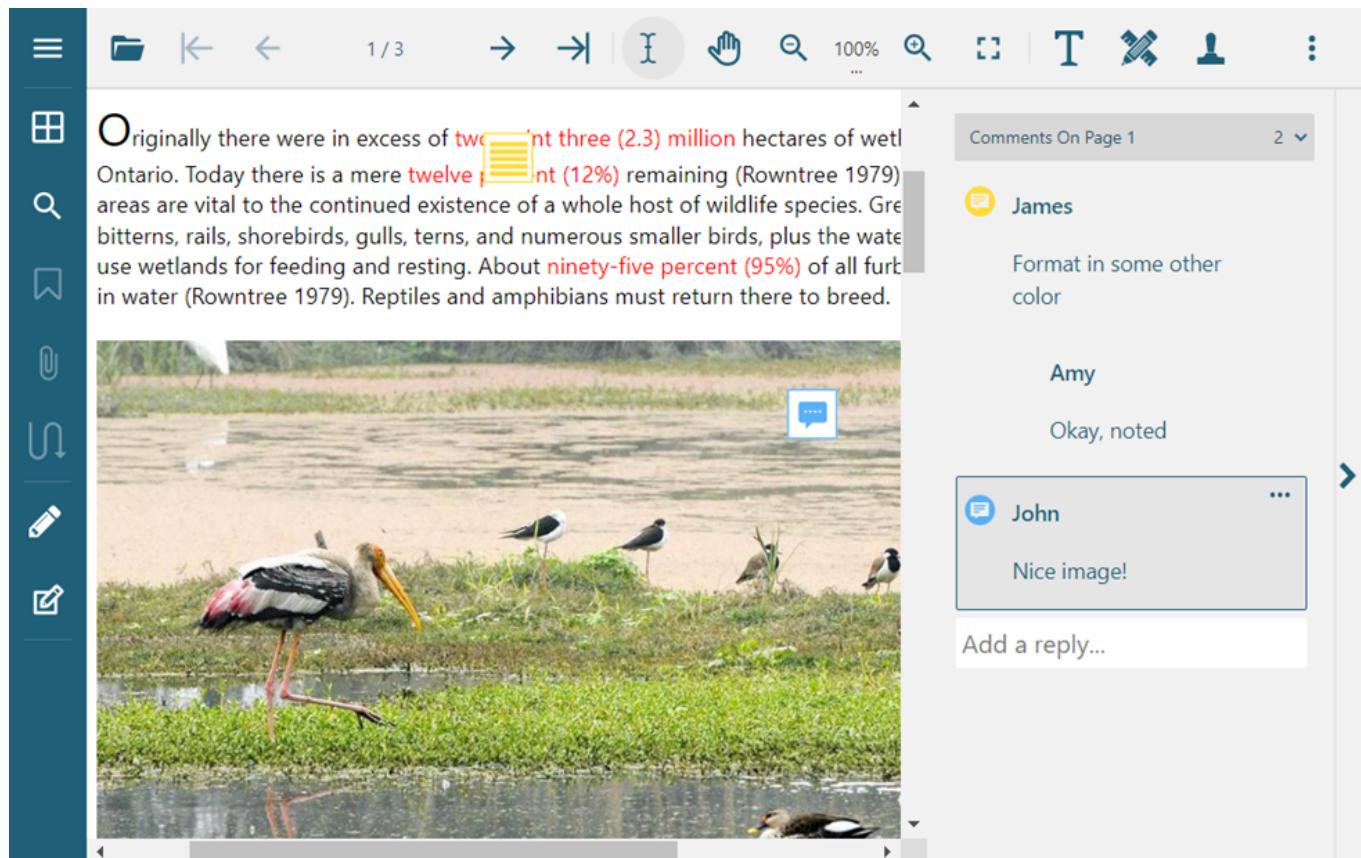
Index.cshtml

```
function addCompletedStatus() {
    viewer.findAnnotations("6R").then(function (searchResult) {
        var userName = "Jane Donahue";
        var replyAnnotation = viewer.cloneAnnotation(searchResult[0].annotation);
        replyAnnotation.title = userName;
        replyAnnotation.stateModel = 'Review';
        replyAnnotation.state = 'Completed';
        replyAnnotation.referenceType = 'R';
        replyAnnotation.referenceAnnotationId = '6R';
        replyAnnotation.contents = 'Status Completed set by ' + userName;
```

```
    viewer.addAnnotation(0, replyAnnotation);  
});  
}  
}
```

Add Sticky Note

You can also add a sticky note anywhere in the PDF document by using context menu. Once it is added, enter some text and click 'Done' in its built-in text editor to retain the note.

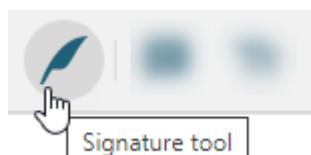


You can also add replies to it, set author name, modify or delete a note and assign its review status in the comments panel.

 **Note:** Sticky note is enabled in the context menu only when Comments tool is enabled.

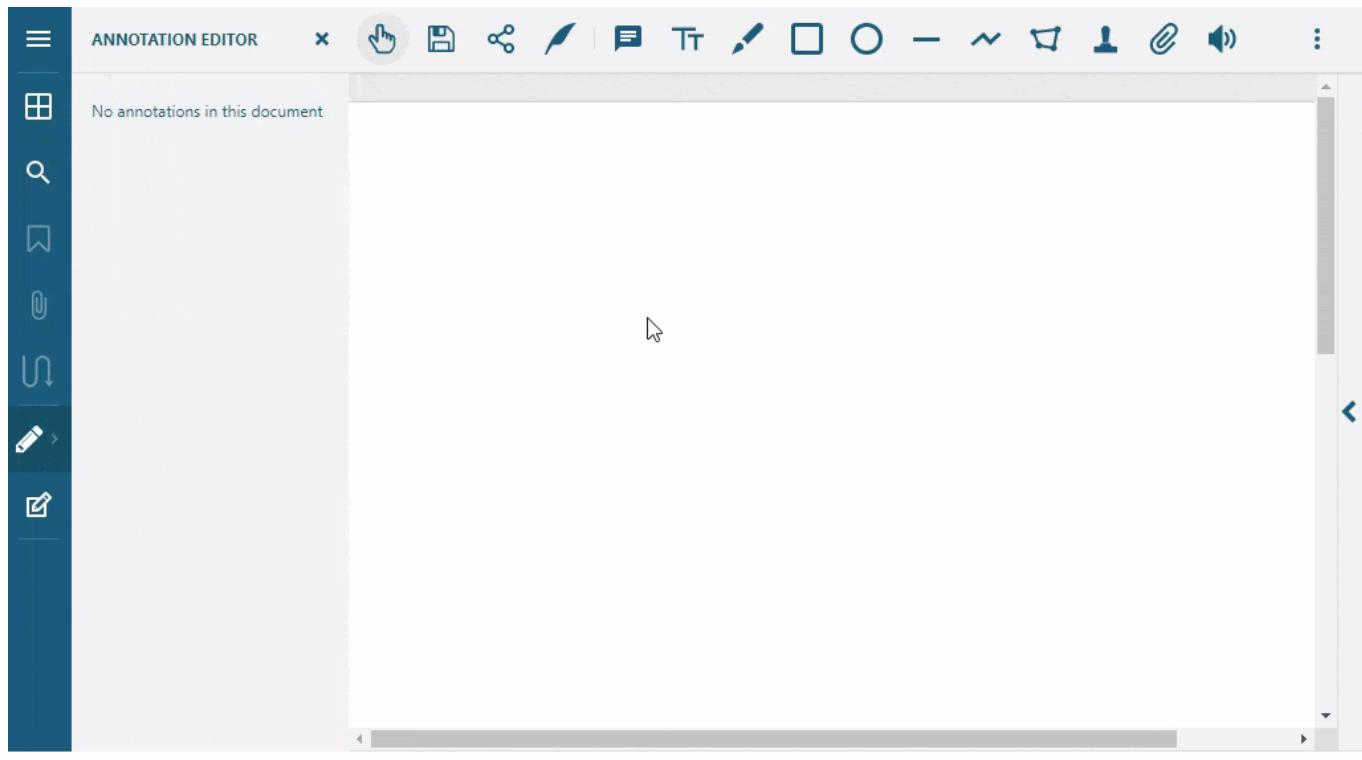
Graphical Signature Tool

GcDocs PDF Viewer allows you to add graphical signatures in PDF documents by using Signature tool in Annotation Editor.

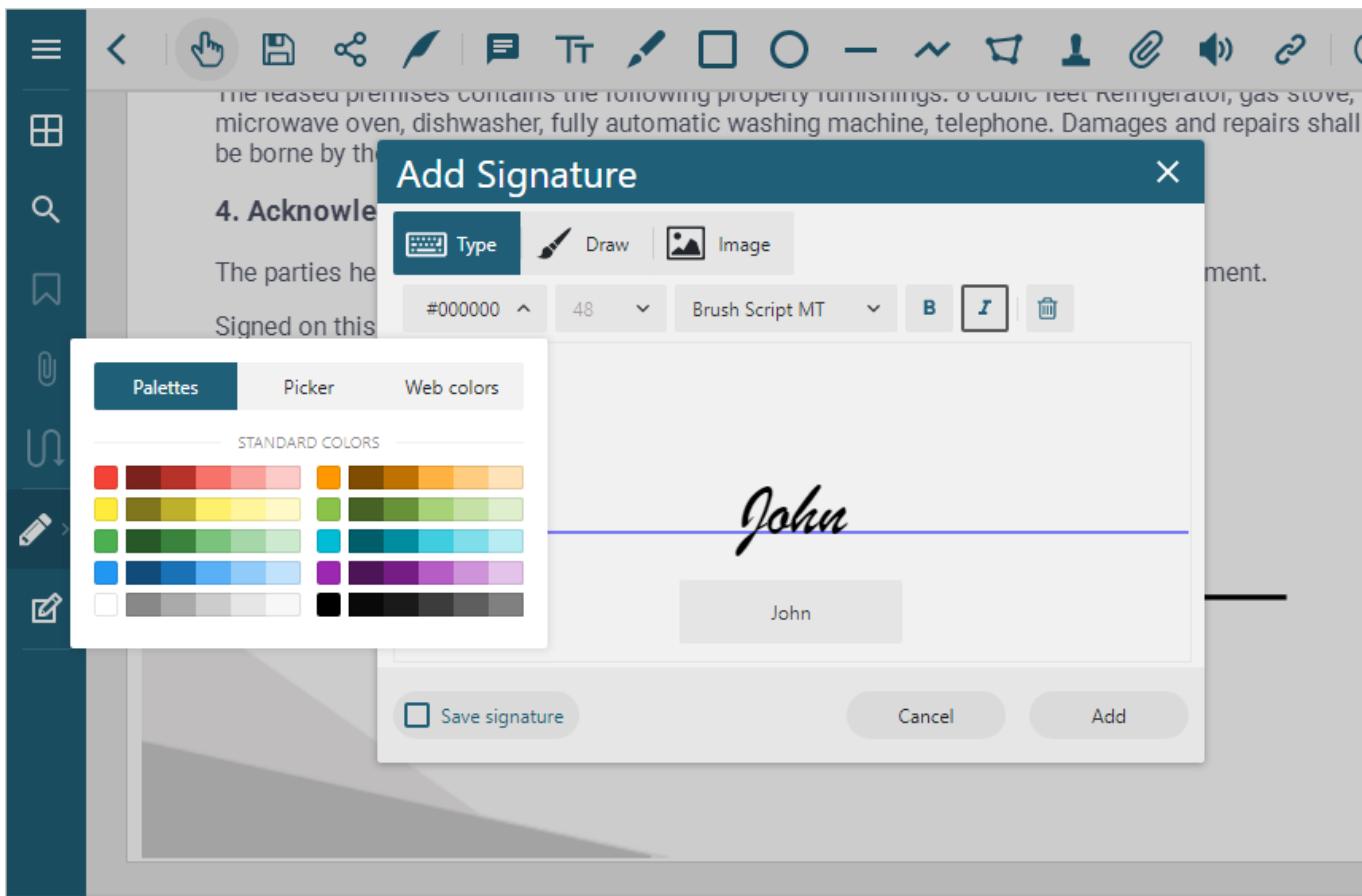


The 'Signature Tool' button opens the 'Add Signature' dialog which lets you type, draw or add the image of a signature in a PDF document. You can also format the signatures by using formatting options while typing or drawing signatures.

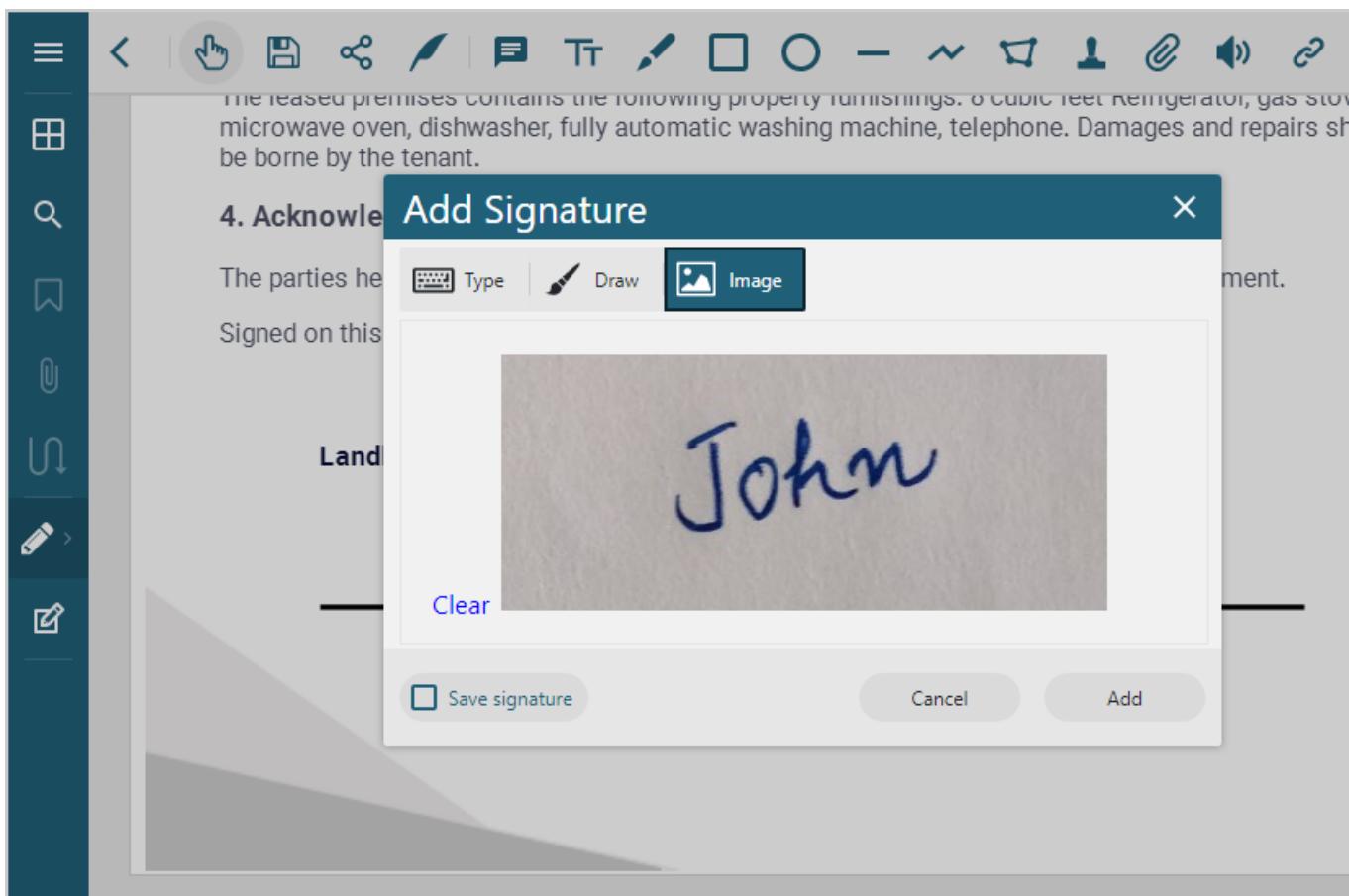
The below GIF demonstrates how to draw, format and add signatures in a PDF document. As can be observed, the signature is added as a Stamp Annotation in the Annotation Editor's property panel. These properties can be used to remove or download signatures, make them printable, change their position, height or width etc.



The below image displays typed signatures with various formatting options available in the 'Add Signature' dialog like font, color, size etc.



The below image displays an uploaded image of signature which can be added into a PDF document.



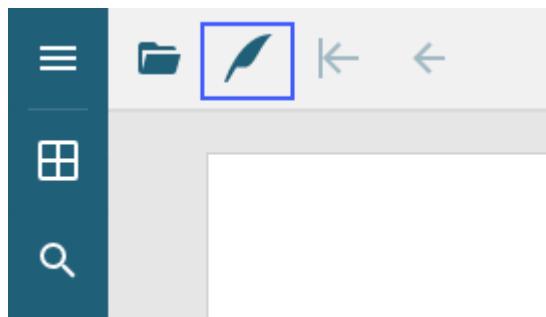
 **Note:** [SupportApi](#) should be configured in order to use Signature Tool, as the editing operation is performed in a PDF document.

Add Signature Tool Button in Viewer Toolbar

The Annotation Editor toolbar contains 'Signature tool' button by default. However, you can also add the button to viewer's toolbar layout by using below code:

Index.cshtml

```
viewer.toolbarLayout.viewer.default.splice(1, 0, 'edit-sign-tool');  
viewer.applyToolbarLayout();
```



Save Signature

You can save signatures by checking the 'Save Signature' checkbox which saves the signature data into browser's local storage for later use.

The tenant agrees to pay for the utilities and other services used in the property on the condition of lease of the Property.



Note: The saved signature data is owned by the active user and can be set by using the **currentUserUsername** property or **userName** option. Therefore, if these properties are changed, saved signature data will also change.

Customize Signature Tool Dialog

The appearance and behavior of 'Add Signature' dialog can be customized by using the signTool option as shown in the below code:

Index.cshtml

```
viewer.options.signTool = {  
    title: 'Please, sign.',  
    selectedTab: 'Draw',  
    hideTabs: true, hideToolbar: true, hideSaveSignature: true,  
    saveSignature: false,  
    penColor: 'black', penWidth: 2,  
    location: 'Center',  
    destinationScale: 1.2  
};
```



 **Note:** The **showSignTool** method takes precedence over the **signTool** option if the settings object is passed as its argument.

For more information, refer [Graphical Signature](#) in GcDocs PDF Viewer demos.

Share and Collaborate

GcDocs PDF Viewer allows you to share a PDF document with other users. The document can be shared in view or/and edit mode and can be worked upon by multiple users in real time. You can also observe the presence of other users and the changes made by them in the shared document, known as real-time collaboration.

The below image shows a PDF document in shared mode which has been edited by multiple users and their changes are visible along with their user names.

The screenshot shows a PDF document with a header 'ACME Inc.' and a title 'Return and Exchange Form'. The document contains several input fields for user information, such as 'Name', 'Address', 'Phone', and 'Email'. There are also checkboxes for selecting options like 'Address Change' or 'Reimburse my original method of payment'. A section for 'Send Refund or Exchange to:' is present, with similar input fields. At the top, there are user profile icons for 'Anonymous' (red) and 'James' (green). A button labeled 'Have Any Questions?' with the text 'Please call us at 800-123-4567.' is visible. The interface includes a toolbar with various icons and a status bar at the bottom.

Once configured to use the [SupportApi](#), a PDF document can be edited in following ways:

- Annotations
- Form fields
- Form filling
- Comments

Set up Collaboration Mode

The collaboration mode, where multiple users can work on a shared document, is designed to work only with persistent client connections. To ensure this, the [SupportApi](#) includes the following nuget package dependencies:

- AspNetCore.SignalR for ASP.NET Core version
- AspNet.SignalR.Core for ASP.NET/OWIN self host version

When a shared document is modified, the server-side SupportApi code instantly pushes the changes to connected clients as soon as the changes become available.

To support persistent client connections for collaboration mode, configure your web application by editing Startup.cs file:

ASP.NET Core Web Application

1. Add SignalR service to the services collection by calling `SupportApi.Connection.GcPdfViewerHub.ConfigureServices(services)`; Inside the `ConfigureServices` method:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddRazorPages();
    // Enable routing:
    services.AddMvc((opts) => { opts.EnableEndpointRouting = false; });
    services.AddRouting();
    // Add SignalR service to the services collection:
    SupportApi.Connection.GcPdfViewerHub.ConfigureServices(services);
}
```

2. Map SignalR hub inside the Configure method as follows:

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.UseEndpoints(endpoints =>
    {
        endpoints.MapRazorPages();
        // Map SignalR hub:
        endpoints.MapHub<SupportApi.Connection.GcPdfViewerHub>("/signalr");
    });
}
```

ASP.NET WebForms Application

```
[assembly: OwinStartup(typeof(SupportApiDemo_WebForms.Startup))]
namespace SupportApiDemo_WebForms {

    public class Startup {
        public void Configuration(IAppBuilder app)
        {
            // ... the rest of the code goes here
            SupportApi.Connection.GcPdfViewerHub.Configure(app);
            // Optionally, you can add known users, these users will be displayed in the user interface as a suggestions pop-up:
            GcPdfViewerController.Settings.AvailableUsers.Add("Anonymous");
            GcPdfViewerController.Settings.AvailableUsers.Add("James");
            GcPdfViewerController.Settings.AvailableUsers.Add("Lisa");
            // For example, you can use your own shared document storage
            // which implements the ICollaborationStorage interface:
            GcPdfViewerController.Settings
                .Collaboration
                .Storage
                .UseCustomStorage(myCloudStorage);
        }
    }
}
```

 **Note:** Using SupportApi in ASP.NET Core projects requires ASP.NET Core 3.0 or later. For more details on how to upgrade an existing ASP.NET Core 2.2 project to 3.0, refer this [link](#).

Store Changes in Collaboration Mode

When you make changes to PDF documents in shared mode, the modified documents are stored in server's memory by default. These changes are discarded in either of the below cases:

- If the server is restarted
- Or after 8 hours, which is the default period for automatically clearing documents in memory

To prevent this, you can do any of the following:

- Use local folder (file system) storage by using below code:

startup.cs

```
GcPdfViewerController.Settings
    .Collaboration
    .Storage
    .UseFileSystem(Path.Combine(env.ContentRootPath, "LocalStorage"), 8);
```

- Change the lifetime of shared documents to retain the changes in shared documents for 24 hours by using below code:

startup.cs

```
GcPdfViewerController.Settings
    .Collaboration
    .Storage
    .UseMemory(24);
```

- Use a custom storage type to store shared documents by implementing ICollaborationStorage interface:

startup.cs

```
public class FileSystemStorage : ICollaborationStorage
{
    private readonly string _directoryPath;
    private object _readLock = new object();
    private object _writeLock = new object();
    public FileSystemStorage(string directoryPath)
    {
        _directoryPath = directoryPath;
    }
    //ICollaborationStorage interface implementation
    public Task<byte[]> ReadData(string key)
    {
        return Task.Factory.StartNew(() =>
        {
            string filePath = Path.Combine(_directoryPath, $"{key}");
            if (File.Exists(filePath))
            {
                lock (_readLock)
                {
                    return File.ReadAllBytes(filePath);
                }
            }
            return null;
        });
    }
    public Task WriteData(string key, byte[] data)
    {
        return Task.Factory.StartNew(() =>
        {
            var filePath = Path.Combine(_directoryPath, $"{key}");
            lock (_writeLock)
            {
```

```
        if (data == null)
    {
        if (File.Exists(filePath))
            File.Delete(filePath);
    }
    else
    {
        File.WriteAllBytes(filePath, data);
    }
}
}
}
```

Share a PDF Document

You can share a PDF document by clicking on the 'share' button in GcDocs PDF Viewer's toolbar.

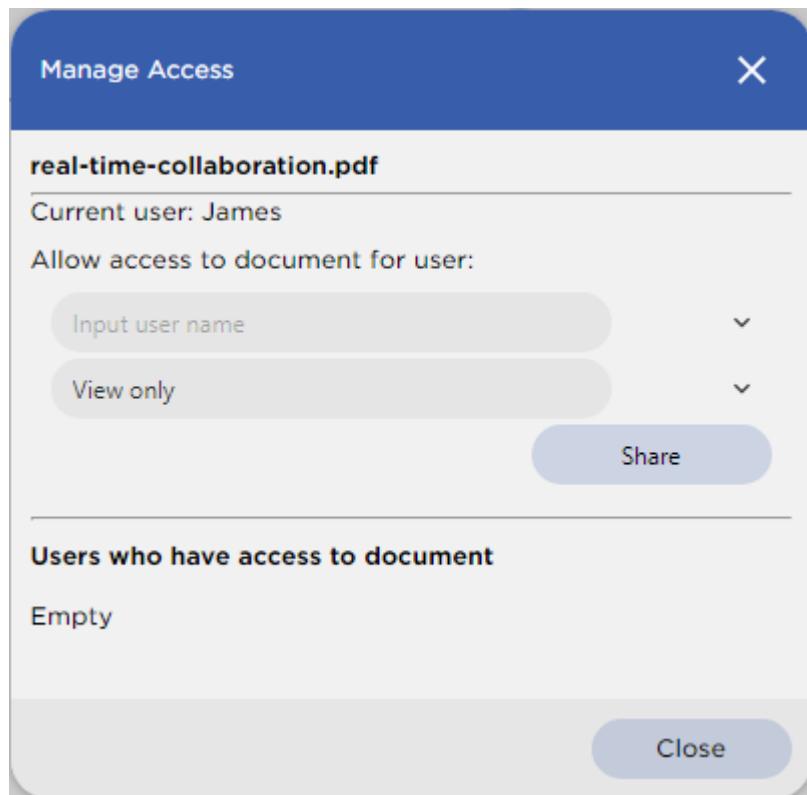


To add the 'share' button in toolbar, use the following code:

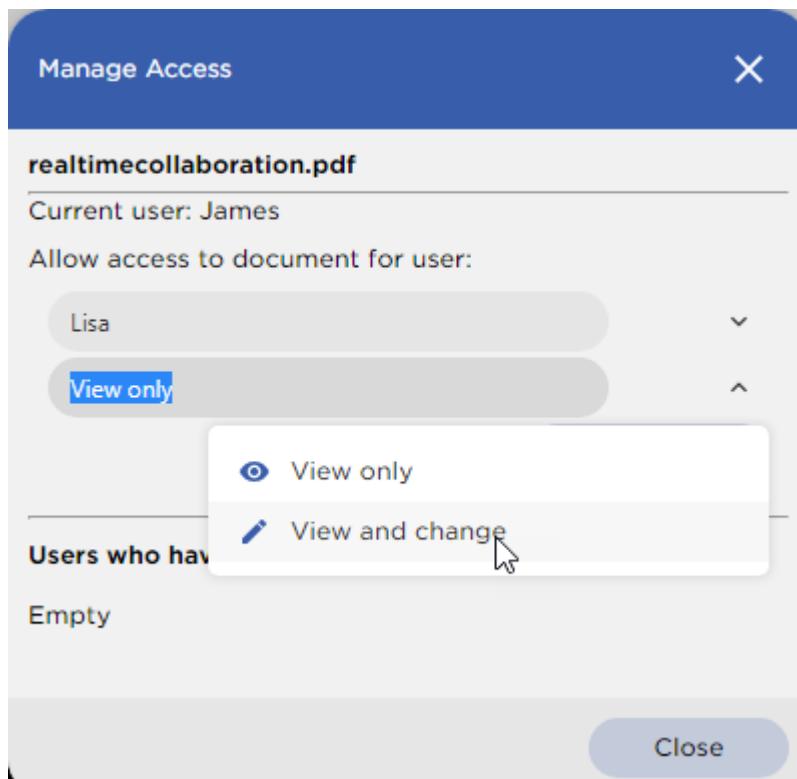
Index.cshtml

```
viewer.toolbarLayout.viewer.default = ["share"];
viewer.applyToolbarLayout();
```

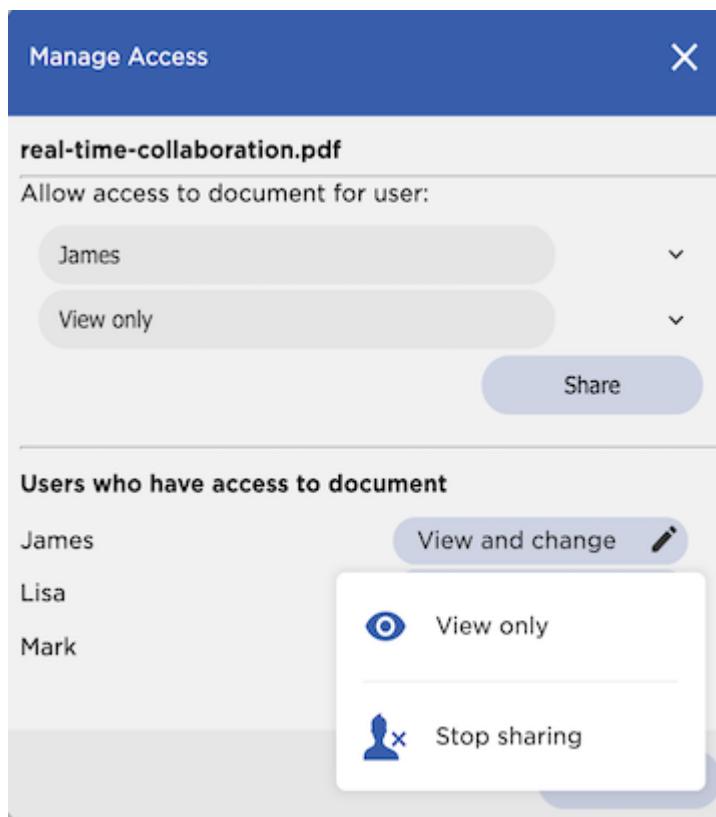
When you click on 'share' button, the 'Manage Access' dialog box opens as shown below:



You can provide access to other users and set their permissions to 'View only' or 'View and Change'.



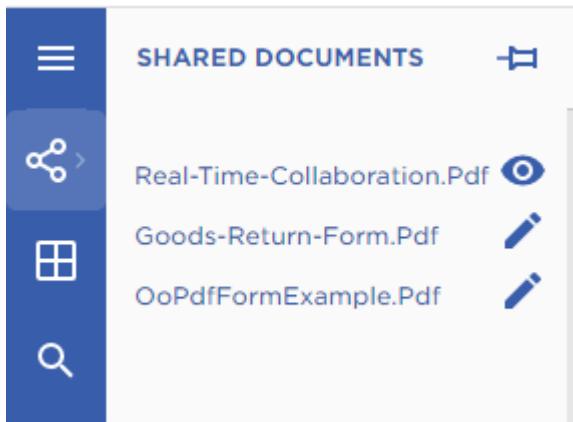
You can also change the access of existing users or stop sharing the document with them.



Note: The PDF document owner can perform all the above operations. Other users cannot edit/delete the document owner or change his permissions. However, other users with edit permissions can add other users, change their permissions or stop sharing the document.

Open a Shared PDF Document

You can open a shared PDF document by using the 'Shared Documents' panel. This panel lists all the documents that the current user has shared and has access to, with information about the access mode.



To add the 'Shared Documents' panel to the sidebar panel, use the following code:

Index.cshtml

```
var viewer = new GcPdfViewer("#root", { supportApi: 'api/pdf-viewer' });
viewer.addSharedDocumentsPanel();
```

Real-time Co-authoring

When multiple users collaborate in a PDF document at the same time, you can see the real-time changes along with the name of users. Whenever a shared document is opened, the 'shared mode' label with the document's access type is shown on the top right corner of the document, as shown below:

shared mode view and change

You can also use "sharing" option in API to configure document sharing options, like:

- Disallow user names unknown to the server by using the following code:

Index.cshtml

```
var viewer = new GcPdfViewer("#root", { userName: "John", sharing: {
    disallowUnknownUsers: true }, supportApi: "api/pdf-viewer" } );
```

- Specify known user names and disallow other user names by using the following code:

Index.cshtml

```
var viewer = new GcPdfViewer("#root", {
    userName: "John",
    sharing: {
        knownUserNames: ["Jaime Smith", "Jane Smith"],
        disallowUnknownUsers: true,
    },
    supportApi: "api/pdf-viewer"
});
```

- Use presenceColors setting to set the exact colors for user presence indicators by using the following code:

Index.cshtml

```
var viewer = new GcPdfViewer("#root", {
    sharing: {
        knownUserNames: ['Jamie Smith', 'Lisa'],
        presenceColors: { 'Anonymous': '#999999', 'Jamie Smith': 'red',
        'Lisa': 'blue' }
    },
    supportApi: "api/pdf-viewer"
});
```

- Use presenceMode setting to change the mode of presence for collaboration users. The possible values for presenceMode setting are:

- 'on' or 'true' - the presence of all users (including the active one) will be shown. This is the default value.
- 'others' - all users except the active user will be shown.
- 'off' or false - users presence will not be shown.

Index.cshtml

```
// change presenceMode to 'others':
var viewer = new GcPdfViewer("#root", {
    sharing: {
        presenceMode: 'others'
    },
    supportApi: "api/pdf-viewer"
});
```

UI Customizations

GcDocs PDF Viewer provides a flexible, customizable and user-friendly interface which can be customized in various ways to suit your requirements.

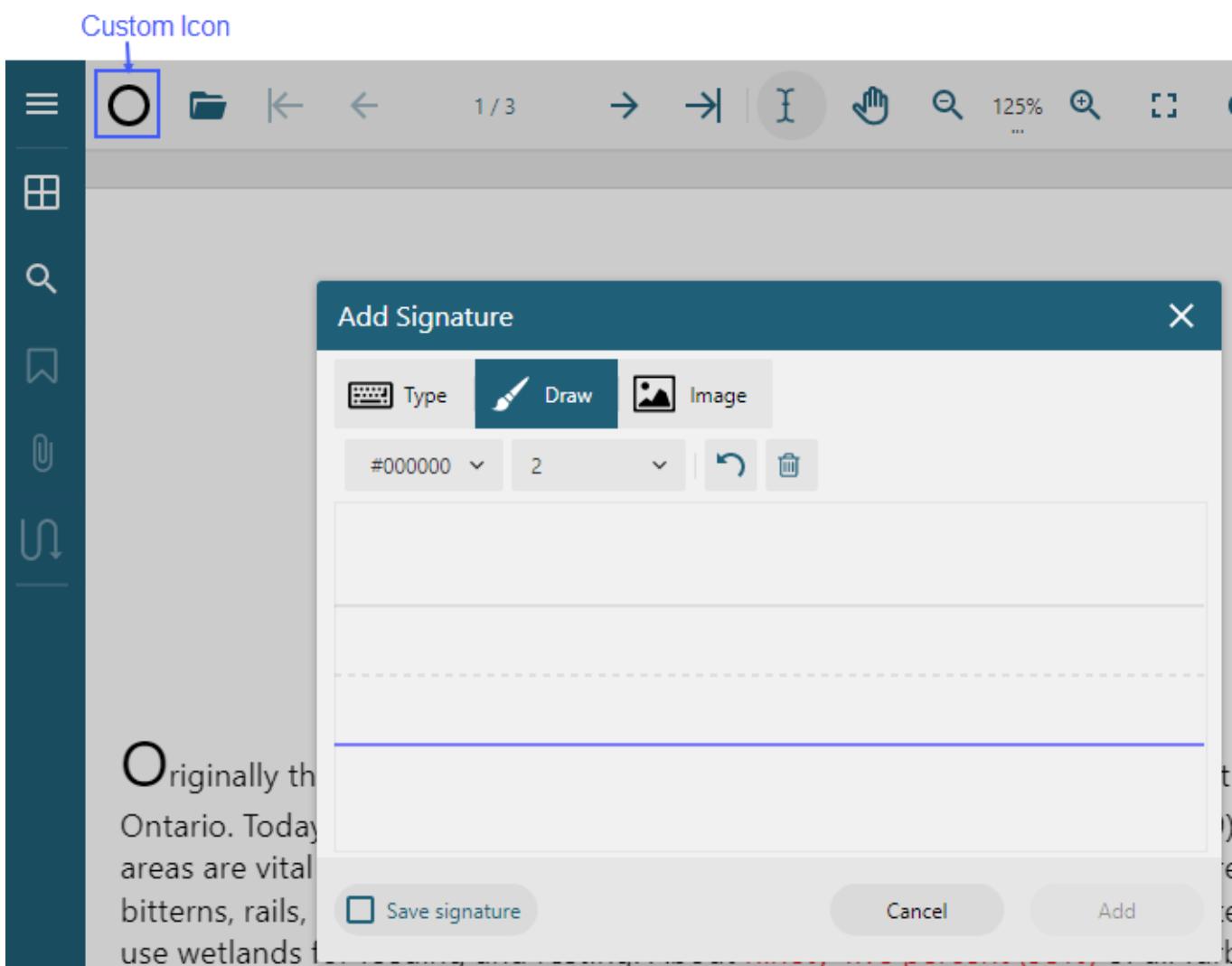
Add a Custom Toolbar Item using SVG Icon

You can use the following code to add a new item in the toolbar using an SVG icon. When clicked, it opens the Graphical signature dialog box after a PDF document is loaded in the viewer.

Index.cshtml

```
viewer.toolbar.addItem({
key: 'custom-sign',
icon: { type: 'svg', content: '<svg xmlns="http://www.w3.org/2000/svg" version="1.1" width="24" height="24" viewBox="0 0 24 24"><path d="M12 0c-6.627 0-12 5.373-12 5.373 12 12 12 12-5.373 12-12-5.373-12-12-12zM12 21c-4.971 0-9-4.029-9-9s4.029-9-9c4.971 0 9 4.029 9 9s-4.029 9-9 9z"></path></svg>' },
title: 'Sign document',
enabled: true,
action: function() {
    // Your action goes here
    // As for example, we will show signature tool dialog:
    viewer.showSignTool();
},
});
```

```
onUpdate: function() {
    return {
        enabled: viewer.hasDocument,
        title: 'Sign'
    }
}
});
// Insert the toolbar item into the default viewer toolbar layout:
viewer.toolbarLayout.viewer.default.splice(0, 0, 'custom-sign');
}
```



Customize Sidebar

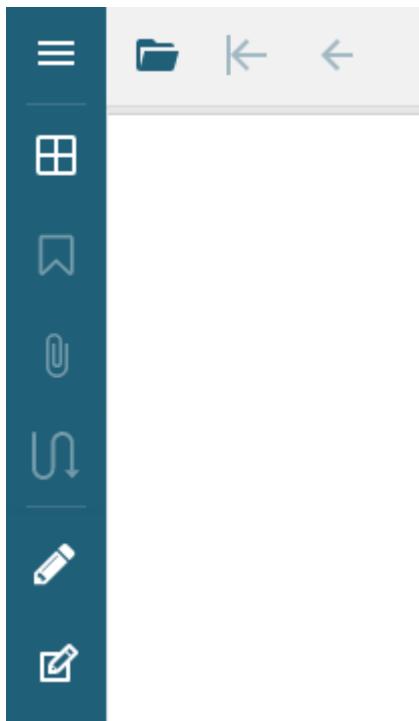
You can customize the sidebar panel of GcDocs PDF Viewer in any of the ways mentioned below:

Toggle Visibility of Sidebar Buttons

You can toggle the visibility of sidebar buttons by using the updatePanel method. It allows you to hide or display an individual panel using a previously saved panel handle. The below example code hides the search panel button:

```
Index.cshtml
```

```
viewer.addThumbnailsPanel();
var searchHandle = viewer.addSearchPanel();
viewer.addOutlinePanel();
viewer.addAttachmentsPanel();
viewer.addArticlesPanel();
viewer.addAnnotationEditorPanel();
viewer.addFormEditorPanel();
//Hide search handle
viewer.updatePanel(searchHandle, { visible: false });
```



Order and Visibility of Sidebar Buttons

You can change the order and visibility of sidebar buttons by using the **layoutPanels** method. It allows you to hide or change the order of sidebar panel buttons. The default sidebar layout is:

```
['DocumentList', 'SharedDocuments', 'Thumbnails', 'Search', 'Outline', 'Attachments', 'Articles', 'sep', 'AnnotationEditor', 'FormEditor']
```

The below example code hides the Annotation Editor button from the sidebar even if the Annotation Editor panel has already been added to the sidebar (using addAnnotationEditorPanel() method):

Index.cshtml

```
viewer.addDefaultPanels();
viewer.addAnnotationEditorPanel();
viewer.addFormEditorPanel();
viewer.addThumbnailsPanel();
viewer.layoutPanels(['Thumbnails', 'Search', 'sep', 'FormEditor']);
```



Custom Sidebar Panel

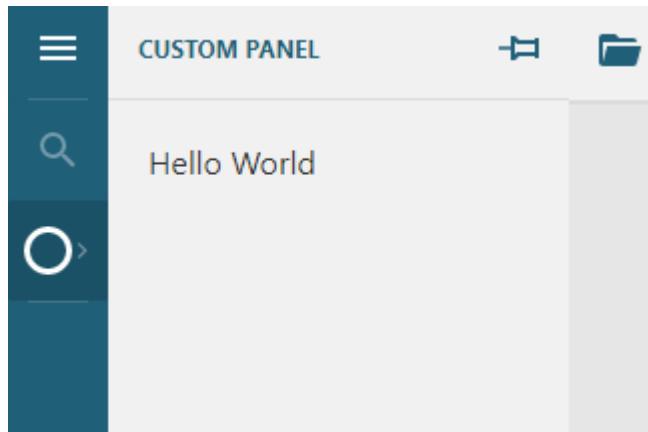
You can create a custom sidebar panel [using React without JSX](#). The below example code shows how to include React CDN links and create the custom sidebar panel:

Index.cshtml

```
<script crossorigin src="https://unpkg.com/react@17/umd/react.production.min.js">
</script>
<script crossorigin src="https://unpkg.com/react-dom@17/umd/react-
dom.production.min.js"></script>
<script>
    function loadPdfViewer(selector) {
        //GcPdfViewer.LicenseKey = 'your_license_key';
        var viewer = new GcPdfViewer(selector, { /* supportApi: { apiUrl: 'api/pdf-
viewer', webSocketUrl: '/signalr' } */ });

        function createPanelContentElement() {
            // Define function component:
            function PanelContentComponent(props) {
                return React.createElement("div", {
                    style: {
                        margin: '20px'
                    }
                }, "Hello ", props.sayHelloToWhat);
            }
            // Create react element:
            return React.createElement(PanelContentComponent, { sayHelloToWhat:
            "World" });
        }
        function createSvgIconElement() {
            return React.createElement("svg", {
                xmlns: "http://www.w3.org/2000/svg",
                version: "1.1",
                width: "24",
                height: "24",
            });
        }
    }
</script>
```

```
viewBox: "0 0 24 24",
fill: "#ffffff",
}, React.createElement("path", {
d: "M12 0c-6.627 0-12 5.373-12 12s5.373 12 12 12 12-5.373 12-12-5.373-12-12-12zM12 21c-4.971 0-9-4.029-9s4.029-9 9-9c4.971 0 9 4.029 9 9s-4.029 9-9 9z"
)));
}
var customPanelHandle = viewer.createPanel(
createPanelContentElement(),
null, 'CustomPanel',
{ icon: { type: "svg", content: createSvgIconElement() },
label: 'Custom panel',
description: 'Custom panel title',
visible: true,
enabled: false,
});
// Add 'CustomPanel' to panels layout:
viewer.layoutPanels(['Thumbnails', 'Search', '*', 'CustomPanel']);
// Enable 'CustomPanel' when needed:
viewer.updatePanel(customPanelHandle, { enabled: true });
}
</script>
```



Document List Panel

In GcDocs PDF Viewer, you can add the document list panel in the left sidebar which helps user view the list of PDF documents and open any of them by clicking on it. To create a list of available PDF documents which you want to add in the document list panel, add the below sample code in the Controller class:

SupportApiController.cs

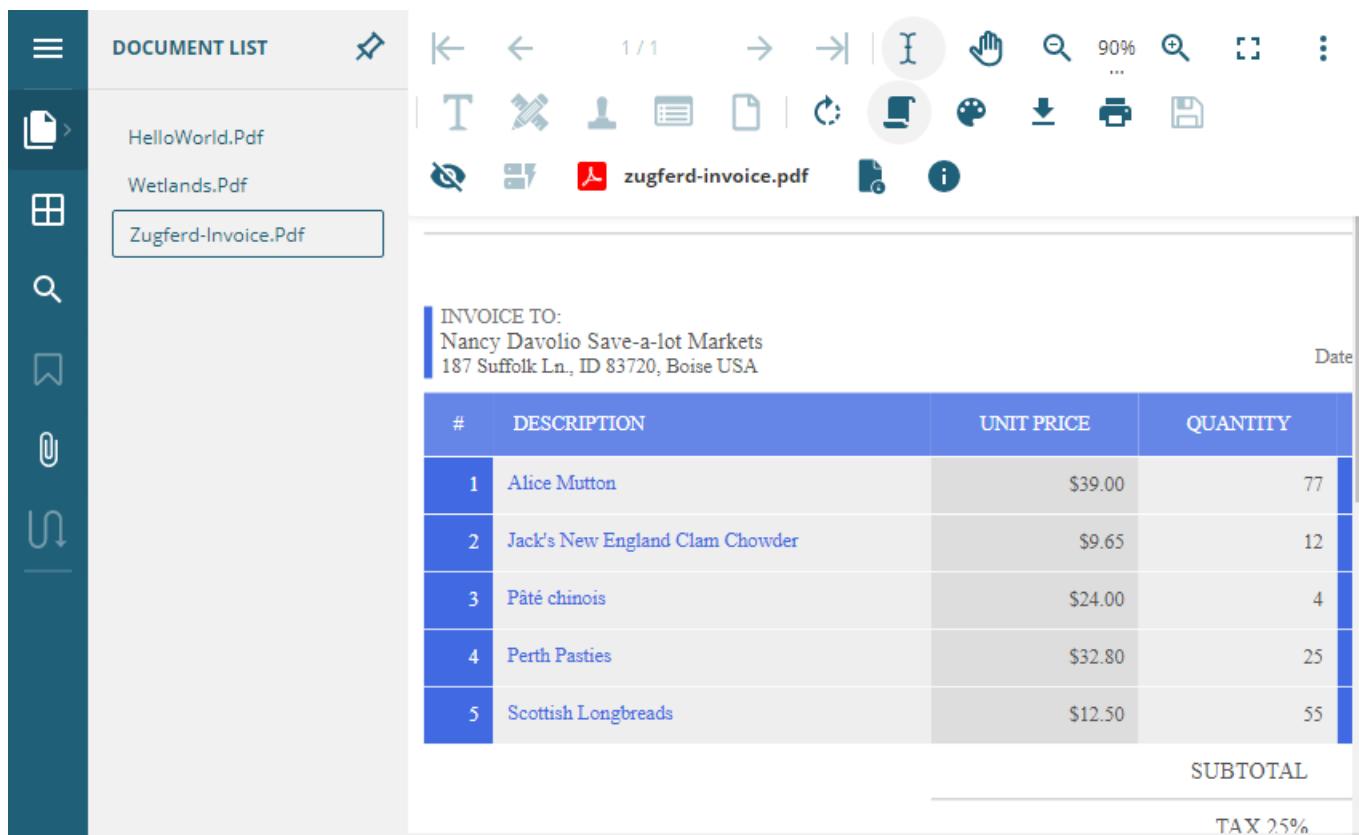
```
public class SupportApiController : Controller
{
    //The method receives requests from the document list panel,
    //see DocumentList below.
    [Route("get-pdf-from-list")]
    public virtual IActionResult GetPdfFromList(string name)
```

```
{  
    Response.Headers["Content-Disposition"] = $"inline; filename=\"{name}\"";  
    var filePath = Path.Combine("Resources", "PDFs", name);  
    return new FileStreamResult(System.IO.File.OpenRead(filePath),  
"application/pdf");  
}  
  
//This method is used by the Document List Panel sample.  
[Route("DocumentList")]  
public object DocumentList()  
{  
    var directoryInfo = new DirectoryInfo(Path.Combine("Resources", "PDFs"));  
    var allPdfs = directoryInfo.GetFiles("*.pdf");  
    return Json(allPdfs.Select(  
        f_ => $"{f_.Name}|api/pdf-viewer/get-pdf-from-list?name={f_.Name}|Open  
{f_.Name}"));  
}  
}
```

To add the document list panel in the left sidebar of the viewer and view the list of available PDF documents, add the following code in Index.cshtml :

Index.cshtml

```
function createPdfViewer(selector, baseOptions) {  
    var options = baseOptions || {};  
    options.documentListUrl = "api/pdf-viewer/DocumentList";  
    var viewer = new GcPdfViewer(selector, options);  
    viewer.addDefaultPanels();  
    var viewerDefault = viewer.toolbarLayout.viewer.default;  
    viewerDefault.splice(viewerDefault.indexOf("open"), 1);  
    var documentListPanelHandle = viewer.addDocumentListPanel();  
    viewer.onAfterOpen.register(function () {  
        viewer.leftSidebar.menu.panels.open(documentListPanelHandle.id);  
    });  
    return viewer;  
}
```

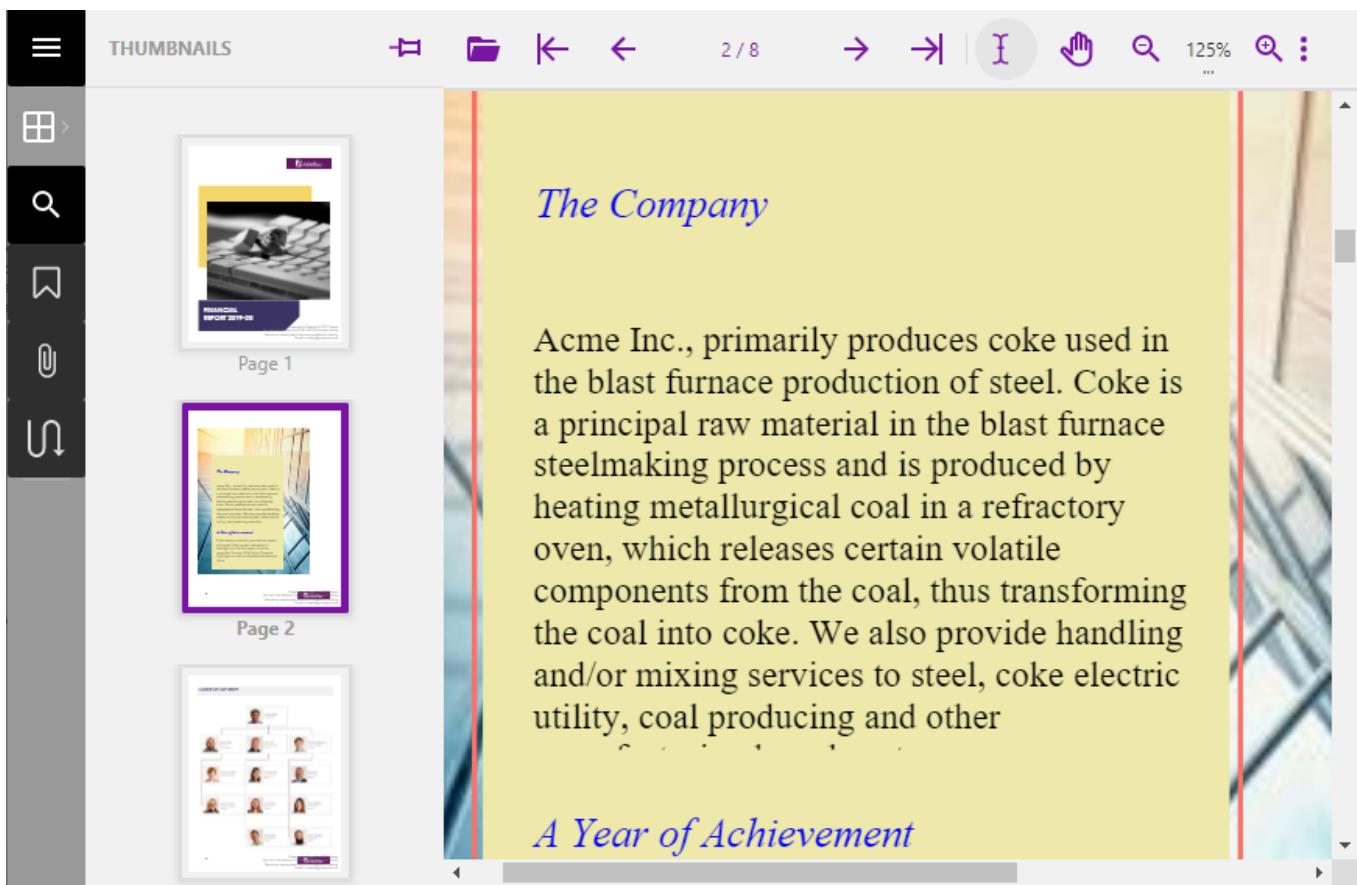


Customize Theme using CSS Styles

You can change the theme of GcDocs PDF Viewer by using CSS styles. The below example code demonstrates how to override the default CSS styles:

Index.cshtml

```
<style>
body .gc-menu__btn-container {
    background-color: #999999;
}
body .gc-btn--accent {
    background-color: #000000;
}
body .gc-btn--accent:not([disabled]):not(.gc-btn--disabled):hover {
    background-color: #000000;
}
body .gc-viewer-host .gc-viewer .gcv-menu .gc-menu__panel-toggle--active .gc-btn {
    background-color: #999999;
}
body .gc-btn[disabled] {
    opacity: 0.7;
}
body .gc-accent-color {
    color: #ff0000;
}
</style>
```



Localization

You can localize the GcDocs PDF Viewer to display the localized UI. The below example code demonstrates how to change the default localization strings to German:

Index.cshtml

```
//German translation of strings
var translation = {
    'error': { 'details': 'Einzelheiten', 'dismiss': 'Entlassen', 'dismiss-all': 'Alle entlassen', 'bad-rotation': 'Rotation kann nicht eingestellt werden, schlechter Rotationswert' }, 'menu': { 'aria-label': 'Menü', 'pin-button-title': 'Pin' }, 'sidebar': { 'expand-btn': 'Expand', 'collapse-btn': 'Zusammenbruch', 'aria-label': 'Sidebar' }, 'cancel-btn': 'Abbrechen', 'toolbar': { 'zoom-fitwidth': 'An Breite anpassen', 'zoom-fitpage': 'Seite anpassen', 'zoom-zoomout': 'Verkleinern', 'zoom-zoomin': 'Vergrößern', 'zoom-menu-header': 'Zoom modus', 'gotofirst': 'Gehe zu zuerst', 'gotoprevious': 'Gehe zu Zurück', 'gonext': 'Weiter', 'gotolast': 'Gehe zu Letzter', 'hist-parent': 'Geschichte: Zurück zum übergeordneten Element', 'hist-back': 'Geschichte: Zurück', 'hist-fwd': 'Geschichte: Weiter', 'movetool': 'Werkzeug bewegen', 'fullscreen': 'Vollbild umschalten', 'refresh': 'Aktualisieren', 'cancel': 'Abbrechen', 'aria-label': 'Symbolleiste', 'cycle-themes': 'Durch verfügbare Themen blättern', 'single-page-view': 'Einzelseitenansicht', 'continuous-view': 'Durchgehende Ansicht', 'show-annotations-fields': 'Anmerkungen/Formularfelder anzeigen', 'hide-annotations-fields': 'Anmerkungen/Formularfelder ausblenden', 'form-filler': 'Formfüller', 'share-document': 'Dokument freigeben', 'download-document': 'Dokument herunterladen', 'save-document': 'Geändertes Dokument speichern', 'new-blank-document': 'Neues leeres Dokument', 'new-blank-page': 'Leere Seite einfügen', }
```

```
'delete-current-page': 'Aktuelle Seite löschen', 'print-document': 'Dokument drucken', 'rotate-document': 'Dokument drehen', 'open-document': 'Dokument öffnen', 'text-selection': 'Textauswahlwerkzeug', 'pan': 'Schwenkwerkzeug', 'add-free-text': 'Freitextanmerkung hinzufügen', 'add-text-note': 'Haftnotiz hinzufügen', 'draw-ink': 'Freihandanmerkung zeichnen', 'draw-square': 'Quadratanmerkung zeichnen', 'draw-line': 'Linienanmerkung zeichnen', 'draw-circle': 'Anmerkung Kreis zeichnen', 'draw-polyline': 'Polylinien-Anmerkung zeichnen', 'draw-polygon': 'Polygon-Anmerkung zeichnen', 'sign-tool': 'Signaturtool', 'add-stamp': 'Stempel hinzufügen', 'add-file-attachment': 'Dateianlage hinzufügen', 'add-sound': 'Klanganmerkung hinzufügen', 'edit-link': 'Link-Anmerkung hinzufügen', 'apply-all-redacts': 'Alle Rechtsakte anwenden', 'redact-region': 'Redact(erase) region', 'select-annotation': 'Anmerkung auswählen', 'select-field': 'Feld auswählen', 'add-text-field': 'Textfeld hinzufügen', 'add-comb-text-field': 'Kammfeld hinzufügen', 'add-password-field': 'Kennwortfeld hinzufügen', 'add-text-area': 'Textbereich hinzufügen', 'add-checkbox': 'Kontrollkästchen hinzufügen', 'add-radio-button': 'Optionsfeld hinzufügen', 'add-push-button': 'Druckknopf hinzufügen', 'add-submit-button': 'Schaltfläche Formular absenden', 'add-reset-button': 'Reset-Formular-Taste hinzufügen', 'add-combobox': 'Kombinationsfeld hinzufügen', 'add-listbox': 'Listenfeld hinzufügen', 'delete-annotations': 'Anmerkungen löschen', 'delete-fields': 'Felder löschen', 'show-annotation-editor': 'Anmerkungs-Editor', 'show-form-editor': 'Formular-Editor', 'toggle-annotation-properties': 'Eigenschaftenfenster umschalten', 'toggle-form-properties': 'Eigenschaftenfenster umschalten', 'show-view-tools': 'Editor schließen, zurück in den Ansichtsmodus', 'undo-changes': 'Änderungen rückgängig machen', 'redo-changes': 'Änderungen wiederherstellen', 'confirm-ok': 'OK', 'confirm-cancel': 'Abbrechen', 'document-properties': 'Dokumenteigenschaften', 'about': 'Info' }, 'errors': { 'noHostElement': 'Das Hostelement wurde nicht gefunden.', 'propertyCannotBeChanged': 'Eigenschaft kann nicht geändert werden.', 'field-already-exists': 'Feld mit dem Namen {{fieldName}} ist bereits vorhanden.', 'cannot-save-document-format': 'Das Dokument kann nicht gespeichert werden. {{reason}}', 'base-viewer-dispose-warn': 'Fehler beim Verwerfen des Basis-Viewers. (dies ist kein schwerwiegender Fehler)', 'dnd-error': '', 'dnd-error-download-image-from-url': '', 'openSharedDocumentError': 'Das freigegebene Dokument kann nicht geöffnet werden. {{reason}}', 'proLicenseRequired': { 'message': 'Für die Verwendung der Editierfunktionen ist eine Professional-Lizenz erforderlich.' }, 'cannotOpenDocumentOnServer': 'Dokument kann nicht auf dem Server geöffnet werden. Bearbeitung deaktiviert.', 'error-opening-document': 'Fehler beim Öffnen des Dokuments', 'cannotedit-field-locked': 'Bearbeitung nicht möglich. Das Feld ist gesperrt.', 'cannotedit-annotation-locked': 'Bearbeitung nicht möglich. Die Anmerkung ist gesperrt.', 'openforViewingOnly': 'Das Dokument ist nur zur Ansicht geöffnet. Bearbeitungswerzeuge sind deaktiviert.', 'supportApiNotAvailable': 'Support API-Server nicht verfügbar. Bearbeitungswerzeuge deaktiviert.', 'print-canceled-by-user': 'Vom Benutzer abgebrochen' }, 'top-bottom-panel': { 'aria-label': 'Zusätzliches Bedienfeld' }, 'document-view': { 'aria-label': 'Dokumentansicht' }, 'progress': { 'page': 'Seite', 'titles': { 'saving-document': 'Saving document' } }, 'messages': { 'preparing-document-uploading-modifications': 'Preparing document, uploading modifications...' } }, 'search': { 'match-case': 'Match Case', 'whole-word': 'Ganzes Wort', 'cancel-btn': 'Abbrechen', 'start-search-btn': 'Suchen', 'clear-btn': 'Clear', 'more-results-btn': 'Mehr Ergebnisse', 'search-results': 'Suchergebnisse', 'search-cancelled-msg': 'Suche auf Seite {{page}} abgebrochen', 'didn-find-msg': 'Ich habe nichts gefunden.' }, 'paneltitle': 'Suchen' }, 'annotation': { 'properties': { 'radios-in-unison': 'Funkgeräte im Einklang', 'printable':
```

```
'Druckbar', 'color': 'Farbe', 'fill-color': 'Füllfarbe', 'icon': 'Symbol', 'line-type': 'Zeilentyp', 'line-start': 'Line start', 'line-end': 'Leitungsende', 'initially-open': 'Anfänglich geöffnet', 'required': 'Erforderlich', 'callout-line-end': 'Zeilenende', 'text-align': 'Align', 'annotation-state': 'Staat', 'annotation-state-model': 'Staatsmodell', 'font-size': 'Schriftgröße', 'border-width': 'Breite', 'border-style': 'Stil', 'border-color': 'Farbe', 'popup-parent-annotation': 'Elternanmerkung', 'irt-annotation': 'In Erwiderung auf', 'file-name': 'Name', 'file': 'Datei', 'sound': 'Sound', 'image': 'Image', 'background-color': 'Backcolor', 'foreground-color': 'Forecolor', 'radio-export-value': 'Exportwert', 'field-value': 'Wert', 'submit-url': 'URL übermitteln', 'link-type': 'Typ', 'link-dest-type': 'Zieltyp', 'link-dest-loading': { 'left-column': 'Bestimmungsort', 'right-column-format': '{destId} wird geladen...' }, 'destination-x': 'X', 'destination-y': 'Y', 'destination-w': 'Breite', 'destination-h': 'Höhe', 'destination-scale': 'Skalieren', 'pageNumber': 'Seitennummer', 'url': 'URL', 'new-window': 'Neues Fenster', 'action': 'Maßnahmen', 'js-action': 'JS-Aktion', 'text': 'Text', 'field-value-on-off': 'Wert', 'choice-options': 'Optionen', 'choice-multi-select': 'Mehrfachauswahl', 'text-max-length': 'Max. Länge', 'combs-count': 'Kämme zählen', 'area-text': 'Text', 'field-name': 'Name', 'read-only': 'Schreibgeschützt', 'author': 'Autor', 'subject': 'Gegenstand', 'is-rich-text': 'Rich Text', 'bounds-width': 'Breite', 'bounds-height': 'Höhe', 'bounds-x': 'X', 'bounds-y': 'Y', 'position-x': 'X', 'position-y': 'Y', 'redacted-fill-color': 'Füllfarbe', 'redacted-overlay-text': 'Overlay Text', 'redacted-text-align': 'Align', 'redacted-repeat-text': 'Text wiederholen', 'redaction-mark-border-color': 'Randfarbe', 'redaction-mark-fill-color': 'Füllfarbe' }, 'property-groups': { 'callout': 'Callout', 'border': 'Grenze', 'link-destination': 'Bestimmungsort', 'bounds': 'Grenzen', 'position': 'Position', 'redacted-area': 'Redacted Area', 'redaction-mark': 'Redaktionszeichen' }, 'enums': { 'line-ending': { 'Square': 'Quadratisch', 'Circle': 'Kreis', 'Diamond': 'Diamant', 'OpenArrow': 'OpenArrow', 'ClosedArrow': 'ClosedArrow', 'None': 'Keine', 'Butt': 'Butt', 'ROpenArrow': 'ROpenArrow', 'RClosedArrow': 'RClosedArrow', 'Slash': 'Schrägstrich' }, 'sound-icon': { 'speaker': 'Lautsprecher', 'mic': 'Mikrofon' }, 'attachment-icon': { 'graph': 'Graph', 'push-pin': 'PushPin', 'paperclip': 'Büroklammer', 'tag': 'Tag' }, 'text-icon': { 'comment': 'Kommentar', 'key': 'Key', 'note': 'Anmerkung', 'help': 'Hilfe', 'new-paragraph': 'NewParagraph', 'paragraph': 'Absatz', 'insert': 'Einfügen' }, 'border-type': { 'solid': 'Solid', 'dashed': 'gestrichelt', 'beveled': 'abgeschrägt', 'inset': 'Inset', 'underline': 'Unterstreichen' }, 'link-type': { 'url': 'URL', 'dest': 'Bestimmungsort', 'action': 'Maßnahmen', 'js': 'JS-Aktion' }, 'dest-type': { 'xyz': 'XYZ', 'fit': 'Fit', 'fit-h': 'FitH', 'fit-v': 'FitV', 'fit-r': 'FitR', 'fit-b': 'FitB', 'fit-bh': 'FitBH', 'fit-bv': 'FitBV', 'first-page': 'Erste Seite', 'last-page': 'Letzte Seite', 'next-page': 'Nächste Seite', 'prev-page': 'Vorherige Seite', 'go-back': 'Zurück', 'go-forward': 'Weiter' }, 'on-off': { 'on': 'Ein', 'off': 'Aus' }, 'text-align': { 'left': 'Links', 'center': 'Mitte', 'right': 'Rechts' }, 'note-status': { 'none': 'Keine', 'accepted': 'Akzeptiert', 'cancelled': 'Storniert', 'completed': 'Abgeschlossen', 'rejected': 'Abgelehnt', 'marked': 'Markiert', 'unmarked': 'Nicht markiert' }, 'annotation-state-model': { 'marked': 'Markiert', 'review': 'Überprüfung' }, 'callout-line-type': { 'none': { 'label': 'Keine', 'title': 'Ohne Legende' }, 'simple': { 'label': 'Simple Line', 'title': 'Simple line callout' }, 'corner': { 'label': 'Ecklinie', 'title': 'Legende der Ecklinie' } } }, 'editors': { 'plain-text-editor': { 'empty-placeholder': '<empty>', 'multiple-values-placeholder': '<leer>' }, 'nullable-number-editor': { 'empty-placeholder': '<empty>', 'multiple-values-placeholder': '<leer>' }, 'number-editor': { 'empty-placeholder': '<empty>', 'multiple-values-placeholder': '<leer>' } },
```

```
'float-editor': { 'empty-placeholder': '<empty>', 'multiple-values-placeholder': '<leer>' }, 'key-value-editor': { 'empty-name': '<empty>', 'empty-value': '<empty>', 'key-display-format': 'Bezeichnung: {{value}}', 'value-display-format': 'Wert: {{value}}' }, 'collection-editor': { 'close-btn-title': 'Schließen', 'show-btn-title': 'Elemente anzeigen', 'add-btn-text': 'Hinzufügen', 'add-btn-title': 'Element hinzufügen', 'empty': 'Sammlung ist leer', 'items': 'Gegenstände' }, 'bool-editor': { 'text-true': 'True', 'text-false': 'Falsch', 'text-undefined': 'Undefiniert' }, 'parent-id-editor': { 'none-item': { 'label': 'Keine', 'title': 'Nicht ausgewählt' } }, 'text-area-editor': { 'type-text-here': 'Text hier eingeben', 'cancel-btn': 'Abbrechen', 'ok-btn': 'OK', 'cancel-btn-title': 'Änderungen abbrechen und zurücksetzen', 'ok-btn-title': 'Änderungen übernehmen', 'edit-btn': 'Bearbeiten' }, 'js-code-area-editor': { 'type-code-here': '<Code hier eingeben>', 'cancel-btn': 'Abbrechen', 'ok-btn': 'OK', 'cancel-btn-title': 'Änderungen zurücksetzen', 'ok-btn-title': 'Änderungen übernehmen', 'edit-code-btn': 'Code bearbeiten' }, 'property-list': { 'no-annotations-label': 'Keine Anmerkungen in diesem Dokument', 'no-form-fields-label': 'Keine Formularfelder in diesem Dokument', 'delete-field-btn-title': 'Feld löschen', 'delete-annotation-btn-title': 'Anmerkung löschen', 'delete-field-btn': 'Löschen', 'delete-annotation-btn': 'Löschen', 'clone-field-btn-title': 'Feld klonen', 'clone-annotation-btn-title': 'Clone-Anmerkung', 'clone-field-btn': 'Clone', 'clone-annotation-btn': 'Clone', 'drag-handle-title': 'Ziehen, um die Reihenfolge der Unterfenster zu ändern', 'revert-redact-btn': { 'label': 'Revert', 'title': 'Redaktion rückgängig machen' }, 'apply-redact-btn': { 'label': 'Anwenden', 'title': 'Redaktion anwenden' }, 'revert-content-btn': { 'label': 'Revert', 'title': 'Konvertierung in Inhalt zurücksetzen' }, 'make-content-btn': { 'label': 'Convert', 'title': 'In Inhalt konvertieren' }, 'page-label': 'PAGE {{number}}', 'page-title-format': 'PAGE {{pageNumber}} Größe: {{pageWidth}} X {{pageHeight}}pt {{pageWidthIn}} X {{pageHeightIn}}in', 'emptyListPlaceholder': 'Es sind keine anzuzeigenden Eigenschaften vorhanden', 'field': { 'title-format': '{{label}}, ID: {{id}}', 'types': { 'signature-field': 'Feld Signatur', 'comb-text': 'Comb-text', 'text-area': 'Textbereich', 'password': 'Kennwort', 'text': 'Text', 'check-box': 'Kontrollkästchen', 'radio': 'Radio', 'submit-form': 'Formular absenden', 'reset-form': 'Formular zurücksetzen', 'push': 'Push', 'unknown-button': 'Unbekannte Taste', 'combo-box': 'ComboBox', 'list-box': 'ListBox', 'unknown-type': 'Unbekannter {{type}}' }, 'annotation': { 'title-format': '{{label}}, ID: {{id}}', 'decorator': { 'hidden': '(versteckt)', 'status-title-format': '({{status}} von {{user}})' }, 'reply': '(antworten)' }, 'parent-item-ref': { 'label': '{{type}} {{id}}', 'title': '{{type}} {{id}}' }, 'types': { 'text': 'Text', 'link': 'Link', 'freetext': 'FreeText', 'line': 'Line', 'square': 'Quadratisch', 'circle': 'Kreis', 'polygon': 'Polygon', 'polyline': 'PolyLine', 'highlight': 'Hervorheben', 'underline': 'Unterstreichen', 'squiggly': 'Squiggly', 'strikeout': 'Strikeout', 'stamp': 'Stempel', 'caret': 'Caret', 'ink': 'Tinte', 'popup': 'Popup', 'fileattachment': 'FileAttachment', 'sound': 'Sound', 'movie': 'Film', 'widget': 'Widget', 'screen': 'Bildschirm', 'printermark': 'PrinterMark', 'trapnet': 'TrapNet', 'watermark': 'Wasserzeichen', 'redact': 'Redact', 'signature': 'Unterschrift', 'threadbead': 'ThreadBead', 'radiobutton': 'RadioButton', 'checkbox': 'Kontrollkästchen', 'pushbutton': 'PushButton', 'choice': 'Choice', 'textwidget': 'TextWidget' } } }, 'choice-options-editor': { 'edit-items-format': '{{count}} Elemente bearbeiten' }, 'color-editor': { 'text-palettes': 'Paletten', 'text-color-picker': 'Picker', 'text-web-colors': 'Web-Farben', 'text-opacity': 'Deckkraft', 'text-standard-colors': 'Standardfarben', 'text-hue': 'Hue', 'text-saturation': 'Saturation', 'text-lightness': 'Leichtigkeit', 'text-hex': 'Hex', 'text-r': 'R', 'text-g': 'G', 'text-
```

```
b': 'B', 'webColorNames': { 'transparent': 'Transparent', 'black': 'Schwarz',  
'darkslategray': 'DarkSlateGray', 'slategray': 'SlateGray', 'lightslategray':  
'LightSlateGray', 'dimgray': 'DimGray', 'gray': 'Grau', 'darkgray': 'DarkGray',  
'silver': 'Silber', 'lightgrey': 'LightGrey', 'gainsboro': 'Gainsboro', 'whitesmoke':  
'WhiteSmoke', 'white': 'Weiß', 'snow': 'Schnee', 'honeydew': 'HoneyDew', 'mintcream':  
'MintCream', 'azure': 'Azure', 'aliceblue': 'AliceBlue', 'ghostwhite': 'GhostWhite',  
'seashell': 'SeaShell', 'beige': 'Beige', 'oldlace': 'OldLace', 'floralwhite':  
'FloralWhite', 'ivory': 'Elfenbein', 'antiquewhite': 'AntiqueWhite', 'linen':  
'Leinen', 'lavenderblush': 'LavenderBlush', 'mistyrose': 'MistyRose', 'pink': 'Pink',  
'lightpink': 'LightPink', 'hotpink': 'HotPink', 'deppink': 'DeepPink',  
'palevioletred': 'PaleVioletRed', 'mediumvioletred': 'MediumVioletRed',  
'lightsalmon': 'LightSalmon', 'salmon': 'Lachs', 'darksalmon': 'DarkSalmon',  
'lightcoral': 'LightCoral', 'indianred': 'IndianRed', 'crimson': 'Crimson',  
'firebrick': 'FireBrick', 'darkred': 'DarkRed', 'red': 'Rot', 'orangered':  
'OrangeRed', 'tomato': 'Tomate', 'coral': 'Koralle', 'darkorange': 'DarkOrange',  
'orange': 'Orange', 'yellow': 'Gelb', 'lightyellow': 'LightYellow', 'lemonchiffon':  
'LemonChiffon', 'lightgoldenrodyellow': 'LightGoldenrodYellow', 'papayawhip':  
'PapayaWhip', 'moccasin': 'Moccasin', 'peachpuff': 'PeachPuff', 'palegoldenrod':  
'PaleGoldenrod', 'khaki': 'Khaki', 'darkkhaki': 'DarkKhaki', 'gold': 'Gold',  
'cornsilk': 'Cornsilk', 'blanchedalmond': 'BlanchedAlmond', 'bisque': 'Bisque',  
'navajowhite': 'NavajoWhite', 'wheat': 'Weizen', 'burlywood': 'BurlyWood', 'tan':  
'Tan', 'rosybrown': 'RosyBrown', 'sandybrown': 'SandyBrown', 'goldenrod':  
'Goldenrod', 'darkgoldenrod': 'DarkGoldenrod', 'peru': 'Peru', 'chocolate':  
'Schokolade', 'saddlebrown': 'SaddleBrown', 'sienna': 'Sienna', 'brown': 'Braun',  
'maroon': 'Maroon', 'darkolivegreen': 'DarkOliveGreen', 'olive': 'Olive',  
'olivedrab': 'OliveDrab', 'yellowgreen': 'YellowGreen', 'limegreen': 'LimeGreen',  
'lime': 'Lime', 'lawngreen': 'LawnGreen', 'chartreuse': 'Chartreuse', 'greenyellow':  
'GreenYellow', 'springgreen': 'SpringGreen', 'mediumspringgreen':  
'MediumSpringGreen', 'lightgreen': 'LightGreen', 'palegreen': 'PaleGreen',  
'darkseagreen': 'DarkSeaGreen', 'mediumaquamarine': 'MediumAquamarine',  
'mediumseagreen': 'MediumSeaGreen', 'seagreen': 'SeaGreen', 'forestgreen':  
'ForestGreen', 'green': 'Green', 'darkgreen': 'DarkGreen', 'aqua': 'Aqua', 'cyan':  
'Cyan', 'lightcyan': 'LightCyan', 'paleturquoise': 'PaleTurquoise', 'aquamarine':  
'Aquamarine', 'turquoise': 'Turquoise', 'mediumturquoise': 'MediumTurquoise',  
'darkturquoise': 'DarkTurquoise', 'lightseagreen': 'LightSeaGreen', 'cadetblue':  
'CadetBlue', 'darkcyan': 'DarkCyan', 'teal': 'Teal', 'lightsteelblue':  
'LightSteelBlue', 'powderblue': 'PowderBlue', 'lightblue': 'LightBlue', 'skyblue':  
'SkyBlue', 'lightskyblue': 'LightSkyBlue', 'deepskyblue': 'DeepSkyBlue',  
'dodgerblue': 'DodgerBlue', 'cornflowerblue': 'CornflowerBlue', 'steelblue':  
'SteelBlue', 'royalblue': 'RoyalBlue', 'blue': 'Blue', 'mediumblue': 'MediumBlue',  
'darkblue': 'DarkBlue', 'navy': 'Navy', 'midnightblue': 'MidnightBlue', 'lavender':  
'Lavender', 'thistle': 'Thistle', 'plum': 'Plum', 'violet': 'Violet', 'orchid':  
'Orchid', 'fuchsia': 'Fuchsia', 'magenta': 'Magenta', 'mediumorchid': 'MediumOrchid',  
'mediumpurple': 'MediumPurple', 'blueviolet': 'BlueViolet', 'darkviolet':  
'DarkViolet', 'darkorchid': 'DarkOrchid', 'darkmagenta': 'DarkMagenta', 'purple':  
'Purple', 'indigo': 'Indigo', 'darkslateblue': 'DarkSlateBlue', 'rebeccapurple':  
'RebeccaPurple', 'slateblue': 'SlateBlue', 'mediumslateblue': 'MediumSlateBlue' } },  
'file-editor': { 'select-file': { 'title': 'Datei auswählen' }, 'remove-file': {  
'title': 'Datei entfernen' }, 'download-file': { 'title': 'Datei herunterladen' },  
'no-file': { 'label': 'Keine Datei' } }, 'image-file-editor': { 'no-image': {  
'label': 'Kein Bild' }, 'select-image': { 'title': 'Bilddatei auswählen' }, 'remove-
```

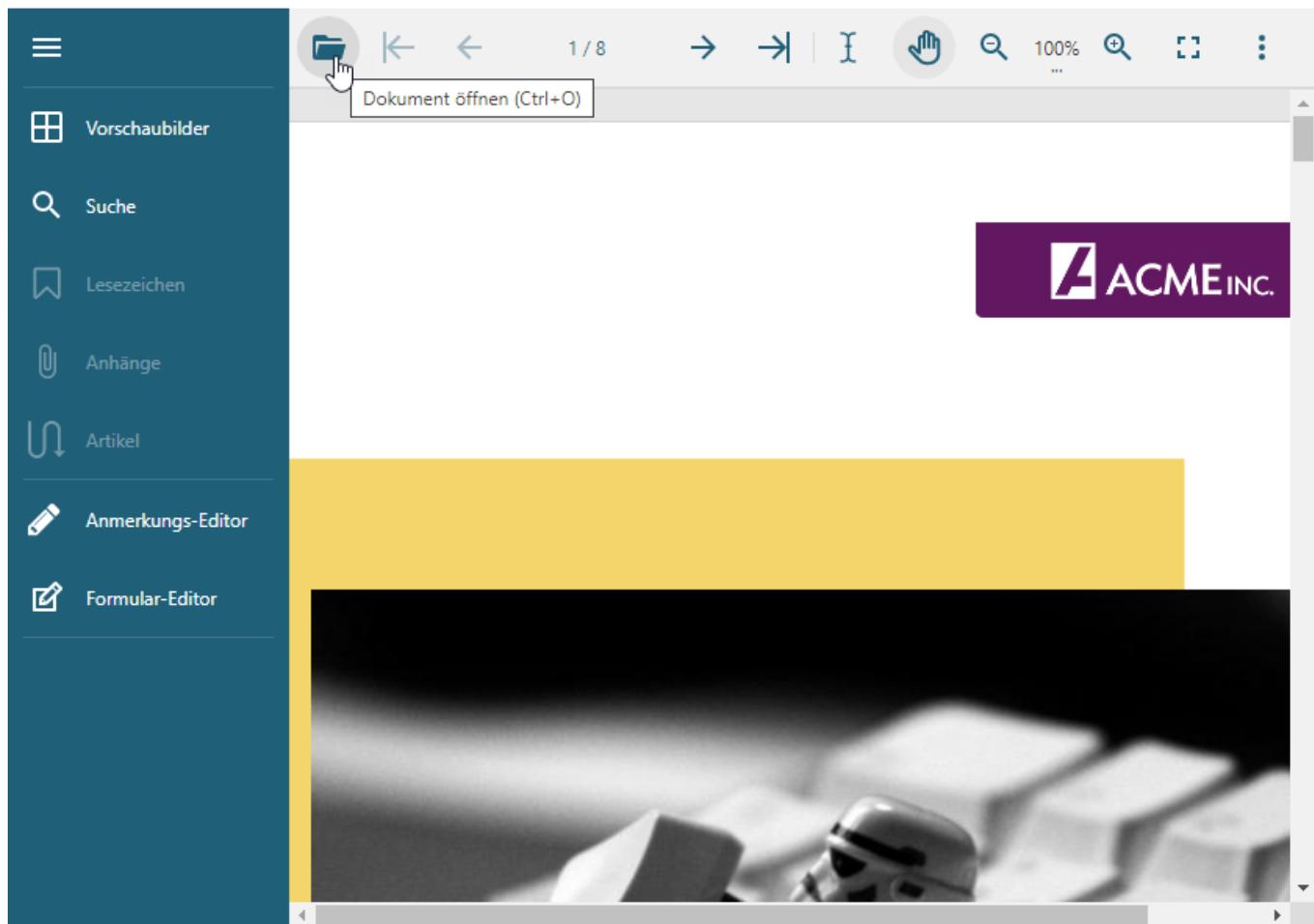
```
image': { 'title': 'Bild entfernen' }, 'download-image': { 'title': 'Image herunterladen' }, 'reset-aspect-ratio': { 'title': 'Bild-Seitenverhältnis zurücksetzen', 'label': 'Seitenverhältnis zurücksetzen' } }, 'link-dest-type-editor': { 'label-xyz': 'Bitte geben Sie die Zielseitennummer, x, y Koordinaten und Maßstab ein. Die x-, y-Koordinaten und der Maßstab sind optional.' }, 'sound-file-editor': { 'select-audio': { 'title': 'Audiodatei auswählen' }, 'remove-audio': { 'title': 'Audio entfernen' }, 'download-audio': { 'title': 'Audio herunterladen' }, 'no-audio': { 'label': 'Kein Audio' } }, 'annotation-editor': { 'hide-list-title': 'Anmerkungsliste ausblenden', 'show-list-title': 'Anmerkungsliste anzeigen', 'label': 'Anmerkungs-Editor', 'title': 'Anmerkungs-Editor' }, 'form-editor': { 'hide-list-title': 'Feldliste ausblenden', 'show-list-title': 'Feldliste anzeigen', 'label': 'Formular-Editor', 'title': 'Formular-Editor' }, 'buttons': { 'close': { 'title': 'Editor schließen' } }, 'comments': { 'no-comments-label': 'Noch keine Kommentare', 'no-comments-details': 'Alle Kommentare zu diesem Dokument werden hier angezeigt.', 'add-reply-placeholder': 'Antwort hinzufügen...', 'cancel-reply-button': { 'title': 'Abbrechen (ESC)', 'label': 'Abbrechen' }, 'post-reply-button': { 'title': 'POST (STRG+EINGABETASTE)', 'label': 'Post' }, 'page-label': 'Kommentare auf Seite {{number}}', 'menu': { 'actions-header': 'Maßnahmen', 'reply-action': { 'label': 'Antworten', 'title': 'Antwort hinzufügen' }, 'delete-action': { 'label': 'Löschen', 'title': 'Kommentar löschen' }, 'status-header': 'Status', 'menu-trigger-title': 'Aktionen' } }, 'annotations': { 'text-annotation': { 'status-title-format': '{{status}} by:\n{{user}}', 'statuses-single-brief-format': '1 Status', 'statuses-brief-format': '{{count}} Status', 'replies-single-brief-format': '1 Antwort', 'replies-brief-format': '{{count}} Antworten' } }, 'annotation-defaults': { 'file-attachment': { 'default-filename': 'Anlage' }, 'sound-annotation': { 'default-filename': 'sound.wav' }, 'stamp-annotation': { 'default-filename': 'image.png' } }, 'push-button': { 'fieldValue': 'Drücken' }, 'reset-button': { 'fieldValue': 'Zurücksetzen' }, 'submit-button': { 'fieldValue': 'Senden' }, 'combo-box': { 'choice-1': 'Auswahl 1', 'choice-2': 'Auswahl 2', 'choice-3': 'Auswahl 3' }, 'list-box': { 'choice-1': 'Auswahl 1', 'choice-2': 'Auswahl 2', 'choice-3': 'Auswahl 3' }, 'default-user-name': 'Anonym' }, 'context-menu': { 'show-comment-panel': { 'label': 'Kommentarfeld anzeigen', 'title': 'Kommentarfeld anzeigen' }, 'paste': { 'label': 'Einfügen (Strg+V)', 'title': 'Einfügen (Strg+V)' }, 'cut': { 'label': 'Ausschneiden (Strg+X)', 'title': 'Ausschneiden (Strg+X)' }, 'copy': { 'label': 'Kopieren (Strg+C)', 'title': 'Kopieren (Strg+C)' }, 'delete': { 'label': 'Löschen (DEL)', 'title': 'Löschen (DEL)' }, 'add-link-over-annotation': { 'label': 'Add Link', 'title': 'Add Link Annotation over this annotation' }, 'move-to-previous-page': { 'label': 'Move to prev page', 'title': 'Move to prev page' }, 'move-to-next-page': { 'label': 'Zur nächsten Seite', 'title': 'Zur nächsten Seite' }, 'add-link-over-text': { 'label': 'Add Link', 'title': 'Add Link Annotation over selected text' }, 'copy-text': { 'label': 'Kopieren (Strg+C)', 'title': 'Ausgewählten Text kopieren (Strg+C)' }, 'print-document': { 'label': 'Drucken (Strg+P)', 'title': 'Dokument drucken (Strg+P)' }, 'add-sticky-note': { 'label': 'Haftnotiz hinzufügen', 'title': 'Haftnotiz hinzufügen' } }, 'panels': { 'documents-list': { 'label': 'Dokumentenliste', 'title': 'Dokumentliste' }, 'bookmarks': { 'label': 'Leszeichen', 'title': 'Leszeichen' }, 'thumbnails': { 'label': 'Vorschaubilder', 'title': 'Vorschaubilder', 'thumbnail-page-label': '{{pageLabel}}', 'thumbnail-page-number': 'Seite {{pageNumber}}' }, 'articles': { 'label': 'Artikel', 'title': 'Artikel-Threads' }, 'attachments': { 'label': 'Anhänge', 'title': 'Anhänge' }, 'search': { 'match-case': 'Streichholzschatz', 'whole-word': 'Ganze Welt', 'starts-with': 'Beginnt mit', 'ends-with': 'Endet mit', 'wildcards': 'Platzhalter', 'wildcards-'
```

```
title': 'Wildcards search. Supported wildcards are * , matching any number of characters and ? , matching a single character zero or one time.', 'proximity': 'Proximity', 'proximity-title': 'Proximity search. Use operator AROUND(n) to specify maximum count of words between search terms. Example query: apple AROUND(4) juice', 'highlight-all': 'Markieren Sie alle', 'highlight-all-title': 'Markieren Sie alle Suchübereinstimmungen', 'cancel-btn': 'Stornieren', 'start-search-btn': 'Suche', 'clear-btn': 'Klar', 'more-results-btn': 'Mehr Ergebnisse', 'search-results': 'Suchergebnisse', 'search-cancelled-msg': 'Suche auf Seite abgebrochen {{page}}', 'didnt-find-msg': 'Kann nichts finden', 'panel-title': 'Suche' }, 'shared-documents': { 'label': 'Veröffentlichte Dokumente', 'title': 'Veröffentlichte Dokumente' } }, 'dialogs': { 'form-filler': { 'empty-value': '<Leer>', 'placeholders': { 'date1': 'MM/DD/YYYY', 'time1': 'HH:mm', 'datetime': 'Datum und Uhrzeit auswählen...', 'tel': 'Telefon eingeben...', 'email': 'E-Mail eingeben...', 'url': 'URL eingeben...', 'password': 'Kennwort eingeben...', 'multiple-choice': 'Optionen auswählen...', 'choice': 'Option auswählen...', 'default': 'Text hier eingeben...' }, 'validation': { 'field-validation-failed': 'Feldüberprüfung fehlgeschlagen', 'required-field-is-empty': 'Erforderliches Feld ist leer', 'incorrect-input': 'Falsche Eingabe', 'required-number-is-incorrect': 'Erforderlicher numerischer Wert ist ungültig', 'min-failed': 'Der Mindestwert muss größer oder gleich {{min}} sein.', 'max-failed': 'Der Maximalwert muss kleiner oder gleich {{max}} sein.', 'multiple-emails-not-allowed': 'Mehrere E-Mails sind nicht zulässig', 'email-failed': 'Die E-Mail-Adresse ist falsch', 'minlength-failed': 'Die Mindestlänge des Eingabewerts muss größer oder gleich {{minlength}} sein.', 'maxlength-failed': 'Die maximale Länge des Eingabewerts muss kleiner oder gleich {{maxlength}} sein.', 'regex-pattern-failed': 'Der Wert stimmt nicht mit dem Muster des regulären Ausdrucks überein: {{pattern}}' }, 'required-field': { 'title': 'Erforderlich' }, 'empty-list-label': '<Leer>', 'alert': { 'validation-failed-confirm-apply': 'Fehler bei der Formularvalidierung. Möchten Sie die Änderungen trotzdem anwenden?', 'validation-failed': 'Fehler bei der Formularvalidierung.' }, 'heading-title': 'Formfüller', 'cancel-btn': { 'label': 'Abbrechen', 'title': 'Änderungen abbrechen' }, 'apply-btn': { 'label': 'Anwenden', 'title': 'Änderungen übernehmen' }, 'loading-label': 'Wird geladen...', 'print-progress': { 'message': 'Dokument wird zum Drucken vorbereitet...' }, 'cancel-btn': { 'title': 'Abbrechen', 'label': 'Abbrechen' }, 'documentProperties': { 'tabs': { 'description': { 'legend': 'Beschreibung', 'fields': { 'file': 'Datei', 'title': 'Titel', 'author': 'Autor', 'subject': 'Gegenstand', 'keywords': 'Schlüsselwörter', 'creationDate': 'Erstellt', 'modDate': 'Geändert', 'creator': 'Anwendung' }, 'advanced': { 'legend': 'Advanced', 'fields': { 'producer': 'PDF Producer', 'pdfFormatVersion': 'PDF-Version', 'fileSize': 'Dateigröße', 'pageSize': 'Seitengröße', 'numberOfPages': 'Anzahl der Seiten', 'fastWebView': 'Fast Web View' } }, 'title': 'Beschreibung' }, 'document-security': { 'legend': 'Dokumentensicherheit', 'access-permissions': { 'legend': 'Dokumenteinschränkungen' }, 'labels': { 'no-security': 'Keine Sicherheit', 'password-security': 'Kennwortsicherheit', 'allowed': 'Erlaubt', 'not-allowed': 'Nicht zulässig', 'printing': 'Druck:', 'content-copying': 'Kopieren von Inhalten:', 'commenting': 'Kommentierend:', 'filling-of-form-fields': 'Ausfüllen von Formularfeldern:', 'signing': 'Signieren:' }, 'description': 'Die Sicherheitsmethode des Dokuments schränkt ein, was für das Dokument getan werden kann.', 'fields': { 'security-method': 'Sicherheitsmethode:', 'encryption-level': 'Verschlüsselungsstufe:', 'document-open-password': 'Kennwort öffnen:', 'has-permissions-password': 'Berechtigungskennwort:' } }, 'used-fonts': { 'legend': 'In diesem Dokument verwendete Schriftarten', 'labels': { 'no-fonts-in-document': 'keine Schriftarten im' } } }
```

Dokument gefunden', 'subset': 'Teilmenge' }, 'properties': { 'type': 'Typ:', 'encoding': 'Codierung:', 'fallback-name': 'Fallbackname:', 'loaded-name': 'Name geladen:', 'monospace': 'Monospace', 'bold': 'Fett:', 'italic': 'Kursiv:', 'vertical': 'Vertikal:', 'embedded': 'Eingebettet' } }, 'security': { 'title': 'Sicherheit' }, 'fonts': { 'title': 'Schriftarten' } }, 'title': 'Dokumenteigenschaften', 'close-title': 'Schließen' }, 'sign-tool': { 'clear-canvas-button': { 'title': 'Leinwand löschen', 'label': 'Löschen' }, 'undo-canvas-btn': { 'title': 'Rückgängig' }, 'clear-canvas-btn': { 'title': 'Löschen' }, 'select-image-button': { 'title': 'Bild auswählen', 'label': 'Bild auswählen' }, 'clear-image-button': { 'title': 'Clear', 'label': 'Löschen' }, 'heading-title': 'Signatur hinzufügen', 'check-save-signature': { 'label': 'Signatur speichern' }, 'cancel-btn': { 'label': 'Abbrechen', 'title': 'Abbrechen' }, 'add-btn': { 'label': 'Hinzufügen', 'title': 'Signatur hinzufügen' }, 'font-name': { 'placeholder': 'Schriftartname' }, 'input-text': { 'placeholder': 'Eingabe hier...' }, 'clear-text-button': { 'title': 'Klartext', 'label': 'Löschen' }, 'font-bold': { 'title': 'Fett' }, 'font-italic': { 'title': 'Italienisch' } } }, 'messages': { 'support-api-connected-format': 'Support API-Server angeschlossen, Version: {{version}}. Bearbeitungstools aktiviert.' }, 'collaboration': { 'shared-mode-label': { 'title': 'Das Dokument steht anderen Benutzern zur Verfügung: {{usersList}}', 'text': 'Freigegebener Modus {{accessMode}}' }, 'share-dialog': { 'title': 'Zugriff verwalten', 'close-btn': { 'label': 'Schließen', 'title': 'Schließen' }, 'users-empty': 'Leer', 'loading-users': 'Wird geladen...', 'current-user-label': 'Aktueller Benutzer: {{currentUser}}', 'heading-text': 'Zugriff auf Dokument für Benutzer zulassen:', 'share-button': 'Teilen', 'users-list-heading': 'Benutzer, die Zugang zu Dokumenten haben', 'change-user-access': { 'change-to-view-only': { 'text': 'Nur Ansicht', 'title': 'Ändern Sie den Zugriffsmodus für den Benutzer {{userName}} in Nur anzeigen.' }, 'change-to-view-and-edit': { 'text': 'Anzeigen und Bearbeiten', 'title': 'Ändern Sie den Zugriffsmodus für den Benutzer {{userName}} in Ansicht und Bearbeitung.' } }, 'stop-sharing': { 'text': 'Freigabe beenden', 'title': 'Freigabe des Dokuments für Benutzer {{userName}} beenden' }, 'messages': { 'userValidation': 'Sie müssen einen Benutzernamen angeben.', 'ownerAccessModeValidation': 'Es ist nicht möglich, den Zugriffsmodus Nur anzeigen für sich selbst festzulegen. Es ist nur der Zugriffsmodus Anzeigen und Bearbeiten zulässig.' }, 'input-user-name': { 'placeholder': 'Benutzername eingeben' }, 'access-mode': { 'view-only': { 'text': 'Nur Ansicht', 'desc': 'Der Benutzer kann das Dokument nur anzeigen' }, 'edit': { 'text': 'Anzeigen und ändern', 'desc': 'Der Benutzer kann das Dokument anzeigen und ändern' }, 'denied': { 'text': 'Zugriff verweigert', 'desc': 'Benutzer kann nicht auf das Dokument zugreifen' }, 'loading': { 'text': 'Wird geladen...' }, 'unknown': { 'text': 'Unbekannt', 'desc': 'Zugriffsmodus unbekannt' } }, 'warnings': { 'securityDoesNotAllowTextCopying': 'Die Sicherheitsberechtigungen des Dokuments erlauben kein Kopieren von Text.', 'securityDoesNotAllowPrinting': 'Die Sicherheitsberechtigungen des Dokuments erlauben kein Drucken.' }, 'openSharedNoSupportApi': 'Freigegebenes Dokument kann nicht geöffnet werden. SupportApi ist nicht konfiguriert.', 'securityDoesNotAllowFillForm': 'Die Sicherheitsberechtigungen des Dokuments erlauben das Ausfüllen von Formularfeldern nicht.', 'securityDoesNotAllowEdit': 'Die Sicherheitsberechtigungen des Dokuments erlauben keine Bearbeitung des Dokuments.' }, 'about': { 'line1-support-api-enabled': 'GrapeCity PDF Viewer Version {{version}} ({{supportApiVersion}})', 'line1': 'GrapeCity PDF Viewer Version {{version}}', 'line2': 'GrapeCity PDF Viewer is only licensed for commercial use when purchased with {{anchorStart}}GrapeCity Documents for PDF{{anchorEnd}}', 'line3': 'We invite you to check out our other Document API' }}

```
Solutions:', 'list-item-1': '{{anchorStart}}Documents for Excel, .NET  
Edition{{anchorEnd}}', 'list-item-2': '{{anchorStart}}Documents for Excel, Java  
Edition{{anchorEnd}}', 'list-item-3': '{{anchorStart}}Documents for  
Word{{anchorEnd}}', 'list-item-4': '{{anchorStart}}Documents for  
Imaging{{anchorEnd}}' }, 'confirm': { 'new-document-message': 'Do you really want to  
create new document? If you dont save the document, any changes you make will be  
lost.' }, 'license': { 'invalidlicensekey': { 'message': { 'line1': 'Invalid license  
key.', 'line2': '', 'line3': 'Contact us.sales@grapecity.com to purchase a license.'  
} }, 'nolicensekey': { 'message': { 'line1': 'License Not Found', 'line2': '',  
'line3': 'You need a valid license key to run GrapeCity PDF Viewer.', 'line4':  
'Temporary keys are available for evaluation.', 'line5': 'If you purchased a license,  
your key is in your purchase confirmation email.', 'line6': 'Email  
us.sales@grapecity.com if you need assistance' }, 'watermark': { 'line1': 'Powered by  
GrapeCity PDF Viewer.', 'line2': 'You can only deploy this EVALUATION version  
locally.', 'line3': 'Temporary deployment keys are available for testing.', 'line4':  
'Email us.sales@grapecity.com.' } }, 'evallicense': { 'watermark': { 'line1':  
'Powered by GrapeCity PDF Viewer.', 'line2': 'Your temporary deployment key expires  
in {{expiresInDays}} day(s).' } }, 'evalexpiredlicense': { 'message': { 'line1':  
'Powered by GrapeCity PDF Viewer.', 'line2': '', 'line3': 'Your temporary deployment  
key has expired.', 'line4': 'Email us.sales@grapecity.com for help.' } },  
'localhostonly': { 'message': { 'line1': 'License Not Found', 'line2': '', 'line3':  
'You need a valid license key to run GrapeCity PDF Viewer.', 'line4': 'Temporary keys  
are available for evaluation.', 'line5': 'If you purchased a license, your key is in  
your purchase confirmation email.', 'line6': 'Email us.sales@grapecity.com if you  
need assistance.' } }, 'keyforanotherproduct': { 'message': { 'line1': 'This license  
key is for a different GrapeCity product.', 'line2': '', 'line3': 'Contact  
us.sales@grapecity.com to purchase a license.' } }, 'keyforanotherdomain': {  
'message': { 'line1': 'A valid license was applied. However, this license does not  
apply to this domain.', 'line2': '', 'line3': 'Contact us.sales@grapecity.com to  
purchase a new license.' } }, 'licensenotfound': { 'message': 'Lizenz nicht gefunden'  
}, 'annotation-resizer': { 'handles': { 'resize-title': 'Größe ändern', 'move-  
title': 'Move', 'end-line-dot-title': 'Endpunkt', 'start-line-dot-title':  
'Startlinienpunkt', 'middle-line-dot-title': 'Punkt der mittleren Linie' },  
'placeholders': { 'type-text-here': 'Text hier eingeben' } }, 'labels': { 'yes':  
'Ja', 'no': 'Nein', 'pageSize': { 'inch': 'in' }, 'fileSize': { 'b': 'B', 'kb': 'KB',  
'mb': 'MB', 'gb': 'GB', 'tb': 'TB' } }, 'support-api': { 'client': { 'downloading-  
file': 'Datei wird heruntergeladen {{num}} / {{count}}', 'uploading-file': 'Datei  
{{num}} wird von {{count}} hochgeladen', 'uploading-file-error': 'Fehler beim  
Hochladen der Datei.', 'uploading-file-error-format': 'Fehler beim Hochladen der  
Datei: {{error}}' } }, 'editable-text': { 'cancel-btn': { 'label': 'Abbrechen',  
'title': 'Abbrechen (ESC)' }, 'done-btn': { 'label': 'Fertig', 'title': 'Fertig  
(Strg+Eingabe)' } }  
};  
  
// Initialize localization resources:  
GcPdfViewer.i18n.init({  
    resources: { 'DE': { viewer: translation } },  
    defaultNS: 'viewer'  
});  
  
var viewer = new GcPdfViewer("#host",
```

```
{ supportApi: 'api/pdf-viewer', language: 'DE' }  
);  
viewer.addDefaultPanels();  
viewer.addAnnotationEditorPanel();  
viewer.addFormEditorPanel();
```

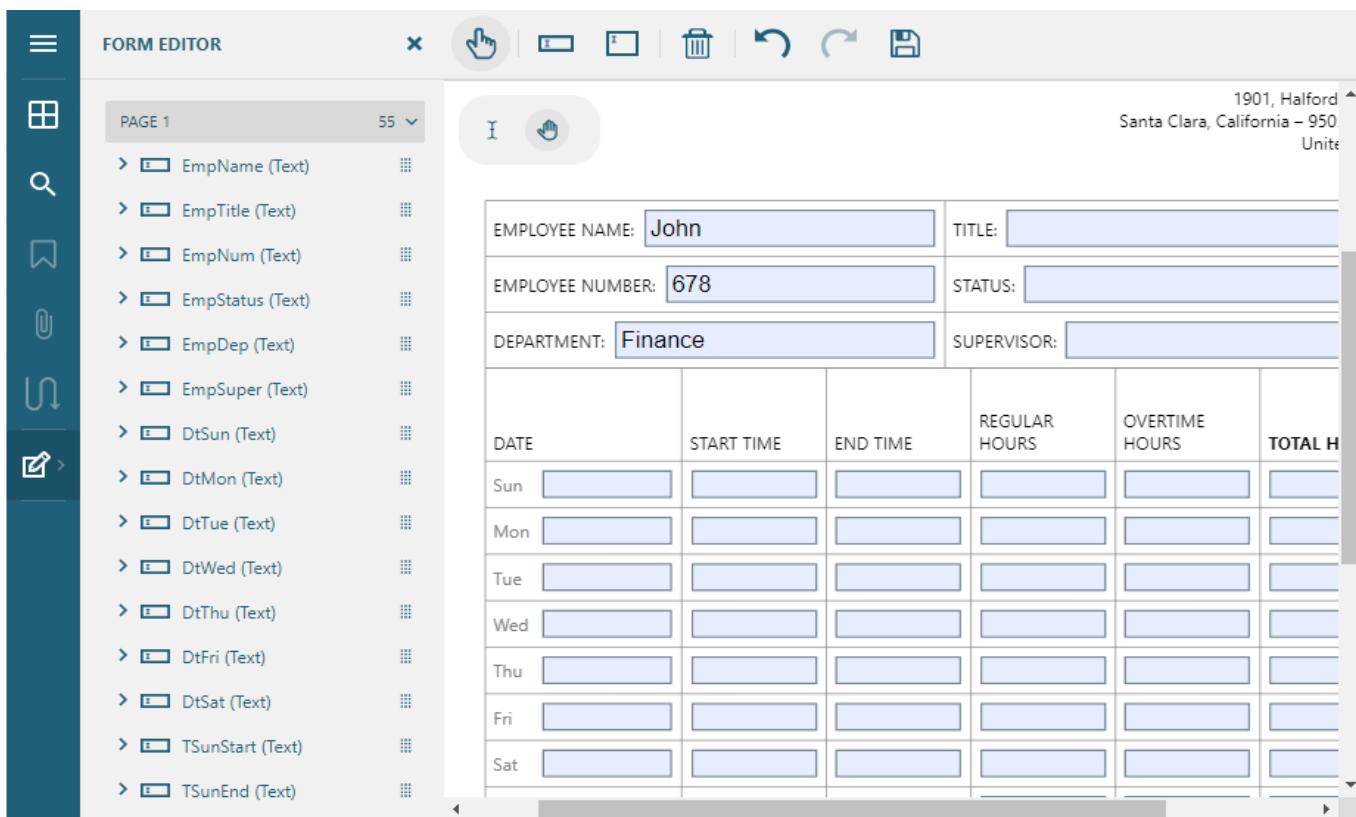


Customizations while Editing PDF Documents

When GcDocs PDF Viewer is configured to [edit PDF documents](#), it displays general editing options, Annotation and Form Editors. The editors are displayed in the left vertical panel of the Viewer whereas the editing options are displayed in the top horizontal toolbar of the Viewer.

Customize Toolbar Options

You can customize the display of editors and editing options appearing in the viewer. For eg. If you want to work with a form in PDF, you can choose to display only the form editor. Also, you can choose to view the selected form editing options (like text fields) as shown below:



Follow the steps listed in [Configure PDF Editor](#) with a modification in **Step 5** (If working on ASP.NET Webforms Application) or **Step 7** (If working on ASP.NET Core Web Application) as shown below:

To customize the toolbar to display only the Annotation editor, replace the code in <script> tag as below:

Index.cshtml

```
<script>
    var viewer = new GcPdfViewer("#host", { supportApi: 'SupportApi/GcPdfViewer' });
    viewer.addDefaultPanels();
    viewer.addAnnotationEditorPanel();
    viewer.beforeUnloadConfirmation = true;
    viewer.open("Home/GetPdf");
</script>
```

To customize the toolbar to display only the Form editor, replace the code in <script> tag as below:

Index.cshtml

```
<script>
    var viewer = new GcPdfViewer("#host", { supportApi: 'SupportApi/GcPdfViewer' });
    viewer.addDefaultPanels();
    viewer.addFormEditorPanel();
    viewer.beforeUnloadConfirmation = true;
    viewer.open("Home/GetPdf");
</script>
```

To customize the toolbar to display only the Annotation editor and text annotation tools, replace the code in <script> tag as below:

Index.cshtml

```
<script>
    var viewer = new GcPdfViewer("#host", { supportApi: 'SupportApi/GcPdfViewer' });
    viewer.addDefaultPanels();
    viewer.addAnnotationEditorPanel();
    //Customize annotation editor tools
    viewer.toolbarLayout.annotationEditor.default = ["edit-select", "$split", "edit-text", "edit-free-text", "$split", "edit-erase", "$split", "$split", "edit-undo", "edit-redo", "save"];
    viewer.beforeUnloadConfirmation = true;
    viewer.open("Home/GetPdf");
</script>
```

To customize the toolbar to display only the Form editor and text fields, replace the code in <script> tag as below:

Index.cshtml

```
<script>
    var viewer = new GcPdfViewer("#host", { supportApi: 'SupportApi/GcPdfViewer' });
    viewer.addDefaultPanels();
    viewer.addFormEditorPanel();
    //customize form editor tools
    viewer.toolbarLayout.formEditor.default = ['edit-select-field', '$split', 'edit-widget-tx-field', 'edit-widget-tx-text-area', "$split", "edit-erase-field", "$split", "edit-undo", "edit-redo", "save"];
    viewer.beforeUnloadConfirmation = true;
    viewer.open("Home/GetPdf");
</script>
```

Custom Validation of Form Fields

You can use the **validateForm** method to validate an active form in GcDocs PDF Viewer. The below example code validates a form field by using **validateForm** method and shows the relevant validation message.

Index.cshtml

```
var viewer;
function gcd(a, b) {
    if (!b) return a;
    return gcd(b, a % b);
}
function solveForm() {
    var fldValue1 = viewer.findAnnotations('fldValue1', { findField: 'fieldName' }),
        fldValue2 = viewer.findAnnotations('fldValue2', { findField: 'fieldName' }),
        fldResult = viewer.findAnnotations('fldResult', { findField: 'fieldName' });
    Promise.all([fldValue1, fldValue2, fldResult]).then(function (arr) {
        fldValue1 = arr[0][0].annotation; fldValue2 = arr[1][0].annotation;
        fldResult = arr[2][0].annotation;
        fldResult.fieldValue = gcd(fldValue1.fieldValue, fldValue2.fieldValue);
        viewer.updateAnnotation(0, fldResult);
    });
}
```

```
function validateForm() {
    var a, b, result, expectedAnswer;
    var validationResult = viewer.validateForm(function (fieldValue, field) {
        switch (field.fieldName) {
            case 'fldValue1':
                a = parseFloat(fieldValue);
                break;
            case 'fldValue2':
                b = parseFloat(fieldValue);
                break;
            case 'fldResult':
                result = parseFloat(fieldValue);
                break;
        }
        if (a && b && result) {
            expectedAnswer = gcd(a, b);
            if (parseFloat(fieldValue) !== expectedAnswer) {
                return 'Incorrect answer.';
            }
        }
        return true;
    }, true);
    if (expectedAnswer) {
        if (validationResult !== true) {
            viewer.showMessage(validationResult, 'Correct answer is ' +
expectedAnswer, 'error');
        } else {
            viewer.showMessage('Your answer is correct.', null, 'info');
        }
    } else {
        viewer.showMessage('Please input your answer.', null, 'warn');
    }
    setTimeout(function () { viewer.repaint([0]); }, 100);
}
function createPdfViewer(selector, baseOptions) {
    var options = baseOptions || {};
    if (!options.supportApi) {
        options.supportApi = {
            apiUrl: 'api/pdf-viewer',
            token: window.afToken || '',
            webSocketUrl: false
        };
    }
    viewer = new GcPdfViewer(selector, options);
    return viewer;
}
createPdfViewer("#host");
viewer.open("viewer-custom-validation.pdf?ts=1");
```

The screenshot shows a PDF document with a math problem: "What is the greatest common factor of the two values?". Below the question are two dice. There are three input fields labeled "VALUE 1" (containing "8"), "VALUE 2" (containing "12"), and "ANSWER =". At the bottom are two buttons: "Validate" (yellow) and "Solve" (green).

Note: You can lock the form fields to prevent them from editing and allow only certain users to edit them. For more information about its implementation, see [Locking Fields](#) in sample browser.

Form Filler

Form Filler feature, in GcDocs PDF Viewer, enhances your PDF form filling experience by allowing you to:

- Add input validations to custom form input types like min and max length, required field, pattern, validation messages etc.
- Add placeholder text
- Customise UI Appearance of custom form input types
- Add custom content to define any terminology, definition etc.
- Fill PDF forms conveniently on small devices due to its responsive web design

While filling PDF forms, you can use either of the following methods:

- Fill within GcDocs PDF Viewer
- Fill using the Form filler dialog (the dialog can also add validations and other properties)

The PDF form field values, when filled in Form filler dialog are reflected in the underlying PDF form. You can also save the filled form on client by using SupportApi.

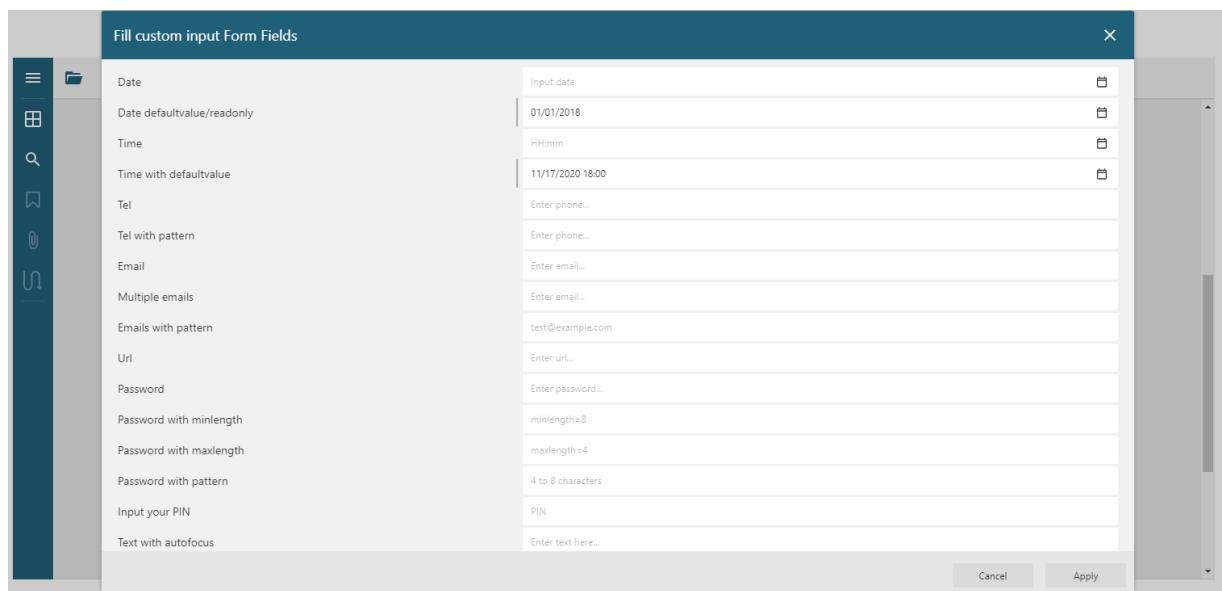
Note: SupportAPI is only needed if you want to save the filled form on client. Otherwise, load a PDF Form with custom form input types (explained below in this topic), enable the Form Filler dialog, fill the PDF Form within the Form Filler dialog, or even print a filled form on the client without using SupportApi.

Fill Forms using Form Filler Dialog

The 'Form Filler' button is provided in GcDocs PDF Viewer toolbar which is always visible but is enabled only when a PDF form is present in the Viewer.



On clicking the 'Form Filler' button, the Form filler dialog is displayed as shown in the below image.



The Form filler dialog can also be displayed by using below code:

```
Index.cshtml
if(viewer.hasForm) {
    viewer.showFormFiller();
}
```

Customize Input Form Types

You can customize the display of custom form input types and add validation options like minimum length, placeholder string, pattern, required field etc. which are displayed in the Form filler dialog. The below example elaborates the same by setting up a GcDocs PDF Viewer project as covered in [View PDF](#).

The GcDocs PDF Viewer's initialization code needs to be updated to set various formFiller options:

```
Index.cshtml
function loadPdfViewer(selector) {
    var options = {};
    options = setupFormFiller(options);
    var viewer = new GcPdfViewer(selector, options);
    viewer.addDefaultPanels();
    viewer.toolbarLayout.viewer = {
        default: ['open', 'form-filler', '$navigation', '$split', 'text-selection', 'pan',
        '$zoom', '$fullscreen', 'print', 'title', 'about'],
        mobile: ['open', 'form-filler', '$navigation', 'title', 'about'],
        fullscreen: ['$fullscreen', 'open', 'form-filler', '$navigation', '$split', 'text-
selection', 'pan', '$zoom', 'print', 'title', 'about']
    };
}
```

```
viewer.open("Commercial-Rental-Application-Form.pdf");
}

function setupFormFiller(baseOptions) {
    var options = baseOptions || {};
    // Form Filler options:
    options.formFiller = {
    };
    return options;
}
```

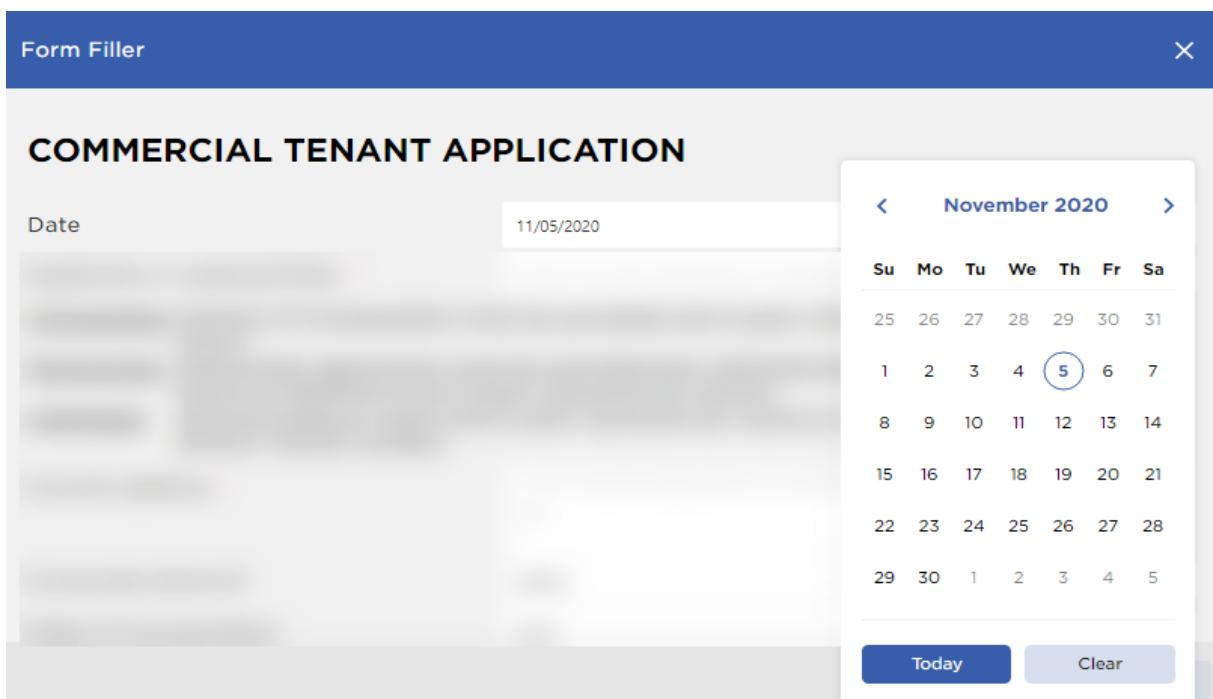
The sample PDF form used in this example is a tenant application form which seeks the details of a tenant for house rental purposes. You can also download the form [here](#).

Customize UI Appearance and Input Type Behavior

- The Date input type in PDF form is customized as:
 - The input type's display label is set to 'Date' using **displayname** property and tooltip to 'Application date' using **title** property
 - The type of input type is set to **date** to display the date picker UI for Application Date
 - The automatic focus is set on the date input type when the Form Filler dialog is opened by using the **autofocus** property
 - The default value is set to today's date by using the **defaultvalue** property

Index.cshtml

```
options.formFiller = {
    mappings: {
        'AppDate': {
            displayname: 'Date',
            title: 'Application date',
            type: 'date',
            autofocus: true,
            defaultvalue: new Date().toJSON().slice(0, 10)
        }
    }
};
```



- The placeholder text is set for 'Application or Leasing Entity' field by using the **placeholder** property.

Index.cshtml

```
'Entity': {
    title: 'Name of individual, partnership, or corporation',
    displayname: 'Applicant or Leasing Entity',
    placeholder: 'Name of individual, partnership, or corporation',
}
```

- The static placeholder is defined with custom HTML content that describes the rules for filling the 'Entity' field.

Index.cshtml

```
'CustomContent_Info1': {
    type: 'custom-content',
    content: `<table>
        <tr><td style='vertical-align:top;'>
            <i><u>Corporation:</u></i>
        </td><td style='vertical-align:top;'>
            <i>Articles of Incorporation must be provided and 2 years' Annual Report and corporate tax return.</i>
        </td></tr>
        <tr><td style='vertical-align:top;'>
            <i><u>Partnership:</u></i>
        </td><td style='vertical-align:top;'>
            <i>Partnership Agreement must be provided plus individual partners' current personal financial statement and 2 years' personal tax returns.</i>
        </td></tr>
        <tr><td style='vertical-align:top;'>
            <i><u>Individual:</u></i>
        </td><td style='vertical-align:top;'>
            <i>Personal balance sheet and 2 years' personal tax returns must be provided. Must include Drivers' license number.</i>
        </td></tr>
    </table>`,
},
```

Corporation: Articles of Incorporation must be provided and 2 years' Annual Report and corporate tax return.

Partnership: Partnership Agreement must be provided plus individual partners' current personal financial statement and 2 years' personal tax returns.

Individual: Personal balance sheet and 2 years' personal tax returns must be provided. Must include Drivers' license number.

- The Address input type is changed to multiline text area by using the **multiline** property. The **hidden** property is used to hide the second field from the fields list.

Index.cshtml

```
'Addr1': {
    title: 'Current corporate headquarters/home address (For partnership/individuals) (Do not use P.O. Box)',
    displayname: 'Current address',
    placeholder: 'Current corporate headquarters/home address (For partnership/individuals) (Do not use P.O. Box)',
    multiline: true,
    required: true,
    validationmessage: 'Address is required',
    validateoninput: true
```

```
        },
        'Addr2': {
            hidden: true,
        }
    }
```

Current address *

Current corporate headquarters/home address (For partnership/individuals) (Do not use P.O. Box)

- A full-width field is displayed without a label by using the **nolabel** property.

Index.cshtml

```
'CustomContent4': {
    type: 'custom-content',
    content: '<h4>Please list all hazardous substances that will be on the premises and the approximate amounts</h4>
    ',
    HazSub1: {
        title: 'Please list all hazardous substances that will be on the premises and the approximate amounts',
        placeholder: 'List all hazardous substances that will be on the premises and the approximate amount (line 1)',
        nolabel: true
    },
    HazSub2: {
        title: 'Please list all hazardous substances that will be on the premises and the approximate amounts',
        placeholder: 'List all hazardous substances that will be on the premises and the approximate amount (line 2)',
        nolabel: true
    },
    HazSub3: {
        title: 'Please list all hazardous substances that will be on the premises and the approximate amounts',
        placeholder: 'List all hazardous substances that will be on the premises and the approximate amount (line 3)',
        nolabel: true
    }
}
```

Please list all hazardous substances that will be on the premises and the approximate amounts

List all hazardous substances that will be on the premises and the approximate amount (line 1)

List all hazardous substances that will be on the premises and the approximate amount (line 2)

List all hazardous substances that will be on the premises and the approximate amount (line 3)

Add Validation Properties for Form Input Types

- A required input type validation is added to 'Entity' field by using the **required** property.

Index.cshtml

```
'Entity': {
    title: 'Name of individual, partnership, or corporation',
    displayname: 'Applicant or Leasing Entity',
    required: true
}
```

```

        placeholder: 'Name of individual, partnership, or corporation',
        required: true
    }
}

```

- On clicking the Apply button, all fields are validated. You can use **validationmessage** property to set a validation message and **validateoninput** property if you want the input type value to be validated immediately during user input. If the validation fails, a failed validation indicator and a validation message is displayed.

Index.cshtml

```

'CorpPhone': {
    title: 'Corporate phone number',
    displayname: 'Corporate phone #',
    placeholder: 'Corporate phone',
    type: 'tel',
    pattern: '^[+]?[()?[0-9]{3}[]?[-\s\.]?[0-9]{3}[-\s\.]?[0-9]{4,6}$',
    validationmessage: 'Valid formats: 1234567890, (123)456-7890,\n 123-456-7890, 123.456.7890, +31636363634, 075-63546725',
    validateoninput: true
},

```

Form Filler

COMMERCIAL TENANT APPLICATION

Date 11/05/2020 Entity name is required

Applicant or Leasing Entity * Name of individual, partnership, or corporation

- A pattern can be set to validate 'Corporate phone number' input type using regular expression by using the **pattern** property.

Index.cshtml

```

'CorpPhone': {
    title: 'Corporate phone number',
    displayname: 'Corporate phone #',
    placeholder: 'Corporate phone',
    type: 'tel',
    pattern: '^[+]?[()?[0-9]{3}[]?[-\s\.]?[0-9]{3}[-\s\.]?[0-9]{4,6}$',
    validationmessage: 'Valid formats: 1234567890, (123)456-7890,\n 123-456-7890, 123.456.7890, +31636363634, 075-63546725',
    validateoninput: true
},

```

Valid formats: 1234567890, (123)456-7890, 123-456...

Corporate phone #

Home phone #

Valid formats: 1234567890, (123)456-7890, 123-456-7890, 123.456.7890, +31636363634, 075-63546725

Home phone (optional)

- The minimum and maximum number of characters can be validated by defining a validation using the **minlength** and **maxlength** properties.

Index.cshtml

```
'BankAcct': {
```

```

        title: 'Bank account number',
        displayname: 'Account number',
        placeholder: 'Account number',
        minlength: 8,
        maxlength: 12,
        validationmessage: 'Expected value between 8 and 12 digits',
        validateoninput: true
    }
}

```

The screenshot shows a form with two fields. The first field is labeled "Any overnight parking?" and contains the placeholder "Expected value between 8 and 12 digits". The second field is labeled "Account number" and contains the value "12345". A red error icon is visible next to the account number field, indicating a validation error.

- A custom validation using javascript code can be provided by using **validator** property. The below example allows only 'Delaware' state as the input for 'State of incorporation' input type.

Index.cshtml

```

'CorpState': {
    title: 'State where the company was registered',
    displayname: 'State of incorporation',
    placeholder: 'State of incorporation',
    validateoninput: true,
    validator: function(fieldValue, field) {
        if(!fieldValue || !fieldValue.trim())
            return 'This field cannot be empty';
        if(fieldValue.toLowerCase().indexOf('delaware') === -1)
            return 'Only Delaware state allowed.';
        return true;
    }
},

```

The screenshot shows three fields. The first field is labeled "Home phone #". The second field is labeled "State of incorporation" and contains the value "Michigan". The third field is labeled "State of incorporation" and contains the value "delaware". A red error icon is visible next to the Michigan field, indicating a validation error. A tooltip "Only Delaware state allowed." is shown above the Michigan field.

- A common **validator** function is called for each input type in the list.

Index.cshtml

```

var viewer = new GcPdfViewer(selector, {
    renderInteractiveForms: true,
    formFiller: {
        validator: function(fieldValue, field) {
            return (fieldValue? true : 'The field cannot be empty');
        },
        mappings: {
            'fld1': {
                title: 'Application date',
                displayname: 'Date',
            },
            'fld2': {
                title: 'Name of individual, partnership, or corporation',
            }
        }
    }
});

```

```

        displayname: 'Applicant or Leasing Entity'
    }
}
}) ;

```

Form Event Handlers

Form filler feature provides various event handlers:

- **onInitialize** - It is called after the list of custom form input types is loaded and initialized but not yet rendered.
- **beforeApplyChanges** - It is called when the Apply button is clicked after the successful validation of all custom form input types.
- **beforeFieldChange** - This event is called right before the input type's value is changed.

Custom Form Input Types

Apart from the form fields supported by standard PDF specification, GcDocs PDF Viewer supports custom form input types. These are inherited from the textbox field and are listed below:

- date
- time
- tel
- number
- email
- password
- text
- url
- week
- month
- range

A PDF form with custom input form fields can be created by using [GcExcel Templates](#) or by using GcProps property in GcPdf. The form can then be filled in GcDocs PDF Viewer or Form Filler dialog with added validations and other properties.

'Date' Custom Form Input Type

The Date input type provides datepicker interface to choose dates easily. However, there might be issues with 'date' type input because of its limited browser support. Refer this [link](#) for more information.

The following table describes the properties which can be applied to a 'date' input type.

Property	Value	Description
type	"date"	Specifies the type of text box.
autofocus	"true" or "false"	Indicates whether an input type should automatically get focus when the Form filler dialog is activated or when the page loads.
autocomplete	"bday" "bday-day" "bday-month" "bday-year"	Uses the autocomplete attribute on the date input component when asking for date of birth. Refer this link for details.
defaultvalue	"2018-01-01"	Default value.
disabled	"true" or "false"	Indicates whether an input type is disabled, or not.
readonly	"true" or "false"	Indicates whether the input type is read-only, or not.
required	"true" or "false"	Indicates whether the form filling is required or not.
title	"Tooltip"	Uses the title property to specify text that most

	"First line\nSecond line"	browsers display as a tooltip.
validateoninput	"true" or "false"	True indicates whether validation should be performed immediately during user input, otherwise input validation will be performed on blur event.
validateonmessage	"The date cannot be empty."	Represents a localized message that describes the validation constraints that the control does not satisfy (if any).

The 'date' input type and its properties can also be added to a PDF form by using the following example code in GcPdf.

C#

```
// Create a new PDF document
var doc = new GcPdfDocument();
var page = doc.NewPage();
var g = page.Graphics;

TextFormat tf = new TextFormat();
tf.Font = StandardFonts.Times;
tf.FontSize = 14;
var field = new TextField();
field.Widget.Page = page;
field.Widget.Rect = new RectangleF(40, 40, 72 * 3, 24);
field.Widget.TextFormat.Font = tf.Font;
field.Widget.TextFormat.FontSize = tf.FontSize;
field.Name = "dateField1";
field.GcProps["type"] = "date";
field.GcProps["required"] = true;
field.GcProps["validationmessage"] = "The date cannot be empty.";
doc.AcroForm.Fields.Add(field);
```

'Time' Custom Form Input Type

Time input type provides time picker interface to choose time easily. Refer this [link](#) for details on browser support and additional attributes.

The following table describes the properties which can be applied to a 'time' input type.

Property	Value	Description
type	time	Specifies the type of text box.
autofocus	true	Indicates whether an input type should automatically get focus when the Form filler dialog is activated or when the page loads.
defaultvalue	18:00	Default value.
disabled	"true" or "false"	Indicates whether an input type is disabled, or not.
readonly	"true" or "false"	Indicates whether an input type is read-only, or not.
required	"true" or "false"	Indicates whether the form filling is required or not.
title	"Tooltip" "First line\nSecond line"	Uses the title property to specify text that most browsers will display as a tooltip.
validateoninput	"true" or "false"	True indicates whether validation should be performed immediately during user input, otherwise input validation will be performed on blur event.
validateonmessage	"The time"	Represents a localized message that describes the validation constraints that the control

	cannot be empty."	does not satisfy (if any).
--	-------------------	----------------------------

The 'time' input type and its properties can also be added to a PDF form by using the following example code in GcPdf.

C#

```
// Create a new PDF document
var doc = new GcPdfDocument();
var page = doc.NewPage();
var g = page.Graphics;

TextFormat tf = new TextFormat();
tf.Font = StandardFonts.Times;
tfFontSize = 14;
var field = new TextField();
field.Widget.Page = page;
field.Widget.Rect = new RectangleF(40, 40, 72 * 3, 24);
field.Widget.TextFormat.Font = tf.Font;
field.Widget.TextFormat.FontSize = tfFontSize;
field.Name = "timeField1";
field.GcProps["type"] = "time";
field.GcProps["required"] = true;
field.GcProps["validateoninput"] = true;
field.GcProps["validationmessage"] = "Please, enter time.";
doc.AcroForm.Fields.Add(field);
```

'Telephone' Custom Form Input Type

Telephone input type can be used to enter and edit a telephone number. Refer this [link](#) for details on browser support and additional attributes.

The following table describes the properties which can be applied to a 'tel' input type.

Property	Value	Description
type	"tel"	Specifies the type of text box.
autofocus	"true"	Indicates whether a input type should automatically get focus when the Form filler dialog is activated or when the page loads.
autocomplete	"on" "off" "tel" "tel-country-code" "tel-national" "tel-area-code" "tel-local" "tel-local-prefix" "tel-local-suffix" "tel-extension"	A typical implementation of autocomplete simply recalls previous values entered in the same input field.
defaultvalue	"123-456-7890"	Default value.
pattern	"^[+]?[()?[0-9]{3}[]]?[-\s\.]?[0-9]{3}[-\s\.]?[0-9]{4,6}\$" "^[0-9]{3}\$"	A regular expression the entered value must match to pass constraint validation.
placeholder	"Input phone"	Text to display inside the input type when it has no value.
disabled	"true" or "false"	Indicates whether an input type is disabled, or not.
readonly	"true" or "false"	Indicates whether an input type is read-only, or not.

required	"true" or "false"	Indicates whether the form filling is required or not.
title	"Tooltip" "First line\nSecond line"	Uses the title property to specify text that most browsers will display as a tooltip.
minlength	1	Minimum number of characters which can be considered valid.
maxlength	10	Maximum number of characters to be accepted.
validateoninput	"true" or "false"	True indicates whether validation should be performed immediately during user input, otherwise input validation will be performed on blur event.
validateonmessage	"The phone cannot be empty."	Represents a localized message that describes the validation constraints that the control does not satisfy (if any).

The 'tel' input type and its properties can also be added to a PDF form by using the following example code in GcPdf.

C#

```
// Create a new PDF document
var doc = new GcPdfDocument();
var page = doc.NewPage();
var g = page.Graphics;

TextFormat tf = new TextFormat();
tf.Font = StandardFonts.Times;
tf.FontSize = 14;
var field = new TextField();
field.Widget.Page = page;
field.Widget.Rect = new RectangleF(40, 40, 72 * 3, 24);
field.Widget.TextFormat.Font = tf.Font;
field.Widget.TextFormat.FontSize = tf.FontSize;
field.Name = "telField1";
field.GcProps["type"] = "tel";
field.GcProps["title"] = "Text area code";
field.GcProps["placeholder"] = "###";
field.GcProps["pattern"] = @"^[\d]{3}$";
field.GcProps["validateoninput"] = true;
field.GcProps["validationmessage"] = @"Valid format: 3 digits";
doc.AcroForm.Fields.Add(field);
```

'Number' Custom Form Input Type

Number input type can be used to enter a number. Refer this [link](#) for details on browser support and additional attributes.

The following table describes the properties which can be applied to a 'number' input type.

Property	Value	Description
type	"number"	Specifies the type of text box.
autofocus	"true"	Indicates whether an input type should automatically get focus when the Form filler dialog is activated or when the page loads.
defaultvalue	"123"	Default value.
placeholder	"Input number"	Text to display inside the input type when it has no value.
disabled	"true" or "false"	Indicates whether an input type is disabled, or not.
readonly	"true" or "false"	Indicates whether the input type is read-only, or not.
required	"true" or "false"	Indicates whether the form filling is required, or not.

title	"Tooltip" "First line\nSecond line"	Uses the title property to specify text that most browsers will display as a tooltip.
min	1	Minimum value to accept for this input.
max	10	Maximum value to accept for this input.
validateoninput	"true" or "false"	True indicates whether validation should be performed immediately during user input, otherwise input validation will be performed on blur event.
validateonmessage	"The number is incorrect."	Represents a localized message that describes the validation constraints that the control does not satisfy (if any).

The 'number' input type and its properties can also be added to a PDF form by using the following example code in GcPdf.

C#

```
// Create a new PDF document
var doc = new GcPdfDocument();
var page = doc.NewPage();
var g = page.Graphics;

var field = new TextField();
field.Widget.Page = page;
field.Widget.Rect = new RectangleF(40, 40, 72 * 3, 24);
field.Widget.TextFormat.Font = tf.Font;
field.Widget.TextFormat.FontSize = tf.FontSize;
field.Name = "numberField1";
field.GcProps["type"] = "number";
field.GcProps["title"] = "Enter number between 1 and 100";
field.GcProps["placeholder"] = "[1-100]";
field.GcProps["required"] = true;
field.GcProps["min"] = 1;
field.GcProps["max"] = 100;
field.GcProps["validateoninput"] = true;
field.GcProps["validationmessage"] = "The number is incorrect.";
doc.AcroForm.Fields.Add(field);
```

'Email' Custom Form Input Type

Email input type can be used to enter an e-mail address. Refer the [link](#) for details on browser support and additional attributes.

The following table describes the properties which can be applied to an 'email' input type.

Property	Value	Description
type	"email"	Specifies the type of the text box.
autofocus	"true"	Indicates whether an input type should automatically get focus when the Form filler dialog is activated or when the page loads.
autocomplete	"email"	A typical implementation of autocomplete simply recalls previous values entered in the same input field, but more complex forms of autocomplete can exist. For instance, a browser could integrate with a device's contacts list to autocomplete email addresses in an email input type.
defaultvalue	"email@example.com"	Default value.
pattern	".+@globex.com" "\S+@\S+\.\S+"	A regular expression the entered value must match to pass constraint validation.

placeholder	"Input your email"	Text to display inside the input type when it has no value.
minlength	1	Minimum number of characters which can be considered valid.
maxlength	10	Maximum number of characters to be accepted.
disabled	"true" or "false"	Indicates whether an input type is disabled, or not.
readonly	"true" or "false"	Indicates whether the input type is read-only, or not.
required	"true" or "false"	Indicates whether the form filling is required, or not.
title	"Tooltip" "First line\nSecond line"	Uses the title property to specify text that most browsers will display as a tooltip.
validateoninput	"true" or "false"	True indicates whether validation should be performed immediately during user input, otherwise input validation will be performed on blur event.
validationmessage	"The email cannot be empty."	Represents a localized message that describes the validation constraints that the control does not satisfy (if any).

The 'email' input type and its properties can also be added to a PDF form by using the following example code in GcPdf.

C#

```
// Create a new PDF document
var doc = new GcPdfDocument();
var page = doc.NewPage();
var g = page.Graphics;

TextFormat tf = new TextFormat();
tf.Font = StandardFonts.Times;
tf.FontSize = 14;
var field = new TextField();
field.Widget.Page = page;
field.Widget.Rect = new RectangleF(40, 40, 72 * 3, 24);
field.Widget.TextFormat.Font = tf.Font;
field.Widget.TextFormat.FontSize = tf.FontSize;
field.Name = "emailField1";
field.GcProps["type"] = "email";
field.GcProps["title"] = "Input your e-mail address";
field.GcProps["placeholder"] = "Input your email";
field.GcProps["pattern"] = @"^.+@globex.com";
field.GcProps["validateoninput"] = true;
field.GcProps["validationmessage"] = @"The domain must be globex.com";
doc.AcroForm.Fields.Add(field);
```

'Password' Custom Form Input Type

Password input type provides a way to securely enter a password. Refer this [link](#) for details on browser support and additional attributes.

The following table describes the properties which can be applied to a 'password' input type.

Property	Value	Description
type	"password"	Specifies the type of text box.
autofocus	"true" or "false"	Indicates whether an input type should automatically get focus when the Form filler dialog is activated or when the page loads.
autocomplete	"on" "off"	Allows the user's password manager to automatically enter the password.

	"current-password" "new-password" "one-time-code"	
defaultvalue	"anypassword!1"	Default value.
inputmode	"numeric"	If your recommended (or required) password syntax rules would benefit from an alternate text entry interface than the standard keyboard, you can use the inputmode attribute to request a specific one. The most obvious use case for this is if the password is required to be numeric (such as a PIN). Mobile devices with virtual keyboards, for example, may opt to switch to a numeric keypad layout instead of a full keyboard, to make entering the password easier.
pattern	"[a-zA-Z0-9.%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,3}\$" "^[0-9]{3,3}\$" "[0-9a-fA-F]{4,8}"	Regular expression that should match the entered value to pass constraint validation.
placeholder	"Input your password"	Text to display inside the input type when it has no value.
minlength	1	Minimum number of characters which can be considered valid.
maxlength	10	Maximum number of characters to be accepted.
disabled	"true" or "false"	Indicates whether an input type is disabled, or not.
readonly	"true" or "false"	Indicates whether an input type is read-only, or not.
required	"true" or "false"	Indicates whether the form filling is required, or not.
title	"Tooltip" "First line\nSecond line"	Uses the title property to specify text that most browsers will display as a tooltip.
validateoninput	"true" or "false"	True indicates whether validation should be performed immediately during user input, otherwise input validation will be performed on blur event.
validateonmessage	"The password cannot be empty." "Password must be 3 digits."	Represents a localized message that describes the validation constraints that the control does not satisfy (if any).

The 'password' input type and its properties can also be added to a PDF form by using the following example code in GcPdf.

```
C#
// Create a new PDF document
var doc = new GcPdfDocument();
var page = doc.NewPage();
var g = page.Graphics;

TextFormat tf = new TextFormat();
tf.Font = StandardFonts.Times;
tf.FontSize = 14;
var field = new TextField();
```

```
field.Widget.Page = page;
field.Widget.Rect = new RectangleF(40, 40, 72 * 3, 24);
field.Widget.TextFormat.Font = tf.Font;
field.Widget.TextFormat.FontSize = tf.FontSize;
field.Name = "pinField1";
field.GcProps["type"] = "password";
field.GcProps["title"] = "Input your PIN";
field.GcProps["placeholder"] = "PIN";
field.GcProps["pattern"] = @"^[\d]{3,3}$";
field.GcProps["maxlength"] = 3;
field.GcProps["validateoninput"] = true;
field.GcProps["autocomplete"] = "one-time-code";
field.GcProps["validationmessage"] = @"The PIN must be 3 digits.";
doc.AcroForm.Fields.Add(field);
```

'Text' Custom Form Input Type

Text input type can be used to create basic single-line text input type. This is the default input type. Refer this [link](#) for details on browser support and additional attributes.

The following table describes the properties which can be applied to a 'text' input type.

Property	Value	Description
type	"text"	Specifies the type of text box.
autofocus	"true" or "false"	Indicates whether an input type should automatically get focus when the Form filler dialog is activated or when the page loads.
autocomplete	"off" "name" "honorific-prefix" "given-name" "additional-name" "family-name" "honorific-suffix" "nickname" "username" "organization-title" "organization" "street-address" "address-line1" "address-line2" "address-line3" "address-level4"	Lets users specify if they need to provide assistance for automated filling of form field values, as well as guidance to the browser about what type of information is expected in the field.

	"address-level3" "address-level2" "address-level1" "country" "country-name" "cc-name" "cc-given-name" "cc-additional-name" "cc-family-name" "cc-number" "cc-exp" "cc-exp-month" "cc-exp-year" "cc-csc" "cc-type"	
defaultvalue	"any text"	Default value.
inputmode	"numeric"	If your recommended (or required) password syntax rules would benefit from an alternate text entry interface than the standard keyboard, you can use the inputmode attribute to request a specific one. The most obvious use case for this is if the password is required to be numeric (such as a PIN). Mobile devices with virtual keyboards, for example, may opt to switch to a numeric keypad layout instead of a full keyboard, to make entering the password easier.
pattern	"^ [0-9]{3,3}\$" "[0-9a-fA-F]{4,8}"	A regular expression the entered value must match to pass constraint validation.
placeholder	"Input here "	Text to display inside the input type when it has no value.
multiline	"true" or "false"	Set this property to true to use the text area as a user input element.
minlength	1	Minimum number of characters which can be considered valid.
maxlength	10	Maximum number of characters to be accepted.
disabled	"true" or "false"	Indicates whether an input type is disabled, or not.
readonly	"true" or "false"	Indicates whether an input type is read-only, or not.
required	"true" or "false"	Indicates whether form filling is required, or not.
spellcheck	"true" or	The spellcheck property is an enumerated attribute that defines whether the element may

	"false"	be checked for spelling errors.
title	"Tooltip" "First line\nSecond line"	Uses the title property to specify text that most browsers will display as a tooltip.
validateoninput	"true" or "false"	True indicates whether validation should be performed immediately during user input, otherwise input validation will be performed on blur event.
validateonmessage	"The password cannot be empty." "Password must be 3 digits."	Represents a localized message that describes the validation constraints that the control does not satisfy (if any).

The 'text' input type and its properties can also be added to a PDF form by using the following example code in GcPdf.

C#

```
// Create a new PDF document
var doc = new GcPdfDocument();
var page = doc.NewPage();
var g = page.Graphics;

TextFormat tf = new TextFormat();
tf.Font = StandardFonts.Times;
tf.FontSize = 14;
var field = new TextField();
field.Widget.Page = page;
field.Widget.Rect = new RectangleF(40, 40, 72 * 3, 24);
field.Widget.TextFormat.Font = tf.Font;
field.Widget.TextFormat.FontSize = tf.FontSize;
field.Name = "textField1";
field.GcProps["type"] = "text";
field.GcProps["title"] = "Input text";
field.GcProps["placeholder"] = "Input here";
field.GcProps["maxlength"] = 10;
field.GcProps["validateoninput"] = true;
field.GcProps["validationmessage"] = "The value is incorrect, maximum length is 10.";
doc.AcroForm.Fields.Add(field);
```

GcPdf Examples for Other Input Types

The 'month' input type and its properties can also be added to a PDF form by using the following example code in GcPdf.

C#

```
// Create a new PDF document
var doc = new GcPdfDocument();
var page = doc.NewPage();
var g = page.Graphics;

TextFormat tf = new TextFormat();
tf.Font = StandardFonts.Times;
tf.FontSize = 14;
var field = new TextField();
```

```

field.Widget.Page = page;
field.Widget.Rect = new RectangleF(40, 40, 72 * 3, 24);
field.Widget.TextFormat.Font = tf.Font;
field.Widget.TextFormat.FontSize = tf.FontSize;
field.Name = "monthField1";
field.GcProps["type"] = "month";
field.GcProps["title"] = "Enter month.";
field.GcProps["placeholder"] = "month";
field.GcProps["required"] = true;
field.GcProps["validationmessage"] = "The month value cannot be empty.";
doc.AcroForm.Fields.Add(field);

```

The 'range' input type and its properties can also be added to a PDF form by using the following example code in GcPdf.

C#

```

// Create a new PDF document
var doc = new GcPdfDocument();
var page = doc.NewPage();
var g = page.Graphics;

TextFormat tf = new TextFormat();
tf.Font = StandardFonts.Times;
tf.FontSize = 14;
var field = new TextField();
field.Widget.Page = page;
field.Widget.Rect = new RectangleF(40, 40, 72 * 3, 24);
field.Widget.TextFormat.Font = tf.Font;
field.Widget.TextFormat.FontSize = tf.FontSize;
field.Name = "rangeField1";
field.GcProps["type"] = "range";
field.GcProps["title"] = " Please select a number between 1 and 1000.";
field.GcProps["min"] = 1;
field.GcProps["max"] = 1000;
doc.AcroForm.Fields.Add(field);

```

Properties Supported by Form Input Types

The following table provides consolidated information about properties supported by different input types.

Attribute	Input Type	Description
autocomplete	All	Input type.
autofocus	All	Automatically focus the form control when the page is loaded.
defaultvalue	All	The default value.
disabled	All	Whether the form control is disabled.
displayname	All	Text label for the input control. Applicable only if the input type appears in the Form Filler dialog box.
min	number and date	Minimum value to accept for the input.
max	number and date	Maximum value to accept for the input.
maxlength	password, search, tel, text and url	Maximum length (number of characters) of value.
minlength	password, search, tel, text and url	Minimum length (number of characters) of value

multiline	text	Set this property to true if you want to use the textarea as a user input element.
multiple	email	Boolean. Whether to allow multiple values or not.
pattern	password, text and tel	Pattern value must match to be valid.
placeholder	password, search, tel, text and url	Text that appears in the form control when it has no value set.
readonly	All	Boolean. The value is not editable.
required	All	Boolean. A value is required or must be check for the form to be submittable.
spellcheck	search and text	Whether the element may be checked for spelling errors.
type	All	Type of form control.
validateonmessage	All	Localized validation message.
validateoninput	All	Indicates whether validation should be performed immediately during user input.

To know more about how to create a PDF form by using custom input form types in GcPdf, refer [Fill Custom Form Input Types](#).

Fill Custom Form Input Types

GcDocs PDF Viewer supports custom form input types which are not supported by standard PDF specification. Hence these input types can only be added to PDF forms via [GcExcel Templates](#) or GcPdf. The forms with custom form input types can only be opened and viewed in GcDocs Pdf Viewer (not in Acrobat or other PDF viewers). These custom input types are very common and useful which allow you to fill PDF forms conveniently. These are listed as below:

- date
- time
- tel
- number
- email
- password
- text
- url
- week
- month
- range

Create PDF form using Custom Form Input Types

You can create a PDF form with custom form input types by using GcPdf's GcProps API where the validations and properties are added to input types in the form itself. The form can be filled in the Viewer or Form filler dialog and the validations and properties are displayed in both places.

The below mentioned code shows how to define custom form input types and their properties by using key value pairs.

C#

```
private Dictionary<string, KeyValuePair<string, object>[]> SampleGcPropTextFields =
new Dictionary<string, KeyValuePair<string, object>[]>()
{
    {
        "date", new KeyValuePair<string, object>[] {
            new KeyValuePair<string, object>("type", "date"),
            new KeyValuePair<string, object>("placeholder", "Enter Date"),
            new KeyValuePair<string, object>("required", true),
            new KeyValuePair<string, object>("minDate", "1900-01-01"),
            new KeyValuePair<string, object>("maxDate", "2050-12-31"),
            new KeyValuePair<string, object>("format", "MM/DD/YYYY"),
            new KeyValuePair<string, object>("value", "2022-01-01")
        }
    }
}
```

```
        new KeyValuePair<string, object>("placeholder", "Input date"))
    },
    {
        "date defaultvalue/readonly", new KeyValuePair<string, object>[] {
            new KeyValuePair<string, object>("type", "date"),
            new KeyValuePair<string, object>("defaultvalue", "2018-01-01"),
            new KeyValuePair<string, object>("readonly", true)
        }
    },
    {
        "time", new KeyValuePair<string, object>[] {
            new KeyValuePair<string, object>("type", "time")
        }
    },
    {
        "time with defaultvalue", new KeyValuePair<string, object>[] {
            new KeyValuePair<string, object>("type", "time"),
            new KeyValuePair<string, object>("defaultvalue", "18:00")
        }
    },
    {
        "tel", new KeyValuePair<string, object>[] {
            new KeyValuePair<string, object>("type", "tel"),
            new KeyValuePair<string, object>("validateoninput", true)
        }
    },
    {
        "tel with pattern", new KeyValuePair<string, object>[] {
            new KeyValuePair<string, object>("type", "tel"),
            new KeyValuePair<string, object>("pattern", @"^[\+]?[()?[0-9]{3}[]]?[-\s\.\.][0-9]{3}[-\s\.\.][0-9]{4,6}$"),
            new KeyValuePair<string, object>("validateoninput", true),
            new KeyValuePair<string, object>("validationmessage", @"Valid
formats:
(123) 456-7890
(123)456-7890
123-456-7890
123.456.7890
1234567890
+31636363634
075-63546725" )
        }
    },
    {
        "email", new KeyValuePair<string, object>[] {
            new KeyValuePair<string, object>("type", "email"),
            new KeyValuePair<string, object>("autocomplete", "email"),
            new KeyValuePair<string, object>("validateoninput", true)
        }
    },
    {
        "multiple emails", new KeyValuePair<string, object>[] {
            new KeyValuePair<string, object>("type", "email"),
            new KeyValuePair<string, object>("autocomplete", "email"),
            new KeyValuePair<string, object>("validateoninput", true),
            new KeyValuePair<string, object>("multiple", true)
        }
    }
```

```
},
{
    "emails with pattern", new KeyValuePair<string, object>[] {
        new KeyValuePair<string, object>("type", "email"),
        new KeyValuePair<string, object>("pattern", @"\S+@example\.com"),
        new KeyValuePair<string, object>("autocomplete", "email"),
        new KeyValuePair<string, object>("placeholder", "test@example.com"),
        new KeyValuePair<string, object>("validationmessage", "Expected
domain @example.com."),
        new KeyValuePair<string, object>("validateoninput", true),
        new KeyValuePair<string, object>("multiple", true) }
},
{
    "url", new KeyValuePair<string, object>[] {
        new KeyValuePair<string, object>("type", "url"),
        new KeyValuePair<string, object>("autocomplete", "url"),
        new KeyValuePair<string, object>("validateoninput", true) }
},
{
    "password", new KeyValuePair<string, object>[] {
        new KeyValuePair<string, object>("type", "password"),
        new KeyValuePair<string, object>("autocomplete", "new-password"),
        new KeyValuePair<string, object>("validateoninput", true) }
},
{
    "password with minlength", new KeyValuePair<string, object>[] {
        new KeyValuePair<string, object>("type", "password"),
        new KeyValuePair<string, object>("placeholder", "minlength=8"),
        new KeyValuePair<string, object>("minlength", 8),
        new KeyValuePair<string, object>("validateoninput", true),
        new KeyValuePair<string, object>("autocomplete", "new-password") } },
{
    "password with maxlength", new KeyValuePair<string, object>[] {
        new KeyValuePair<string, object>("type", "password"),
        new KeyValuePair<string, object>("placeholder", "maxlength=4"),
        new KeyValuePair<string, object>("maxlength", 4),
        new KeyValuePair<string, object>("validateoninput", true),
        new KeyValuePair<string, object>("autocomplete", "off") } },
{
    "password with pattern", new KeyValuePair<string, object>[] {
        new KeyValuePair<string, object>("type", "password"),
        new KeyValuePair<string, object>("placeholder", "4 to 8 characters"),
        new KeyValuePair<string, object>("pattern", @"^(?=.*\d).{4,8}$"),
        new KeyValuePair<string, object>("validateoninput", true),
        new KeyValuePair<string, object>("validationmessage", "The password
must be between 4 and 8 characters."),
        new KeyValuePair<string, object>("autocomplete", "off") }
},
{
    "password PIN", new KeyValuePair<string, object>[] {
        new KeyValuePair<string, object>("type", "password"),
```

```
        new KeyValuePair<string, object>("title", "Input your PIN"),
        new KeyValuePair<string, object>("placeholder", "PIN"),
        new KeyValuePair<string, object>("pattern", @"^+[0-9]{3,3}$"),
        new KeyValuePair<string, object>("maxlength", 3),
        new KeyValuePair<string, object>("validateoninput", true),
        new KeyValuePair<string, object>("validationmessage", "PIN password
must be 3 digits."),
        new KeyValuePair<string, object>("autocomplete", "one-time-code") }
    },
    {
        "text with autofocus", new KeyValuePair<string, object>[] {
        new KeyValuePair<string, object>("type", "text"),
        new KeyValuePair<string, object>("autofocus", "true") }
    },
    {
        "text with required", new KeyValuePair<string, object>[] {
        new KeyValuePair<string, object>("type", "text"),
        new KeyValuePair<string, object>("required", true),
        new KeyValuePair<string, object>("validateoninput", true) }
    },
    {
        "text, spellcheck='true'", new KeyValuePair<string, object>[] {
        new KeyValuePair<string, object>("type", "text"),
        new KeyValuePair<string, object>("defaultvalue", "mistaake"),
        new KeyValuePair<string, object>("spellcheck", "true") }
    },
    {
        "text, spellcheck='false'", new KeyValuePair<string, object>[] {
        new KeyValuePair<string, object>("type", "text"),
        new KeyValuePair<string, object>("defaultvalue", "mistaake"),
        new KeyValuePair<string, object>("spellcheck", "false") }
    },
    {
        "month", new KeyValuePair<string, object>[] {
        new KeyValuePair<string, object>("type", "month") }
    },
    {
        "week", new KeyValuePair<string, object>[] {
        new KeyValuePair<string, object>("type", "week") }
    },
    {
        "number with min/max", new KeyValuePair<string, object>[] {
        new KeyValuePair<string, object>("type", "number"),
        new KeyValuePair<string, object>("placeholder", "number"),
        new KeyValuePair<string, object>("required", true),
        new KeyValuePair<string, object>("title", "Please enter a number
between 1 and 100"),
        new KeyValuePair<string, object>("min", 1),
        new KeyValuePair<string, object>("max", 100),
        new KeyValuePair<string, object>("validateoninput", true),
        new KeyValuePair<string, object>("validationmessage", "Expected
```

```
number from 1 to 100") }  
    },  
    {  
        "search", new KeyValuePair<string, object>[] { new  
KeyValuePair<string, object>("type", "search") }  
    },  
    {  
        "range", new KeyValuePair<string, object>[] { new  
KeyValuePair<string, object>("type", "range"),  
            new KeyValuePair<string, object>("title", "Please select a number  
between 1 and 100"),  
            new KeyValuePair<string, object>("defaultvalue", 50),  
            new KeyValuePair<string, object>("min", 1),  
            new KeyValuePair<string, object>("max", 100) }  
    },  
    {  
        "color", new KeyValuePair<string, object>[] { new  
KeyValuePair<string, object>("type", "color"),  
            new KeyValuePair<string, object>("title", "Please select a  
color"),  
            new KeyValuePair<string, object>("defaultvalue", 50),  
            new KeyValuePair<string, object>("min", 1),  
            new KeyValuePair<string, object>("max", 100) }  
    }  
};
```

The below mentioned code adds the custom form input types defined above to a PDF form.

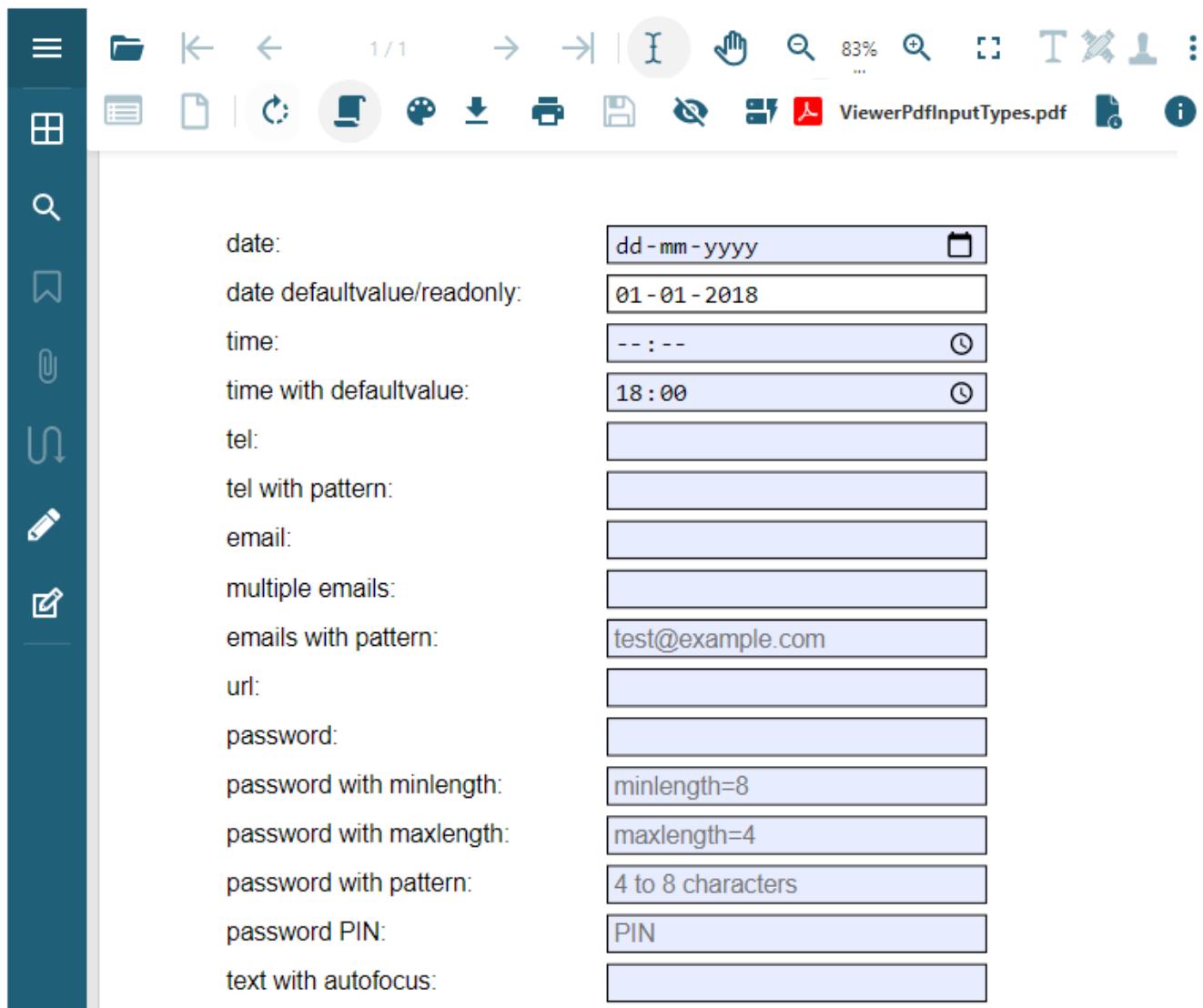
```
C#  
  
public void CreateFormFieldsPDF(Stream stream)  
{  
  
    var doc = new GcPdfDocument();  
    var page = doc.NewPage();  
    var g = page.Graphics;  
    TextFormat tf = new TextFormat();  
    tf.Font = StandardFonts.Helvetica;  
    tf.FontSize = 14;  
  
    PointF ip = new PointF(72, 72);  
    float fldOffset = 72f * 3f;  
    float fldHeight = tf.FontSize * 1.6f;  
    float dY = 28;  
  
    // Add sample fields:  
    int orderindex = 1;  
    foreach(var data in SampleGcPropTextFields)  
    {  
        string fieldName = data.Key;
```

```
g.DrawString(${fieldName}:", tf, ip);

TextField field = new TextField();
// Fill GcProps dictionary:
KeyValuePair<string, object>[] gcProps = data.Value;
foreach(var gcProp in gcProps)
{
    field.GcProps[gcProp.Key] = gcProp.Value;
}
field.GcProps["orderindex"] = orderindex;
field.Name = fieldName;
field.Widget.Page = page;
field.Widget.Rect = new RectangleF(ip.X + fldOffset, ip.Y, 72 * 3,
fldHeight);
field.Widget.TextFormat.Font = tf.Font;
field.Widget.TextFormat.FontSize = tf.FontSize;
doc.AcroForm.Fields.Add(field);
ip.Y += dY;
orderindex++;
}

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

The PDF form created by using the above code will look like below:



 **Note:** You can also create a PDF form by using custom form input types in GcExcel. For more information, refer [Custom Form Input Types](#).

Client API Reference

Please refer [GcDocs PDF Viewer API](#) for all the assemblies required to create applications using GcDocs PDF Viewer.

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	
	Quick Start	Tutorial - Create PDF documents on Windows, MAC, Linux in .NET Core.	
	Hello, World!	Create PDF documents on Windows, MAC, Linux in .NET Core.	
Document	Document Properties	Set document properties of PDF documents in .NET Core.	
	Linearized PDF	Creating linearized PDF file in .NET Core.	
	Page Size/Orientation	Set page size and orientation of PDF Documents in .NET Core.	
	PDF/A	Creating PDF/A-3u conformant document in .NET Core.	
	Large Document	StartDoc/EndDoc	Create large PDF document using less memory in .NET Core.
	Large Document	Large Document	Create large PDF Document in simple way in .NET Core.
	Large TextLayout	Large TextLayout	Creating a large PDF using a single TextLayout method in .NET Core.
Fonts	Font Collection	Create PDF documents with different fonts in .NET Core, using FontCollection method.	
	Font from File	Loading font loaded from a font file in PDF document in .NET Core.	
	Standard PDF Fonts	Create PDF Documents with Standard PDF Fonts in .NET Core.	
	Fallback Fonts	Create PDF Documents using fallback fonts in .NET Core.	
	Bold/Italic Emulation	Control of bold and/or italic emulation with normal fonts in PDF Documents in .NET Core.	
	EUDC Fonts	Rendering private use Unicode characters (PUA) with custom EUDC fonts (.tte) in PDF Documents in .NET Core.	
	Font Features	Create PDF Documents with Font features in .NET Core.	
	Surrogates etc.	Rendering surrogate pairs and Unicode characters in PDF Documents in .NET Core.	
	Surrogates Portable	Rendering surrogate pairs in platform-independent way in PDF documents in .NET Core.	

Unicode Ranges		Create PDF Documents with multiple Unicode ranges in .NET Core.
Text	Text Rendering	Render text using DrawString/DrawTextLayout methods in PDF Documents in .NET Core.
	Character Formatting	Create PDF Documents with character formatting in .NET Core.
	Paginated Text	Create PDF Documents with long text spanning multiple pages in .NET Core.
	Text Trimming	Create PDF Documents with trimmed text and ellipsis at end of string in .NET Core.
	Multi-formatted Text	Create PDF Documents with multiple formatting in a single paragraph in .NET Core.
	Word and Char Wrap	Create PDF Documents using Word wrap and Character wrap when rendering text in .NET Core.
	Sub and Super Script	Render text as subscript or superscript in PDF Documents in .NET Core.
	Multiple Languages	Create PDF Documents in multiple languages in .NET Core.
	Paragraph Alignment	Align paragraphs in PDF Documents in .NET Core.
	Paragraph Formatting	Add paragraph formatting in PDF documents in .NET Core.
	Text Alignment	Aligning and justifying paragraph text in PDF Documents in .NET Core.
	Numbered List	Render a numbered list in PDF Documents in .NET Core.
	Nested Lists	Add styles to numbered lists in PDF Documents in .NET Core.
	Keep With Next	Preventing page breaks between paragraphs in PDF Documents in .NET Core.
Advanced	Balanced Columns	Create multi-column page layout with balanced columns and document outline in PDF Documents in .NET Core.
	Multi-column Text	Create magazine styled multi-column page layout in PDF Documents in .NET Core.
	Tate Chu Yoko	Rendering horizontal runs in a vertical text (tate chu yoko) in PDF Documents in .NET Core.
	Outlined Text	Outline and fill text using brushes in PDF documents in .NET Core.
	Ligatures	Create PDF Documents with typographics ligatures in .NET Core.
	Drop Caps	Add text with drop cap in PDF Documents in .NET Core.
	Text Around Images	Rendering text around images in PDF Documents in .NET Core.
	Inline Images	Add text with inline images in PDF documents .NET Core.
	Arabic Columns	Rendering Arabic text using a columnar layout in PDF Documents in .NET Core.
	Japanese Columns	Rendering vertical Japanese text using a layout with horizontal columns in PDF Documents in .NET Core.

	Rotated Text	Rotating text by an arbitrary angle in PDF Documents in .NET Core.
	Rotated Text (alt)	Rotating text by an arbitrary angle using matrix multiplication in PDF Documents in .NET Core.
	Tabs Alignment	Render aligned columns of numbers in PDF Documents in .NET Core.
	Vertical Text	Render vertical text in LTR and RTL modes in PDF Documents in .NET Core.
Images	Slide Pages	Generate pages of slides from all images in PDF Documents in .NET Core.
	Raw Images	Create PDF documents with images and image alignment in .NET Core.
	Get Images	Extract images from PDF Documents in .NET Core.
	Image Transparency	Rendering images with transparency-opacity in PDF Documents in .NET Core.
Graphics	Gradients	Using linear and radial gradient brushes in PDF Documents in .NET Core.
	Round Rectangles	Render round rectangles in PDF Documents in .NET Core.
	Shapes	Add graphics in PDF Documents in .NET Core.
	Transformations	Create PDF Documents with graphics transformations in .NET Core.
	Soft Mask	Use soft mask for drawing semi-transparent content and clipped graphics in PDF Documents in .NET Core.
Navigation	Link To URL	Adding text with URL link in PDF Documents in .NET Core.
	Outlines	Adding bookmarks to a PDF Document in .NET Core.
	Destinations	Jump to link within PDF Document in .NET Core.
	Image Links	Add Images with destination links to PDF Documents in .NET Core.
	Page Labels	Add Page Labels to a PDF Document in .NET Core.
	Image Article Threads	Create article threads over related images in a PDF document in .NET Core.
Annotations	Annotations	Add annotations to PDF Documents in .NET Core.
	File Attachments	Attach files to a PDF document in .NET Core.
	Stamp Annotation	Add stamp annotation with a custom image in PDF Document in .NET Core.
	Sound Annotations	Add sound annotations to PDF Document in .NET Core.
	Ink Annotations	Add Ink Annotation in PDF Document in .NET Core.
	Find and Redact	Find and redact text in a PDF Document in .NET Core.
Forms	AcroForm Fields	Creating AcroForm fields in PDF Document in .NET Core.
	Submit Form	Submit PDF Form data to a server in .NET Core.
	Import XML	Import submitted form data from XML in PDF Document in .NET Core.

	Import Form Data	Import form data submitted from the client in PDF Document in .Net Core.
Tags	Tagged Paragraphs	Create tagged PDF documents with tagged paragraphs in .NET Core.
	Tagged TextLayout	Create tagged PDF documents with long text spanning multiple pages in .NET Core.
Merge PDFs	Merge PDFs	Merge PDFs in a single PDF document in .NET Core.
	Page FormXObject	Draw PDF Pages on another PDF Document in .NET Core.
Editing	Load PDF	Load and modify an existing PDF file in .NET Core.
	Fill PDF Form	Load and fill PDF Form in .NET Core.
	List Fonts	Enumerate fonts found in a loaded PDF in .NET Core.
	Extract Text	Extract text from a PDF file in .NET Core.
	Find Text	Find text and highlight all occurrences in a PDF File .NET Core.
	Find Transformed	Find and highlight transformed text in a PDF File in .NET Core.
Miscellaneous	Signing PDF	Create and digitally sign PDF Documents with .pfx file.
	Incremental Update	Sign a signed PDF file in .NET Core.
	Visual Signature	Sign a PDF file using a visual signature with a .pfx file.
	Document Attachments	Attach files to a PDF document in .NET Core.
	Easy Page Headers	Add page headers and footers to a PDF Document in .NET Core.
	JavaScript Action	Add JavaScript action in PDF Document in .NET Core.
	Security Handlers	Encrypt/decrypt PDF Files in .NET Core.
	CMAP Resources	Extract text encoded with CMaps in PDF File in .NET Core.
	Remove Signature Fields	Find and remove existing signature fields from PDF Document in .NET Core.
	Remove Signatures	Find and remove digital signature from PDF Document in .NET Core.
	Save as Image	Save PDF pages as images in .NET Core.

HTML Samples

Sample	Description
Hello World (HTML)	Render an HTML string to PDF files in .NET Core.
Google Home Page	Render the Google home page to PDF files with default options in .Net Core.
HTML Settings	Render a Web page to PDF with options (orientation, margins, headers, footers).
Simple HTML table	Render a simple table using HTML.
Dynamic Table	Render a multi-page HTML table.
Product Price List	Render a data table from an XML data set using an HTML template.
Merge Table Rows	Build a table using JavaScript to merge cells with same values.

Barcode Samples

Supported Barcodes	Add barcodes to PDF files in .NET Core.
Shipping Labels	Create shipping labels with barcodes in PDF document in .NET Core.

Fancy Doc Samples

Sample	Description
Goods Return Form	Creating complex layout Acroform in PDF document.
Time Sheet Form	Create a Time Sheet PDF Form in .NET Core.
Data Sheet	Create PDF data sheet with a complex layout in .NET Core.
Wetlands	Create a text document in PDF with images and text highlights in .NET Core.

Use Case Samples

Sample	Description
Time Sheet	Create a Time Sheet PDF template, fill and protect it in .NET Core.
Time Sheet Incremental	Sign TimeSheet PDF Form using two signatures in .NET Core.
Word Index	Generate and append alphabetical index to a PDF file in .NET Core.

Walkthrough

Follow the walkthrough, which is a step-by-step tutorial, to understand more complex tasks that can be accomplished using GcPdf.

- [Convert HTML to PDF Report](#)

Convert HTML to PDF Report

The following walkthrough takes you through a step by step process on how to render a PDF report using [Render HTML to PDF](#) feature.

Consider a scenario where a PDF report is to be generated regarding the products available in a supermarket. These products have been grouped into different categories like beverages, condiments, dairy products, seafood etc. The unit price and stock of each product as well as the whole category is also maintained in separate columns.

The PDF report is generated by using a sample database for the data of products and an HTML template file which outlines the UI of the final PDF report.

Suppose we have the following pre-requisites available using which we will render a PDF report:

- Data source to provide data for the report. Here, we are using 'GcNWInd.xml' which contains the schema and required data.
- HTML template file which defines the layout of the final report to be created. Here, we are using 'ProductListTemplate.html' which uses {{Mustache}} syntax to specify data bound fields.
- Stubble.core package which is available on [Nuget](#) and is used to bind data to template.

HTML template file

```
ProductListTemplate.html

<!DOCTYPE html>
<html>
<head>
    <style>
        html * {
            font-family: 'Trebuchet MS', Arial, Helvetica, sans-serif !important;
        }

        h1 {
            color: #1a5276;
            background-color: #d2b4de;
            text-align: center;
            padding: 6px;
        }

        thead {
            display: table-header-group;
        }

        #products {
            font-family: 'Trebuchet MS', Arial, Helvetica, sans-serif;
```

```
        border-collapse: collapse;
        width: 100%;
    }

    #products td, #products th {
        border: 1px solid #ddd;
        padding: 8px;
    }

    #products tr:nth-child(even) {
        background-color: #f2f2f2;
    }

    #products tr:hover {
        background-color: #ddd;
    }

    #products th {
        padding-top: 12px;
        padding-bottom: 12px;
        text-align: left;
        background-color: #a569bd;
        color: white;
    }
    span{
        float:right;
    }
</style>
</head>

<body>

<h1>Products Stock Report</h1>
<table id='products'>
    <thead>
        <tr>
            <th>Description</th>
            <th>Unit Price</th>
            <th>Quantity Per Unit</th>
        </tr>
    </thead>
    {{#Query}}
    <tr>
        <th colspan="3">{{CategoryName}}      <span align='right'>Total Units in
Stock: {{CategoryStockTotal}}</span></th>
    </tr>
    {{#ProductList}}
    <tr>
        <td>{{ProductName}}</td>
        <td align='right'>{{UnitPrice}}</td>
        <td>{{QuantityPerUnit}}</td>
    </tr>
    {{/ProductList}}
</table>
</body>
```

```
<td align='right'>{{UnitsInStock}}</td>

</tr>
{{/ProductList}}
{{/Query}}
</table>
</body>
</html>
```

The below image shows the output of HTML template file:

Products Stock Report		
Description	Unit Price	Quantity Per Unit
{{CategoryName}}		Total Units in Stock: {{CategoryStockTotal}}
{{ProductName}}	{{UnitPrice}}	{{UnitsInStock}}

Create a console application

1. Create a .NET Core console application in Visual Studio and install 'GrapeCity.Documents.Pdf' package from nuget by following the steps mentioned in '[Getting Started](#)'.
2. Create a class file with name 'ProductListTemplate' and define a method 'CreatePdf' which would be used to convert the HTML to PDF report.

Fetch data from XML

1. Load the datasource XML file "GcNWInd.xml" using **DataSet** class and read its XML schema using **ReadXML** method of DataSet class.
2. Fetch the DataTable "CategoriesAndProducts" from the DataSet using **DataTable** class. Cast the XML data of datasource into a list and use a LINQ query to create grouped data from the data table.

```
C#
using System;
using System.IO;
using System.Drawing;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Reflection;
using GrapeCity.Documents.Pdf;
using GrapeCity.Documents.Html;
using Newtonsoft.Json;
using System.Text.RegularExpressions;
```

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

        using (var ds = new DataSet())
        {
            //Fetch data
            ds.ReadXml(Path.Combine("Resources", "data", "GcNWind.xml"));

            DataTable dtCPts = ds.Tables["CategoriesAndProducts"];
            var categoryProducts =
                from prod in dtCPts.Select()
                group prod by prod["CategoryName"] into newGroup
                select newGroup;

            //Create grouped data, to display the product list based on
            Category groups
            //and calculate the total units in stock for each category
            List<object> lstCategory = new List<object>();
            foreach (var nameGroup in categoryProducts)
            {
                List<object> lstProduct = new List<object>();
                int TotalCategoryStock = 0;
                foreach (var pro in nameGroup)
                {
                    lstProduct.Add(new
                    {
                        ProductName = pro["ProductName"].ToString(),
                        UnitsInStock = pro["UnitsInStock"].ToString(),
                        UnitPrice = pro["UnitPrice"].ToString()
                    });
                    TotalCategoryStock +=
                    Convert.ToInt32(pro["UnitsInStock"]);
                }

                lstCategory.Add(new
                {
                    CategoryName = nameGroup.Key.ToString(),
                    CategoryStockTotal = TotalCategoryStock,
                    ProductList = lstProduct
                });
            }
            var dataBound = new { Query = lstCategory };
        }
    }
}
```

Bind HTML template with data

1. Load the HTML template 'ProductListTemplate.html' using **ReadAllText** method of **File** class which reads the template file.
2. Bind the template to datasource using the **Render** method of StubbleBuilder class (from the **StubbleBuilder** library).

C#

```
var template = File.ReadAllText(Path.Combine("Resources", "Misc",
"ProductListTemplate.html"));

//Bind the template to data
var builder = new Stubble.Core.BUILDERS.StubbleBuilder();
var boundTemplate = builder.Build().Render(template, dataBound);
```

Generate PDF report

1. Create an instance of **GcHtmlRenderer** class and pass the data bound template as a parameter.
2. Define the PDF settings, like header and footer, for rendering HTML to PDF using **PdfSettings** class.
3. Render the PDF file using **RenderToPdf** method of **GcHtmlRenderer** class.

C#

```
//Render the bound HTML
var tmp = Path.GetTempFileName();
using (
    var re = new GcHtmlRenderer(boundTemplate))
{
    //PdfSettings allow to provide options for HTML to PDF conversion
    var pdfSettings = new PdfSettings()
    {
        Margins = new Margins(0.2f, 1, 0.2f, 1),
        IgnoreCSSPageSize = true,
        DisplayHeaderFooter = true,
        HeaderTemplate = "<div style='color:#1a5276; font-size:12px;
width:1000px; margin-left:0.2in; margin-right:0.2in'>" +
            "<span style='float:left;'>Product Price List</span>" +
            "<span style='float:right'>Page <span class='pageNumber'></span> of
<span class='totalPages'></span></span>" +
            "</div>",
        FooterTemplate = "<div style='color: #1a5276; font-size:12em;
width:1000px; margin-left:0.2in; margin-right:0.2in;'>" +
            "<span>(c) GrapeCity, Inc. All Rights Reserved.</span>" +
            "<span style='float:right'>Generated on <span class='date'></span>
</span></div>"
    };
    //Render the generated HTML to the temporary file
    re.RenderToPdf(tmp, pdfSettings);
}
//Copy the created PDF from the temp file to target stream
using (var ts = File.OpenRead(tmp))
    ts.CopyTo(stream);
//Clean up
File.Delete(tmp);
}
```

Save PDF report

Invoke 'CreatePdf' method to save the PDF report.

C#

```
static void Main(string[] args)
{
    var fname = "ProductListTemplate.pdf";
    Console.WriteLine($"Press ENTER to create '{fname}' in the current directory,\nor
enter an alternative file name:");
    var t = Console.ReadLine();
    if (!string.IsNullOrWhiteSpace(t))
        fname = t;
    fname = Path.GetFullPath(fname);
    Console.WriteLine($"Generating '{fname}'...");

    var sample = new GcPdfWeb.Samples.ProductListTemplate();
    using (FileStream fs = new FileStream(fname, FileMode.Create))
    {
        sample.CreatePDF(fs);
        Console.WriteLine($"Created '{fname}' successfully.");
    }
    Console.WriteLine("Press any key to exit...");
    Console.ReadKey();
}
```

The final PDF report is a multi-page report, one of whose pages is displayed below:

Product Price List

Page 1 of 4

Products Stock Report

Description	Unit Price	Quantity Per Unit
Beverages		Total Units in Stock: 559
Chai	18	39
Chang	19	17
Guaraná Fantástica	4.5	20
Sasquatch Ale	14	111
Steeleye Stout	18	20
Côte de Blaye	263.5	17
Chartreuse verte	18	69
Ipoh Coffee	46	17
Laughing Lumberjack Lager	14	52
Outback Lager	15	15
Rhönbräu Klosterbier	7.75	125
Lakkalikööri	18	57
Condiments		Total Units in Stock: 507
Aniseed Syrup	10	13
Chef Anton's Cajun Seasoning	22	53
Chef Anton's Gumbo Mix	21.35	0
Grandma's Boysenberry Spread	25	120
Northwoods Cranberry Sauce	40	6

(c) GrapeCity, Inc. All Rights Reserved.

Generated on 14/2/2020

API Reference

This section contains documentation for all the assemblies required to create applications using GcPdf.

Assembly	Description
Grapecity.Documents.Barcode	Cross-platform library that provides an object model for creating barcodes.
Grapecity.Documents.Common	Cross-platform library that provides infrastructure and interfaces, font processing and text analysis.
Grapecity.Documents.Common.Windows	Platform-specific library that allows other GrapeCity.Documents packages to work with optimized system APIs on Windows.
Grapecity.Documents.Html	Cross-platform library that provides HTML processing and rendering features.
Grapecity.Documents.Imaging	Cross-platform library for working with raster images.
Grapecity.Documents.Pdf	Cross-platform library that allows you to create, analyze, and modify PDF documents.

Release Notes

Release notes are available for both GcPdf and GcPdfViewer.

- [Breaking Changes](#)
- [GcPdf Release Notes](#)
- [GcPdfViewer Release Notes](#)

Breaking Changes

Refer to the GcPdf breaking changes for specific versions:

- [Version 5.1.0.790](#)
- [Version 5.0.0.762](#)
- [Version 4.2.0.719](#)
- [Version 4.1.0.658](#)
- [Version 4.0.0.632 \(Maintenance\)](#)
- [Version 3.1.0.548](#)

Refer to the GcPdfViewer breaking changes here:

- [Version 3.1.2](#)

GcPdf Release Notes

Refer to the following release notes for the major releases of GcPdf.

- [Release Notes for Version 5.1.0.790](#)
- [Release Notes for Version 5.0.0.762](#)
- [Release Notes for Version 4.2.0.719](#)
- [Release Notes for Version 4.2.0.715](#)
- [Release Notes for Version 4.1.0.658](#)
- [Release Notes for Version 4.0.0.616](#)
- [Release Notes for Version 3.2.0.548](#)
- [Release Notes for Version 3.1.0.508](#)
- [Release Notes for Version 3.0.0.414](#)
- [Release Notes for Version 2.2.0.310](#)
- [Release Notes for Version 2.1.0.260](#)
- [Release Notes for Version 2.0.0.200](#)

For details about latest hot fixes, see the [nuget page](#).

Version 5.1.0.790

Breaking Changes

- GrapeCity.Documents.Pdf.TimeStamp.HashDelegate declaration changed to HashDelegate(byte[] dataToHash, Stream streamToHash, out byte[] hash, out OID hashAlgorithmOid).

New Features and Improvements

- Added GrapeCity.Documents.Pdf.Layers.ViewState class which represents the view state of a PDF document.

Allows specifying visible layers, current zoom and other transient info when exporting PDFs to images, searching for text etc.

- Added GcPdfDocument.SavingDocument event which is fired periodically when the document is being saved or exported. It can be used to implement a progress indicator.
- Added GrapeCity.Documents.Pdf.Security.DocumentSecurityStore class which represents a Document Security Store (DSS). It holds information that can be used to verify signatures offline.
- Added GcPdfDocument.SecurityStore property which gets the DocumentSecurityStore object associated with this document. It enables support for PAdES B-LT, B-LTA and LTV enabled signatures.
- Added GrapeCity.Documents.Pdf.TimeStampProperties class which represents properties used to time stamp a PDF document.
- Added GcPdfDocument.TimeStamp() methods which add a document time stamp and save the current document to a file or stream.
- Added GrapeCity.Documents.Pdf.Page.SaveAsSvg() methods which save the current page to a stream of file in SVG format.
- Added GrapeCity.Documents.Pdf.Page.ToSvgz() method which saves the page to a byte array in SVGZ format.

Version 5.0.0.762

New Features and Improvements

- Ability to render SVG (Scalable Vector Graphics) images to PDF, see GrapeCity.Documents.Svg.GcSvgDocument class and GcGraphics.DrawSvg()/MeasureSvg() methods.
- Method GcPdfDocument.OptimizeFonts(): optimizes font usage by merging subsets of same fonts, and by removing duplicate and unused fonts.
- An arbitrary PDF can be linearized ("fast web view") by loading it into GcPdfDocument and saving with SaveMode.Linearized parameter passed to an appropriate Save()/Sign() method overload.

Changes From the Previous Version

- When using a Security Handler Revision 4 or earlier with unspecified owner password, it is set to user password. This behavior is consistent with PDF spec.

Resolved Issues

- After loading some PDFs the value of GcPdfDocument.Linearized property is incorrect.
- StackOverflow exception occurs when merging or linearizing certain PDFs.
- Pen.DashOffset is not handled correctly when rendering to GcPdfGraphics.
- Saving a certain PDF to PNG produces incorrect result.

Breaking Changes

- GcPdfDocument.Linearized property is now read-only (was read-write). Use GcPdfDocument.Save(.., SaveMode.Linearized) to linearize the current document.

Breaking changes affecting all GrapeCity.Documents packages:

- GrapeCity.Documents.Common package has been removed, types defined in it have been moved to GrapeCity.Documents.Imaging.
- GrapeCity.Documents.Common.Windows package has been replaced by GrapeCity.Documents.Imaging.Windows.
- GrapeCity.Documents.Pdf.Resources has been removed, types defined in it have been moved to GrapeCity.Documents.Pdf.

Version 4.2.0.719

New Features and Improvements

- Added GrapeCity.Documents.Pdf.ISignatureParser interface that defines properties and methods that allow parsing and validating a PDF binary signature.
- Added GrapeCity.Documents.Pdf.ISignatureBuilder interface that defines methods used to build the signature PDF dictionary and the binary signature container.
- Added GrapeCity.Documents.Pdf.IPkcs7SignatureGenerator interface that defines properties and methods that are used to sign the attribute set in a PKCS#7 signature.
- Added GrapeCity.Documents.Pdf.Security.OID class that represents a cryptographic object identifier. Defines IDs of many popular cryptographic items such as HASH algorithms, encoding algorithms etc.
- Added GrapeCity.Documents.Pdf.Pkcs7SignatureBuilder class that implements ISignatureBuilder, can be used to build a PKCS#7 signature.
- Added TimeStamp.HashAlgorithm property that specifies the ID of the hash algorithm used to encode the time-stamp request.
- Added TimeStamp.HashMethod property that specifies the delegate used to hash the time-stamp request.
- Added Signature.CreateParser() method that creates an ISignatureParser object that can be used to parse a binary signature.
- Added SignatureProperties.CreatePAdES_B_B() method that creates a SignatureProperties object and initializes it so that it will create a PAdES B-B signature.
- Added SignatureProperties.CreatePAdES_B_T() method that creates a SignatureProperties object and initializes it so that it will create a PAdES B-T signature.
- Added SignatureProperties.SignatureBuilder property that specifies the ISignatureBuilder object used to build the signature.
- Added TextLayout.SimplifiedAlignment property in GrapeCity.Documents.Common package that specifies whether to use simplified text alignment rules. In particular, the same rules will be applied to narrow and wide (East Asian) characters.

Changes From the Previous Version

- Enum GrapeCity.Documents.Pdf.SignatureFormat marked as obsolete.
- Enum GrapeCity.Documents.Pdf.SignatureDigestAlgorithm marked as obsolete.
- Property SignatureProperties.SignatureDigestAlgorithm marked as obsolete.
- Property SignatureProperties.SignatureFormat marked as obsolete.
- Property SignatureProperties.Certificate marked as obsolete.

Resolved Issues

- Incorrect fallback font is selected in some cases.

Breaking Changes

- Property TimeStamp.DigestAlgorithm removed. Use TimeStamp.HashAlgorithm and TimeStamp.HashMethod instead.
- Class GrapeCity.Documents.Pdf.Security.SignatureContent removed. Use Signature.CreateParser() instead.

Version 4.2.0.715

New Features and Improvements

- Added StandardSecurityHandlerRev6 class that allows encoding or decoding PDF documents using AES Revision 6 encryption.

- Added GcPdfDocument.OutputIntents property, OutputIntent and ICCProfile classes to support for PDF Output Intents.
- Added GcPdfDocument.OptionalContent property, types in GrapeCity.Documents.Pdf.Layers namespace to support Layers (PDF Optional Content).
- Added GcPdfGraphics.BlendMode property, GcPdfGraphics.IsBlendModeSupported method to support blend modes and transparency groups.

Changes From the Previous Version

- Moved GcPdfGraphics extension methods from GrapeCity.Documents.Pdf.Util to GrapeCity.Documents.Pdf namespace: DrawPdfPage, DrawCheckBox, DrawComboBox, DrawCombTextBox, DrawListBox, DrawPushButton, DrawImage, DrawUnsignedSignature, DrawTextBox, DrawPdfLine.

Resolved Issues

- Miscellaneous bug fixes.

Version 4.1.0.658

New Features and Improvements

- Added GrapeCity.Documents.Pdf.Recognition.Structure.LogicalStructure class which represents the parsed logical structure of a document, created from tags in the PDF structure tree.
- Added GrapeCity.Documents.Pdf.Recognition.Structure.Element class which represents a parsed PDF tag (structure element) in the document's logical structure.
- Added class GrapeCity.Documents.Pdf.Recognition.Structure.ContentItemBase: abstract representing a portion of document content associated with a PDF structure tag (element).
- Added class GrapeCity.Documents.Pdf.Recognition.Structure.ContentItem : ContentItemBase: abstract class representing a content item associated with a portion of a content stream.
- Added class GrapeCity.Documents.Pdf.Recognition.Structure.McidContentItem : ContentItem: represents a parsed GrapeCity.Documents.Pdf.Structure.McidContentItemLink.
- Added class GrapeCity.Documents.Pdf.Recognition.Structure.McrContentItem : ContentItem: represents a parsed GrapeCity.Documents.Pdf.Structure.McrContentItemLink.
- Added class GrapeCity.Documents.Pdf.Recognition.Structure.ObjrContentItem : ContentItemBase: represents a parsed GrapeCity.Documents.Pdf.Structure.ObjrContentItemLink.
- Added GcPdfDocument.GetLogicalStructure() method which parses the PDF's structure tree and creates a LogicalStructure object that represents the logical structure of the document.
- Added GrapeCity.Documents.Pdf.TextMap.ITextRun interface which represents a portion of a text paragraph with the same formatting, possibly spanning several lines.
- Added GrapeCity.Documents.Pdf.TextMap.ITextRunFragment interface which represents a fragment of a text run that resides on a single text line.
- Added properties on ITextParagraph: TextMap, Page, Runs.
- Added properties on ITextLine: Paragraph, RunFragments.
- Added ITextParagraph.GetTextRuns() method which finds text runs in the paragraph that contain a specified text fragment.
- Added FindTextParams.Regex property which specifies whether the search text should be interpreted as a regular expression. Also added a corresponding optional argument to FindTextParams ctor, and a utility method FindTextParams.CreateRegex().

Resolved Issues

- Incorrect rendering of PDFs containing Adobe Type 1 Fonts.

- Free text annotation border is drawn even if line width is 0.
- In some cases incorrect fonts are used when a PDF is saved as image.
- Memory usage is too high when a PDF containing many large images is saved as image.
- In some scenarios GcPdfDocument.MergeWithDocument() may produce invalid PDFs.

Breaking Change

- Removed events GcPdfDocument.GeneratingDocument and GcPdfDocument.SavingDocument.

Version 4.0.0.616

New Features and Improvements

- Added Page.GetTable() method which tries to find a table within specified bounds and returns an ITable interface that provides access to common table OM (Rows, Columns, Cells).
- Added GrapeCity.Documents.Pdf.Recognition.TableExtractOptions class which represents the table extractor algorithm options.
- Added GrapeCity.Documents.Pdf.Recognition.RecognitionAlgorithm enumeration which defines possible algorithms that can be used to recognize the logical structure of a PDF when building text maps.
- Added GcPdfDocument.RecognitionAlgorithm property which specifies the algorithm that is used for PDF content recognition when building page text maps
- Added GetPoints() to ITextMap, ITextLine, ITextParagraph which gets the polygon that contains a text fragment, useful when the text bounds are non-rectangular.
- Added Field.GcProps property which gets the PdfDict object that can be used to associate arbitrary data with this field.
- Added Field.HasGcProps property which indicates whether the GcProps is not empty.
- Added GrapeCity.Documents.Pdf.Spec.IPdfDictHolderExt class which provides extension methods for working with IPdfDictHolder.
- Added GrapeCity.Documents.Pdf.Spec.IPdfArrayHolderExt class which provides extension methods for working with IPdfArrayHolder.

Resolved Issues

- Fixed the issue where text bounds found by GcPdfDocument.FindText() methods may be incorrect if a transformation is applied to the page.
- Fixed the issue where an exception occurred while rendering certain PDFs to images.

Version 4.0.0.632 (Maintenance) - Breaking Changes

- Removed WidgetAnnotation.DefaultAppearanceString, FreeTextAnnotation.DefaultAppearanceString, RedactAnnotation.OverlayTextAppearanceString and RedactAnnotation.OverlayTextFormat properties to ensure compliance with PDF specification. Newly added properties DefaultAppearance and OverlayAppearance can be used instead.
- WidgetAnnotation.TextFormat is deprecated, the new WidgetAnnotation.DefaultAppearance property should be used instead.

Version 3.2.0.548

New Features and Improvements

- Support to extract text paragraphs (see `ITextParagraph` interface). Added `Paragraphs` property to `ITextMap` and `ITextLine` interfaces.
- Enhanced support for font subsetting. Added new properties to `FontHandler` class: `Utf32CodeSet` and `FontSubsetFlags`.
- Added new member to `FontEmbedMode` enum: `EmbedSubsetNoForms`. Allows to embed a font's subset for static content while specifying that font as not embedded for form fields.
- Added `GcPdfDocument.FormEmbedUtf32CodeSet` property: specifies which Unicode characters to include in embedded font subsets for fonts that are used in AcroForms.
- Added `GrapeCity.Documents.Pdf.Util.GcGraphicsExt` class providing extension methods that allow to draw certain PDF elements (such as whole PDF pages or AcroForm fields) on a `GcGraphics`.

Changes From the Previous Version

- The `options` parameter of the `GcPdfDocument.MergeWithDocument()` method now is optional.
- The `searchRange` parameter of the `GcPdfDocument.FindText()` method now is optional.
- Improved handling of fallback fonts when saving PDFs to images.

Resolved Issues

- Fixed incorrect loading of some PDFs (`DecodeParams`).
- Fixed incorrect loading of some PDFs (`LZWDecode`).

Breaking Changes

- The type of property `FoundPosition.Bounds` changed from `Quadrilateral` to `Quadrilateral[]`.

Version 3.1.0.508

New Features and Improvements

- Added `TextFormat.EnableFontHinting` property, true by default. It enables executing TrueType hinting instructions on rendering text using `BitmapRenderer`.
- Added `Font.EnableHinting` property, true by default. If set to false, it prevents executing TrueType hinting instructions in `BitmapRenderer` for the current font.
- Added `TextLayout.Append()` overloads that allow to append an array of UTF-32 code points to the `TextLayout`.
- Added static `GcHtmlRenderer.SetGcPdfLicenseKey()` and instance `GcHtmlRenderer.ApplyGcPdfLicenseKey()` methods. It allows to create up to five PDFs without a license.
- Added static `GcHtmlRenderer.SetGclImagingLicenseKey()` and instance `GcHtmlRenderer.ApplyGclImagingLicenseKey()` methods. It allows to create up to five images without a license.
- Added `PdfSettings.TaggedPdf` Boolean property. It allows to generate tagged (accessible) PDF files from HTML.
- Added `GcHtmlRenderer.AuthServerWhitelist` property. It allows to authenticate the user with Integrated Authentication to an Intranet server or proxy without prompting the user for a user name or password.
- Added `GcHtmlRenderer.ProxyServer` property. It allows to use a custom proxy configuration.
- Added `authServerWhitelist` and `proxyServer` parameters to `DrawHtml()` and `MeasureHtml()` extension methods.
- Added support for applying redact annotations.
- Added Boolean property `SaveAsImageOptions.IgnoreErrors` (true by default).
- Added optional Boolean parameter `ignoreErrors` to `Page.Draw()` and `Page.DrawAnnotations()` methods.
- Added support for JPEG2000 images in PDF when saving PDF file as image.
- Added `WidgetAnnotation.CheckStyle` property. It specifies the style of check mark used if the `WidgetAnnotation` is linked to `CheckBoxField` or `RadioButtonField`.
- Added `RadioButtonField.AddItem()` method. Creates and adds the `WidgetAnnotation` to the `Field.Widgets` collection.

- Added the ability to add named actions and assign them to GcPdfDocument.OpenAction.
- Added Boolean property SaveAsImageOptions.Print, which specifies whether an image is generated for printing. Visibility of PDF elements can depend on whether a PDF is previewed or printed. This property affects whether such elements are rendered.
- Added an optional print parameter to the Page.Draw() method.
- Added SaveAsImageOptions.EnableFontHinting Boolean property (true by default).
- Added Signature.Content property, the returned object can be used to fetch additional info from a signature's binary data, e.g. to get the X509Certificate that was used to generate the signature. (DOC-1355)

Changes From the Previous Version

- In previous versions, RadioButtonField was always rendered as a circle (with a point inside if checked). Now, the look is determined in the same way as for CheckBoxField, see WidgetAnnotation.Border and WidgetAnnotation.CheckStyle.
- Improved the handling of broken PDF files. Now, GcPdf will try to ignore errors in the PDF content if possible.

Resolved Issues

- Fixed the issue where GcHtmlRenderer did not work correctly with file URIs.
- Fixed the issue where WatermarkAnnotation was not printed when saving PDF as image.
- Fixed the issue where an exception occurs when building text maps for certain PDFs.
- Fixed the issue where while creating a PDF with radio buttons, setting FieldFlags.RadiosInUnison to false was ignored.
- Fixed the incorrect embedding of a subset of IPAmj Mincho font.
- Fixed the errors that occurred when processing certain PDF files.
- Fixed the errors that occurred while saving some PDF files to images.
- Fixed the errors that occurred where Method Page.Draw() works incorrectly if the destination graphics has a non-identity transform.

Version 3.0.0.414

New features and improvements

- Added GcPdfDocument.ImportFormDataFromCollection(KeyValuePair<>[] fieldValues) method. It allows to import fields' values from collection.
- Added support for rendering gradients and tiling patterns.
- Added support for rendering cache for glyph paths and images.

Changes From the Previous Version

- All CMap files have been converted to binary form and moved to GrapeCity.Documents.Pdf.dll.

Resolved Issues

- The Newly created GcPdfDocument using HatchBrush renders correctly using SaveAsXXX(...) methods.
- Fixed the issue where an exception occurred on merging PDF files.
- Annotation now renders correctly if it has appearance content stream and annotation's size does not match content stream size.
- Fixed the issue where certain PDF files were rendered incorrectly if they contained images with soft-mask whose size differed from image's size.

Version 2.2.0.310

New features and improvements

The following features have been added with this version of the product.

- Added GcPdfDocument.ImportFormDataFromFDF(), GcPdfDocument.ImportFormDataFromXFDF() and GcPdfDocument.ImportFormDataFromXML() methods to import document's form data from FDF, XFDF and XML formats.
- Added GcPdfDocument.ExportFormDataToFDF(), GcPdfDocument.ExportFormDataToXFDF() and GcPdfDocument.ExportFormDataToXML() methods to allow users to export document's form data to FDF, XFDF and XML formats.
- Added GcPdfDocument.ImportFormDataToFDF(), GcPdfDocument.ImportFormDataToXFDF() and GcPdfDocument.ImportFormDataToXML() methods to allow users to import document's form data to FDF, XFDF and XML formats.
- Added RedactAnnotation class to support redact annotations. However, it is important to note that GcPdf doesn't support the actual removal of the content which is marked for redaction.
- Added Page.Draw() and Page.DrawAnnotations() methods to support PDF rendering on the GcGraphics class.
- Added SaveAsBmp(), SaveAsPng(), SaveAsGif(), SaveAsJpeg() and SaveAsTiff() methods in the GcPdfDocument class, which convert the PDF document to various image formats. Also, the methods Page.SaveAsPng() and Page.SaveAsBmp() have been added.

Changes From the Previous Version

This version of the product has the following changes.

- Earlier, some PDF generators used to render the same text multiple times at nearly the same position in order to emulate bold the text. But now, GcPdf handles such scenarios more efficiently while building the text map.
- Now, the property SignatureField.Value is writable. This means that it will allow users to remove an existing signature.

Resolved Issues

The following issues have been resolved since the last release.

- The internal document signature flags are now correctly updated while adding or removing the signature fields.
- The method GcPdfDocument.GetText() now works correctly with all the PDF files.
- Now, the StackOverflow exception is not thrown while loading certain PDF files.
- Now, the NotImplementedException is not thrown while enumerating the ThreadArticleBeadCollection and everything works appropriately.

Version 2.1.0.260

New features and improvements

The following features have been added with this version of the product.

- Added static property ICMAPProvider GcPdfDocument.CMapProvider, allows to define ICMAPProvider which will be used by GcPdfDocument to request predefined CMap's not existing in GrapeCity.Documents.Pdf.dll.
- Added GrapeCity.Documents.Pdf.Resources.dll containing additional resources used by GrapeCity.Documents.Pdf.dll, currently it contains additional predefined CMap's.
- AnnotationBase.DefaultAppearanceString was removed, WidgetAnnotation.DefaultAppearanceString and FreeTextAnnotation.DefaultAppearanceString were added, use them instead.
- Added properties TextField.RichTextValue, TextField.DefaultStyleString, allows to set value of field as rich

formatted string, see PDF specification for more details. Note! GcPdf does not generate automatically appearance streams for RTF text fields, it is not supported.

- Added AssociatedFiles property for Page, GcPdfDocument, StructElement, AnnotationBase objects, allows to associate list of embedded files with object. This property can be used to generate PDF/A-3x documents if they contains embedded files.
- Added PdfAConformanceLevel Metadata.PdfA and GcPdfDocument.PdfAConformanceLevel properties allows to set PDF/A conformance level, GcPdfDocument.PdfAConformanceLevel is a wrapper around Metadata property, property GcPdfDocument.PdfACompliant now obsolete.
- Added Metadata.CreatorTool property it has same purpose as DocumentInfo.Creator.
- Added support of Sound annotations, see SoundAnnotation class.
- Added GcPdfGraphics.SoftMask property, allows to set FormXObject as a drawing mask.
- Method named ITextMap.GetSelectionxxx(...) is renamed to ITextMap.GetFragmentxxx(...).
- Added Page.TransitionEffect, Page.TransitionDuration properties, allows to define page behavior in presentation mode.
- Added GcPdfDocument.ArticleThreads list. It allows to define article threads in the document.
- Added ITextMap Page.GetTextMap(float dpiX = 72, float dpiY = 72) method. It allows to get page's text map represented by ITextMap interface which can be used for hit testing, retrieve list of text fragments on the page etc.
- Added GcPdfDocument.PageLabelingRanges property, allows to define dictionary of PageLabelingRange objects which define labels for document's pages.

Resolved Issues

The following issues have been resolved since the last release.

- Fixed the issue where exception occurred on using SignatureLockedFields(SignatureLockedFieldsType, IEnumerable<string>) constructor.
- Fixed the issue where appearance streams are generated incorrectly for Acroform fields in certain scenario.
- Fixed the issue where text renders incorrectly in GcPdfGraphics if FontSizeInGraphicUnits is true and GcPdfGraphics.Resolution is not 72.

GcPdfViewer Release Notes

Refer to the following release notes for the major releases of GcPdfViewer.

- [Release Notes for Version 3.1.2](#)
- [Release Notes for Version 3.0.10](#)
- [Release Notes for Version 2.2.15](#)
- [Release Notes for Version 2.2.11](#)
- [Release Notes for Version 2.1.14](#)
- [Release Notes for Version 2.0.10](#)
- [Release Notes for Version 1.2.88](#)
- [Release Notes for Version 1.1.56](#)
- [Release Notes for Version 1.0.42](#)

For details about latest hot fixes, see the [nuget page](#).

Version 3.1.2

Breaking Changes

- Version mismatch warning will not be shown anymore if the connected SupporApi has version 0.0.0.0 (was built from sources).

New Features and Improvements

- Added the new invalidate method to ensure all child elements of the viewer get properly updated for layout.
- Added the new requiredSupportApiVersion property to get the require version of SupportApi that is compatible with the current version of GcPdfViewer.
- Added the new gcPdfVersion property, which gets the version of GcPdf library used by the connected SupportApi, if available.
- [Editor] Added the support to rotate stamp and free text annotations using rotation handles.
- Added the support for pressing the Shift key and snapping the rotation angle to a multiple of 90 degrees.
- [XFA forms] Added support for print, submit, reset, JavaScript actions, links.
- [Editor] Added the support to persist the visibility state of optional content groups (layers) when saving the PDF.
- [Editor] Added the support to change a widget's content orientation using the orientation property.
- [Editor] Added the support for Sticky behavior for toolbar buttons.
- Added the ability to specify button fields render type.
- Added stickyBehavior setting to toolbarLayout, an array with button keys that will have sticky behavior. Note that only annotation and form editor toolbar buttons can be made sticky.
- Added the support to programmatically hide the left sidebar.
- Added the support to programmatically hide or show the toolbar.
- Added the support for Option hideAnnotationPopups, which hides all annotation popups.
- Added the support to close the currently loaded document.
- Added New events: onBeforeAddAnnotation, onAfterAddAnnotation, onBeforeUpdateAnnotation, onAfterUpdateAnnotation, onBeforeRemoveAnnotation, onAfterRemoveAnnotation. The events BeforeAddAnnotation, BeforeUpdateAnnotation and BeforeRemoveAnnotation are also cancelable.
- Added the support to listen to and trigger custom events.
- Added the support to specify authentication or other HTTP headers in the open() method.
- Added the support to specify authentication or other HTTP headers in SupportApi requests.
- [Form Editor] Added "Editable" property to combo boxes.
- [Form Editor] Added New values to "Tab order" property: "Annotations" and "Widgets".
- Added support to close the current document.
- Added support for editable combo boxes.
- [XFA forms] Added the support to select or copy text content.
- [Editor] Added support to remember the last-used editor values.
- Added new settings to the "editorDefaults" option: "rememberLastValues" and "lastValueKeys". If "rememberLastValues" is set to true or undefined, the last used property values will be used as default values for new annotations.
- [Android] Added support for zooming using pinch gesture.

Bug Fixes

- [Editor] Fixed the issue where the annotations and fields were added in incorrect orientation when the document was rotated.
- [Android] Fixed the issue where zooming using pinch did not work.
- [Collaboration] Fixed the issue where sharing a multi-page PDF was not working correctly. (DOC-4143)
- [Collaboration] Fixed the issue where The file name was incorrect in the "Manage Access" dialog. (DOC-4144)
- [JavaScript actions] Fixed the issue where the Viewer properties were undefined in JavaScript actions. (DOC-4154)
- Fixed the issue where in some cases the visibility state of layers was incorrect after saving a PDF. (DOC-4067)
- [Editor] Fixed the issue where Arrow keys moved annotations incorrectly when the document was rotated. (DOC-4129)
- [Editor] Fixed the issue where Stamp annotation disappeared from page after printing the document. (DOC-4127)
- [Form Editor] Fixed the issue where the Text cursor moves when an annotation was moved using the arrow

keys. (DOC-4123)

- Fixed the issue where in some cases the visibility state of layers was incorrect after loading or saving a PDF. (DOC-4068)
- Fixed the issue where the renderInteractiveForms was false and checkbox values were not displayed. (DOC-4022)
- Fixed the issue where the highlight on the search text disappeared when the document was zoomed in or out. (DOC-4028)
- [Editor] Fixed the issue where the custom image stamps were gone when the viewer was closed and recreated. (DOC-4023)
- [Editor] Fixed the issue where the text added in the free text annotation editor disappeared on resizing the annotation. (DOC-3988)
- Fixed the issue where the undo history should be cleared by the close() method. (DOC-3989)
- Fixed the issue where the incorrect tab cycle was shown for an editable combo box. (DOC-3967)
- Fixed the issue where the Tab order differs from Adobe Acrobat Reader in some cases. (DOC-3668)
- [Form Editor] Fixed the issue where the user cannot reset the "Tab order" property to "Not specified". (DOC-3968)
- [Editor] Fixed the issue where the Backspace key did not work in the free text annotation editor. (DOC-3983)
- [Editor] Fixed the issue where a new page was inserted and the document was saved, the tab order of following pages was incorrect. (DOC-3985)
- [iPad] Fixed the issue where the PDF will not render after max zooming when iPad was running in Desktop mode. (DOC-3765)
- [iOS][Android] Fixed the issue where the main toolbar collapsed when the secondary toolbar was clicked. (DOC-3843)
- [Editor] Fixed the issue where the Method showSecondToolbar did not activate editor mode correctly. (DOC-3892)
- [Editor] Fixed the issue where there was incorrect undo/redo behavior when editing ink annotations. (DOC-3924)
- Fixed the issue where the incorrect zoom controlled behavior. (DOC-3929)

Version 3.0.10

New Features and Improvements

- Added Layers panel which lists and enables users to show/hide individual PDF layers (optional content).
- Added ability to edit document without switching to Annotation Editor or Form Editor modes.
- Added support for second horizontal toolbar.
- Added secondary editing toolbars to the main viewer toolbar: "Text tools", "Draw tools", "Attachments and stamps", "Form tools", "Page tools".
- Added API to display a custom second toolbar.
- Added Light and Dark themes.
- Support page content accessibility for tagged PDFs containing logical structure information for screen readers.
- Added ability to show the structure tree of tagged PDFs.
- Added goToPage method: navigate to the page with a specified 0-based index.
- Added option maxCanvasPixels: maximum supported canvas size in pixels, i.e. width * height. Undefined or -1 means no limit.
- Added the ability to use GET method to submit a form.
- Action reset: added support for FieldNames, ExcludeSpecifiedFields and Next properties.
- Added new option fieldsAppearance - specifies how form fields are rendered.
- Added enableXfa option: render XFA (XML Forms Architecture) forms if any; the default is true.
- Added requireTheme option. Use this option to apply a built-in CSS theme, this will inject the theme styles directly into the page head.
- Added onThemeChanged event: raised when the user changes the viewer theme.
- Added onInitialized option: the onInitialized handler will be called immediately after the viewer is instantiated.

Resolved Issues

- Multiple bug fixes.

Breaking Changes

- All public APIs that used page numbers now use zero-based page indices.
- PDF.js library was updated from v2.0.943 to v2.10.377, see [PDF.js [Release Notes](#)].
- Method goToPageNumber deprecated, use goToPage method or pageIndex property instead.
- By default, radio buttons and checkboxes now do not use predefined appearances from the PDF.
- Use the fieldsAppearance option to revert to the old behavior:
- [SupportApi client] The ping() method has been deprecated and is no longer used; instead, the serverVersion() method is used.
- Properties SubmitForm/ResetForm renamed to submitForm/resetForm.

Version 2.2.15

New Features and Improvements

- Added Layers panel which lists and enables users to show/hide individual PDF layers (optional content).
- Added openPanel() method to open a side panel.
- Added closePanel() method to close the side panel.
- Added resetChanges() method to reset the document to its original state, discarding all changes.
- Added setPageRotation(pageIndex, rotation) method to enable users to rotate a specific page in the PDF. This method requires SupportApi. Valid values for are 0, 90, 180, and 270 degrees.
- Added getPageRotation(pageIndex) method to get the rotation value for a specified page.

Resolved Issues

- When an annotation is added in code and saved, its name changes.
- The state of the Signature Tool is not cleared after recreating the GcPdfViewer component.
- Console shows an error after calling viewer.newDocument().
- Incorrect behavior if viewer.newDocument() is called immediately after opening a PDF.

Version 2.2.11

New Features and Improvements

- Added predefined stamps support to the Stamp tool.
- Allow the user to select font family for text fields and free text annotations.
- Allow the user to specify opacity for annotations.
- Allow the user to specify annotations tab order.

Resolved Issues

- Multiple bug fixes.

Version 2.1.14

New Features and Improvements

- Added Graphical Signature tool/dialog.
- Added Stamp Annotation tool (allows to add images as page content).
- Added the ability to lock annotations or fields in editors.
- Added Link Annotations support.
- Improved support for mobile devices.

Version 2.0.10

New Features and Improvements

- Added Form Filler feature
- Added Collaboration Mode feature
- Added the ability to convert annotations and fields to content
- Using SupportApi in ASP.NET Core projects now requires ASP.NET Core 3.0 or later

Version 1.2.88

New Features and Improvements

- Added ability to specify boolean value for snap tolerance and disable vertical or horizontal snap.
- Added alignment property for Comb-text field.
- Added support for JavaScript actions from additional-actions dictionary for field widgets.
- Added 'highlight all' feature for Search panel.
- Added zoomMode property.
- Added snap alignment in Annotation/Form editor, enabled by default. Press Alt key if you wish to temporarily disable snap during resize or move action.
- Arrow keys move the current element and Shift-arrow resize the element in Annotation/Form editor.
- Viewer search improvements:
 - Proximity search
 - Starts with/ends with
 - Wildcards
- Added the ability to copy and paste annotations or fields in Annotation/Form editor.
- Added the ability to clone the current annotation/field in Annotation/Form editor.
- Added text annotation reply tool (opens on the right side of the viewer). To enable, use the addReplyTool() method.
- Added viewer context menu and the ability to customize it.
- Added editorDefaults option, use this option if you wish to change some default editor values.
- Show the total number of results in the search panel.

Changes From the Previous Version

- Free text annotation: property Color renamed to Backcolor; property Border color replaced by Forecolor.
- Text annotation: group all replies to an annotation together as threaded comments.
- Form editor: field captions in the list now show field names.
- Added the ability to collapse the property panel pages using the chevron icon.

Resolved Issues

- Miscellaneous bug fixes.

Version 1.1.56

New Features and Improvements

- Added new property supportApi, which connects the viewer to a server running GcPdf and enables PDF modification features (annotations, form design, redact etc.).
- Added Annotation editor panel that requires supportApi.
- Added Form editor panel that requires supportApi.
- Added Redact tools that requires supportApi.
- Added security tab in document properties.
- Added Navigation methods, goToPageNumber(), goToFirstPage(), goToPrevPage(), goToNextPage() and scrollPageIntoView().
- Added Localization support and properties dialog localization.
- Added error message to be displayed when document size exceeds SupportApi server size limit.
- Added ability to edit basic Link annotation properties.
- Added a new property: viewer.zoomValue (gets or sets the current zoom percentage level).

Changes From the Previous Version

- The viewer.toolbar.updateLayout() method is obsolete and should not be used, as it does not take the new editor modes into account. Instead use the viewer.toolbarLayout property and viewer.applyToolbarLayout() method.
- Updated Properties dialog style.
- Updated behavior of Annotation and Form editor's sidebar.
- Updated close button style and added translation keys to Properties dialog.

Resolved Issues

- Link annotation can now be navigated in editor mode.
- Sidebar pinned state is not retained when switching from editor mode to viewer mode.
- Properties panel does not collapse when button 'back to view tools' clicked in some cases.
- Fixed problem with incorrect color conversion (ColorUtils, hexToRgb) when color argument is specified using rgb model.
- Fixed the incorrect thumbnail scale when browser zoom level changed.

Version 1.0.42

New Features and Improvements

- Added the Document Properties dialog (including fonts).

Changes From the Previous Version

- Added the ability to open PDF attachments from Attachment panel in GcPdfViewer.
- Changed icon and button name for Outline panel button to "Bookmarks", and also added the ability to collapse outline items.

Resolved Issues

- The Next page button now scrolls to the page even if part of it is already visible.
- The icon button in navigation panel now fully displays in edge browser.

- Now, the (iOS) Document content does not disappear when changing zoom option.
- Fixed the issue where on posting a form, incorrect values were sent for some radio buttons.
- Fixed the issue where on posting a form, incorrect names were sent for combo and list boxes.
- Fixed the issue of Text alignment in document properties dialog.
- The Close button now appears normally on the edge of the browser.
- Now, Thumbnails are shown correctly for non-standard page sizes.
- Fixed the issue where the page orientation was incorrect for print preview in some cases.
- Now, the Text outside of the document can be selected.

version 5.0.0.767

ReleaseNotes GcPDFViewer

version 3. 0. 13