Table of Contents

Document Solutions for PDF Overview

Document Solutions is a cross-platform solution for document management which provides a near universal document, editor and viewer solution for all popular document formats.

Document Solutions for PDF (DsPdf, previously GcPdf), is a part of Document Solutions 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, DsPdf 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

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

- **Generate, load, modify and save PDFs**
  Using DsPdf, 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**
  DsPdf 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**
  DsPdf 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**
  DsPdf 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**
  DsPdf library allows you to apply robust security while generating PDF documents. DsPdf 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**
  DsPdf supports incremental update, which among other things allows the addition of multiple digital signatures to a PDF document, while keeping them all valid.
- **Add form fields**
  DsPdf allows you to add, modify, and delete different form fields, such as text, check box, radio buttons, signature etc., to create interactive forms. With the help of form fields, you can easily create fillable forms in your PDF document.
- **Import and export form data**
  DsPdf provides the capability to import or export PDF forms data from or to XML, FDF and XFDF files.
- **Generate linearized PDF**
  DsPdf allows generation of linearized PDF files to help you load your files quickly.
- **Rich set of features**
  DsPdf library provides a rich set of features allowing the generation of complex PDF documents with content including text, graphics, images, annotations, outlines and more.
- **Document Solutions PDF Viewer**
  Document Solutions PDF Viewer is a fast javascript based client-side application allowing users to view PDF documents. It supports many of the standard PDF features.
- **Seamless HTML to PDF rendering**
  DsPdf library along with DsHtml library, allows you to render HTML text or files to PDF documents.

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

Getting Started

System Requirements

The DsPdf packages are fully supported on Visual Studio 2017 or later for Windows, Visual Studio for MAC, and Visual Studio Code for Linux and are compatible with the following:

- .NET 5, .NET 6, and .NET 7
- .NET Core 2.x and 3.x
- .NET Standard 2.x
- .NET Framework 4.6.1 or higher

Setting up an Application

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

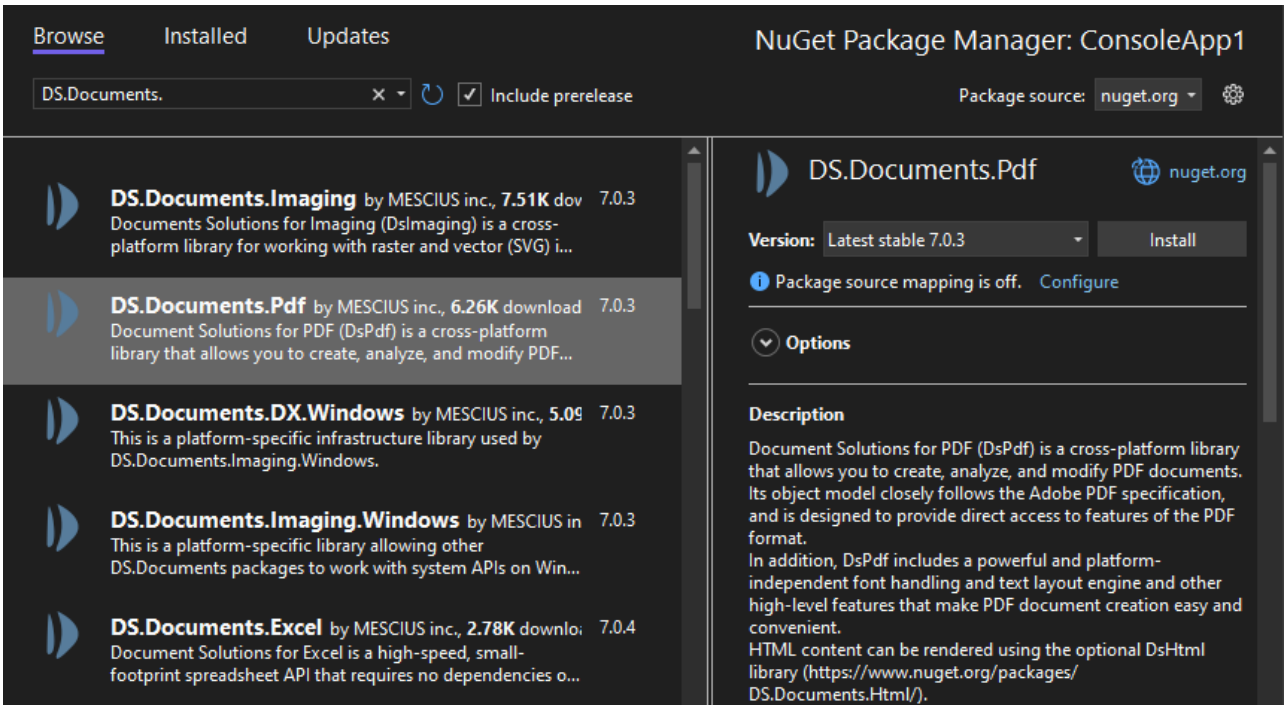
Reference	Purpose
DS.Documents.Pdf	To use DsPdf in an application, you need to reference (install) just the DS.Documents.Pdf package. It will pull in the required infrastructure packages.
DS.Documents.BarCode	To render barcodes, install the DS.Documents.Barcode (aka DsBarcode) package. It provides extension methods allowing to render barcodes when using DsPdf.
DS.Documents.Imaging	DS.Documents.Imaging provides image handling. You do not need to reference it directly.
DS.Documents.DX.Windows	DS.Documents.DX.Windows is an infrastructure package. You do not need to reference it directly.

Add reference to DsPdf in your application from NuGet.org

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

Visual Studio for Windows

1. Open Visual Studio for Windows.
2. Create a .NET Core Console/.NET Framework Windows Forms Application.
3. Right-click the project in Solution Explorer and choose **Manage NuGet Packages**.
4. In the **Package source** on top right, select **nuget.org**.
5. Click **Browse** tab on top left and search for "DS.Documents".
6. On the left panel, select **DS.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 **DS.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 "DS.Documents".
5. From the list of packages displayed in the left panel, select **DS.Documents.Pdf** (and **DS.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 "**DS**" and press Enter.
   Observe: DS packages get displayed in the dropdown.
10. Choose **DS.Documents.Pdf**.

11. (Optional) Repeat above steps to add **DS.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

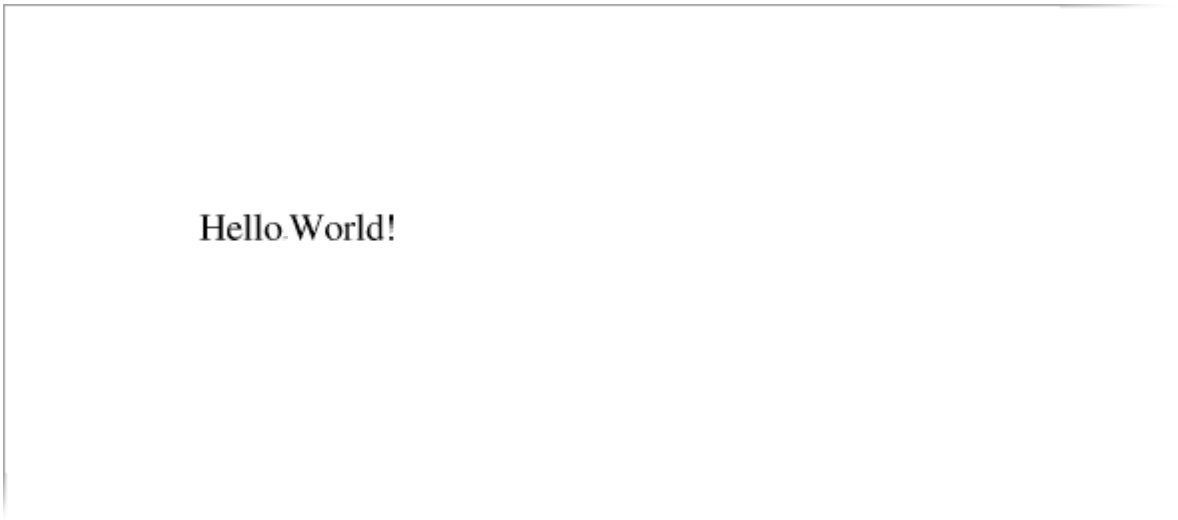
The following quick start sections help you in getting started with the DsPdf library:

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

Create and Save a PDF Document

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

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



Step 1: Create a new PDF document

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

C#

```
// Create a new PDF document:
GcPdfDocument doc = new GcPdfDocument();
// Add a page, and get its Graphics object to draw on:
GcPdfGraphics g = doc.NewPage().Graphics;
// Create a text format for the "Hello World!" string:
TextFormat tf = new TextFormat();
```

```
// Use standard Times font
tf.Font = StandardFonts.Times;
// Pick a font size:
tf.FontSize = 14;
```

Back to Top

Step 2: Draw a string on the PDF document

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

C#

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

Back to Top

Step 3: Save the document

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

C#

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

Back to Top

Load and Modify a PDF Document

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

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

Step 1: Load an existing document in DsPdf

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

C#

```
GcPdfDocument doc = new GcPdfDocument();

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

Back to Top

Step 2: Modify the document

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

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

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

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

Back to Top

Step 3: Save the document

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

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

Back to Top

License Information

Types of Licenses

Document Solutions 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.
- When saving a PDF file, a watermark is displayed on all the pages in that file.
  'Unlicensed copy of Document Solutions for PDF. Loading is limited to 5 pages. Contact us.sales@mescius.com to get your 30-day evaluation key.'

> Note that if you run a sample that uses a signed PDF without a valid DsPdf license, 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

DsPdf 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@mescius.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 Document Solutions for PDF. Loading is limited to 5 pages. Contact us.sales@mescius.com to purchase license.'

> Note that if you run a sample that uses a signed PDF without a valid DsPdf license, 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

DsPdf 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 DsPdf without any limitations.

Apply License

To apply evaluation/production license in DsPdf, 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 a DsPdf license from a previous major version to the current major version, you will need to obtain a new license key from the sales team. Once you have received a new license key through email, follow these steps:

1. Open an existing application created using DsPdf 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 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");`

> **Note:** With v7.0, GrapeCity.Documents.Pdf (GcPdf) package is renamed to DS.Documents.Pdf (DsPdf). The

namespaces and classes within DS.Documents.Pdf remain the same, which provide the same functionality and are backwards compatible with GrapeCity.Documents.Pdf, ensuring minimal impact on your existing projects.

To upgrade GcPdf package to DsPdf package in your existing projects, follow one of the below options:

- Update package using Migration tool:
  1. The migration tool is present in the package downloaded from the website. Follow the instructions displayed in the UI when using the tool for a seamless migration from GcPdf to DsPdf.
- Update package manually from NuGet package manager:
  1. In **Solution Explorer**, right-click either **Dependencies** or a project and select **Manage NuGet Packages**.
  2. In **Installed** tab, click on **GrapeCity.Documents.Pdf** package and click **Uninstall** to remove it and its dependencies from the project.
  3. In **Browse** tab, type "ds.documents" or "DS.Documents" in the search text box at the top and find the package "DS.Documents.Pdf".
  4. Click Install to add the **DS.Documents.Pdf** package and its dependencies to the project.

Technical Support

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

- Product forum: https://developer.mescius.com/forums
- Email: us.sales@mescius.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://developer.mescius.com/
E-mail	us.sales@mescius.com
Phone	(800) 858-2739 or (412) 681-4343 outside the U.S.A.
Fax	(412) 681-4384

Redistribution

To distribute an application containing the DsPdf API, it is required to have a valid Distribution License and environments meeting the minimum System Requirements.

DsPdf 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://developer.mescius.com/
E-mail	us.sales@mescius.com
Phone	1.800.858.2739 or 412.681.4343
Fax	(412) 681-4384

End-User License Agreement

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

Product Architecture

Packaging

DsPdf is a collection of cross-platform .NET class libraries written in C#, that provides an API that allows the creation of PDF files from scratch as well as loading, analyzing, and modifying existing documents.

DsPdf is compatible with .NET Core 2.x/3.x, .NET Standard 2.x, .NET Framework 4.6.1 or higher, and .NET 6 or higher.

DsPdf and supporting packages are available on nuget.org:

- **DS.Documents.Pdf**
- **DS.Documents.BarCode**
- **DS.Documents.Imaging**
- **DS.Documents.Imaging.Windows**
- **DS.Documents.DX.Windows**

To use DsPdf in an application, simply reference the **DS.Documents.Pdf** package. All other required packages that DsPdf utilizes will be installed automatically.

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

DS.Documents.DX.Windows provides access to the native imaging APIs to DsPdf if it runs on a Windows system.

DsPdf API Overview

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

Namespaces

Namespaces	Description
GrapeCity.Documents.Drawing	Framework for drawing on the abstract GcGraphics surface.
GrapeCity.Documents.Pdf	Types used to create, process and modify PDF documents includes GcPdfGraphics. Nested namespaces contain types supporting specific PDF spec areas: • GrapeCity.Documents.Pdf.AcroForms • GrapeCity.Documents.Pdf.Actions • GrapeCity.Documents.Pdf.Annotations • GrapeCity.Documents.Pdf.Graphics • GrapeCity.Documents.Pdf.Log • GrapeCity.Documents.Pdf.Parser • GrapeCity.Documents.Pdf.Security
GrapeCity.Documents.Text	Text processing sub-system.

GcPdfDocument

A PDF document in DsPdf 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 the analysis and/or modification of 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

DsPdf 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** must be used for each page. 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.)

DsHtml API Overview

DsHtml is a utility library that renders HTML to PDF file or an image in PNG, JPEG, and WebP format. DsHtml uses a Chrome or Edge browser (already installed in the current system, or downloaded from a public web site) in headless mode. 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 DS.Documents.Html library consists of a platform-independent main package that exposes the HTML rendering functionality. The main package contains the following namespaces:

Namespaces	Description
GrapeCity.Documents.Pdf	It provides the extension methods for rendering HTML to PDF file and represents the formatting attributes for rendering HTML to PDF file. The namespace comprises the following classes: • GcPdfGraphicsExt • HtmlToPdfFormat
GrapeCity.Documents.Html	It provides methods for converting HTML to PDF or images and defines parameters for the PDF or image. The namespace comprises the following classes: • BrowserFetcher • GcHtmlBrowser • HtmlPage • ImageOptions • JpegOptions • LaunchOptions • PageOptions • PdfMargins • PdfOptions • PngOptions • TimeOutOptions • WebpOptions
GrapeCity.Documents.Drawing	It provides the extension methods and formatting attributes for rendering HTML to image. The namespace comprises the following classes:

	• GcBitmapGraphicsExt • HtmlToImageFormat

GrapeCity.Documents.HTML.BrowserFetcher

The BrowserFetcher class has two static methods: **GetSystemChromePath**() and **GetSystemEdgePath**(). The methods return the path to an executable file of Chrome or Edge browsers correspondingly. Another option is to download and install Chromium into a local folder. You can create an instance of BrowserFetcher and pass the information such as host, platform, revision, and the destination folder, if needed. Then, execute the BrowserFetcher.**GetDownloadedPath**() method which downloads Chromium, if required, and returns the path to an executable file for running the Chromium.

GrapeCity.Documents.Html.GcHtmlBrowser

The GcHtmlBrowser class provides methods for converting HTML to PDF and images. With a path to an executable file for running either the Chromium or Edge browsers discovered in the BrowserFetcher class, we can create an instance of GcHtmlBrowser class, which effectively runs the browser process in the background. GcHtmlBrowser also accepts another parameter of **LaunchOptions** type. The LaunchOptions class provides various settings specific to launching the browser.

The class has two important methods: **NewPage(Uri uri)** and **NewPage(string html)**. Both methods return an instance of **HtmlPage** class which represents a browser tab after navigating to the specified web address, file, or the arbitrary HTML content. The second parameter of PageOptions type provides various properties to be applied to the new browser page such as username, password for HTTP authentication, disabling JavaScript, lazy loading etc.

> **Note:**
>
> - We recommend using Chrome browser with GcHtmlBrowser class as Edge has some differences in the implementation of some DevTools features.
> - It is important to dispose every instance of the **GcHtmlBrowser** and **HtmlPage** classes after use.

Grapecity.Documents.Html.HtmlPage

The HtmlPage class represents a browser tab after navigating to the specified web address, file, or the arbitrary HTML content. The class has methods such as **SaveAsPdf**, SaveAsPng, SaveAsJpeg, and SaveAsWebp to save the current page as a PDF or as a raster image of PNG, JPEG, or WebP formats respectively. The first parameter of these methods specifies the destination file or stream. The second parameter passes the additional options for rendering HTML page as single PDF page, setting page size, margins, header and footer etc.

The HtmlPage class contains the additional methods that help to interact with HTML page content. For example, you can obtain the full HTML content of the page using the GetContent method. The SetContent method updates the HTML markup. You can reload the web page with the Reload method or even execute a script in the browser context using the EvaluateExpression method. The WaitForNetworkIdle method helps with loading asynchronous web content.

GrapeCity.Documents.Html.PdfOptions

The PdfOptions class represents output settings for rendering HTML to PDF and defines parameters for the Chromium PDF exporter. In the case of PDF, it doesn't support any transparency.

If **PageWidth** and **PageHeight** properties are not set, the Letter paper size (8.5 by 11 inches) is used by default. **Landscape** property of the class indicates the paper orientation and is ignored if **FullPage** property is set to true. The **Margins** property specifies page margins, in inches and its default value is 0. The **Scale** property scales the content of PDF on the scale of 0.1 to 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 you 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 you to export the whole HTML as single PDF page. All other layout settings (except Scale) are ignored in that case.

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 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.

Other properties of HtmlToPdfFormat are mapped to the corresponding properties of the PageOptions/PdfOptions class:

HtmlToPdfFormat Property	PageOptions/PdfOptions Property
WindowSize	PageOptions.WindowSize
DefaultBackgroundColor	PageOptions.DefaultBackgroundColor
FullPage	PdfOptions.FullPage
DisplayBackgroundGraphics	PdfOptions.PrintBackground
Scale	PdfOptions.Scale
MaxPageWidth	PdfOptions.PageWidth
MaxPageHeight	PdfOptions.PageHeight

GcPdfGraphics Extension Methods

DsHtml 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(GcHtmlBrowser browser, string html, float x, float y, HtmlToPdfFormat format, out SizeF size, bool loadLazyImages = false)
- Draws an HTML page specified by a URI on this GcPdfGraphics at a specified position:
  bool GcPdfGraphics.**DrawHtml**(GcHtmlBrowser browser, Uri htmlUri, float x, float y, HtmlToPdfFormat format, out SizeF size, bool loadLazyImages = false)
- Measures an HTML text for this GcPdfGraphics:
  SizeF GcPdfGraphics.**MeasureHtml**(GcHtmlBrowser browser, string html, HtmlToPdfFormat format, bool loadLazyImages = false)
- Measures an HTML page specified by a URI for this GcPdfGraphics:
  SizeF GcPdfGraphics.**MeasureHtml**(GcHtmlBrowser browser, Uri htmlUri, HtmlToPdfFormat format, bool loadLazyImages = false)

> **Note**: In DsImaging release version 6.0.0, the **GcHtmlRenderer** class has been marked **obsolete** and has been replaced by the new **GcHtmlBrowser** class. This is done to avoid GPL or LGPL licensed software that had to be used in the custom chromium build. For tips about migration from obsolete GcHtmlRenderer class, see Tips to Migrate from Obsolete GcHtmlRenderer class.

Features

This section comprises the features available in the DsPdf.

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

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 it's orientation and size, and work with content streams.

Layouts
: Place multiple elements on a PDF page or image without having to calculate positions of each element relative to other ones.

Complex Graphic Layouts
: Draw complex graphics, text, and images.

Tables
: Create and work with tables easily and straightforwardly without having to think much about the size of table columns, merged cells, or the layout of rotated text.

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.

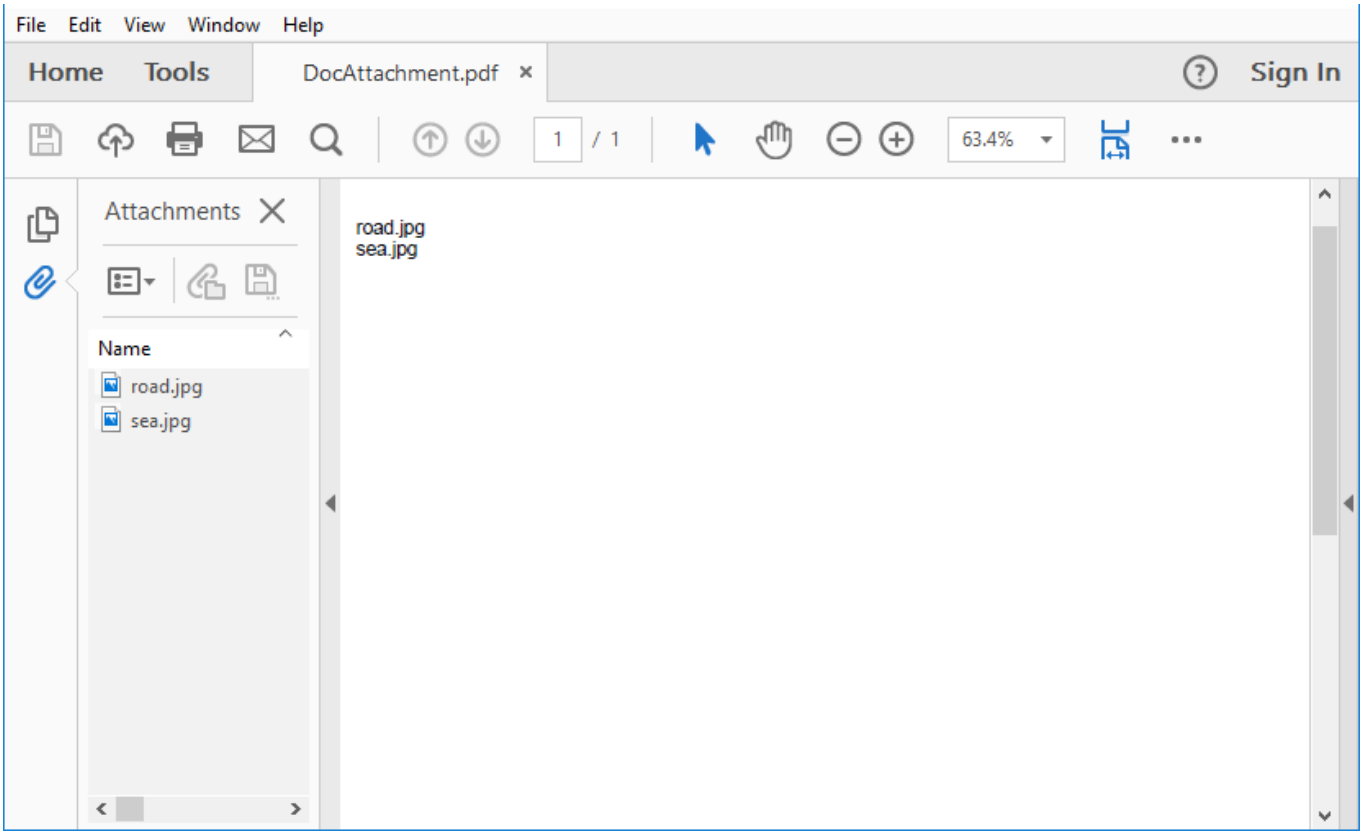
> **Note: DsPdf** library provides the following classes and interfaces in **GrapeCity.Documents.Pdf.Spec** and **GrapeCity.Documents.Pdf.Wrappers** namespaces to work directly with the low-level PDF primitives, which are the building blocks of any PDF document:
>
> - PDF array: PdfArray, PdfArrayObject, IPdfArray, IPdfArrayExt, and PdfArrayWrapper.
> - PDF bool: PdfBool, PdfBoolObject, IPdfBool, and IPdfBoolExt.
> - PDF dictionary: PdfDict, PdfDictObject, IPdfDict, IPdfDictExt, and PdfDictWrapper.
> - PDF name: PdfName, PdfNameObject, IPdfName, and IPdfNameExt.
> - PDF null: PdfNull, PdfNullObject, IPdfNull, and IPdfNullExt.
> - PDF number: PdfNumber, PdfNumberObject, IPdfNumber, and IPdfNumberExt.
> - PDF reference: PdfRef, PdfRefObject, IPdfRef, and IPdfRefExt.
> - PDF stream: PdfStreamObjectBase.
> - PDF string: PdfString, PdfStringObject, IPdfString, and IPdfStringExt.

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 DsPdf. 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 DsPdf. 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. DsPdf 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#

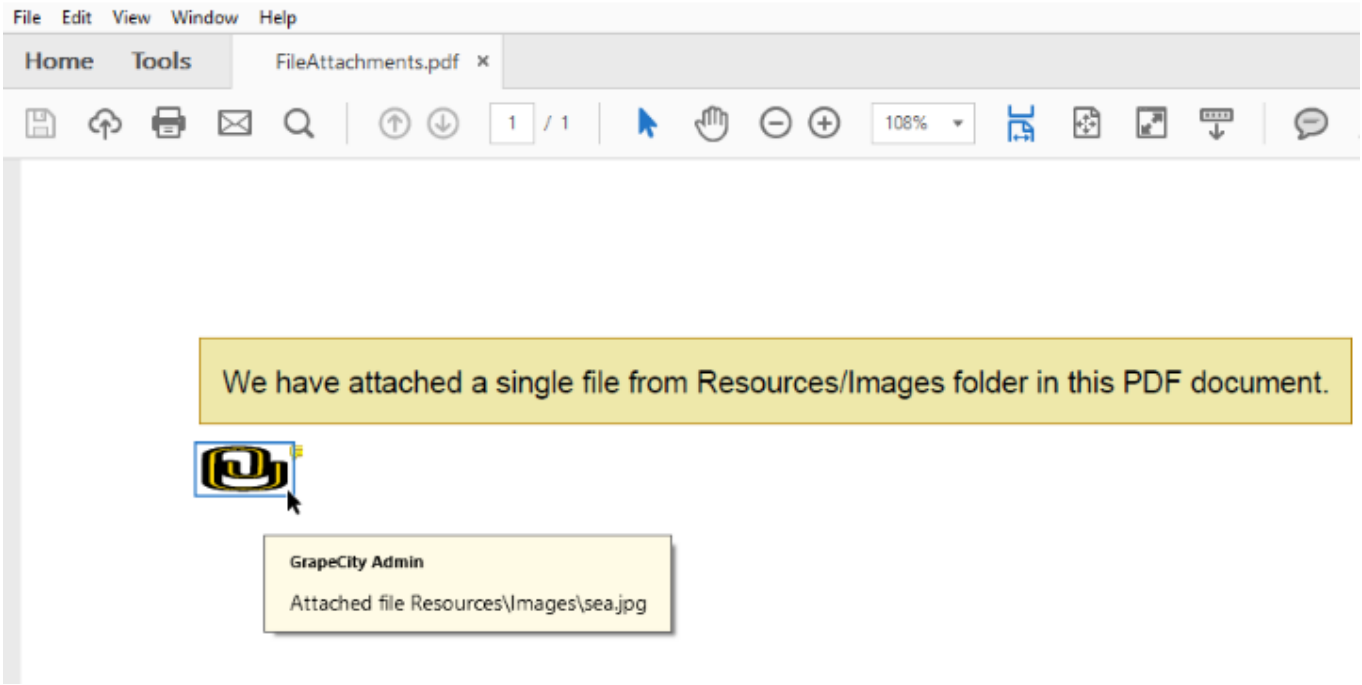
```
GcPdfDocument doc = new GcPdfDocument();
    Page page = doc.NewPage();

    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. DsPdf 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 = "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

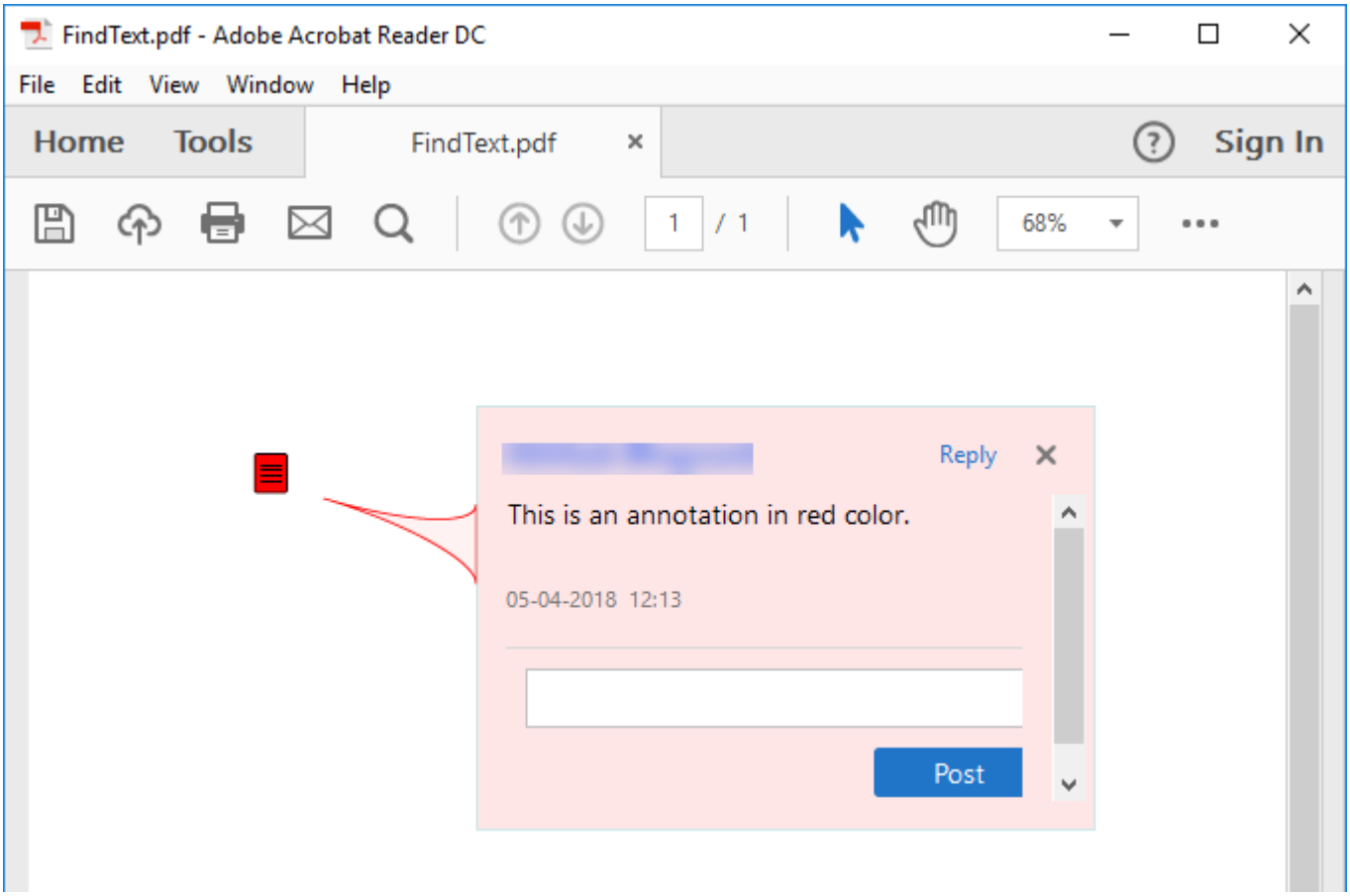
For more information about how to work with file attachments using DsPdf, see DsPdf sample browser.

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).

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

All the listed annotations have a dedicated class and properties in the DsPdf library which makes it easier to implement different annotations. DsPdf 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

DsPdf 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

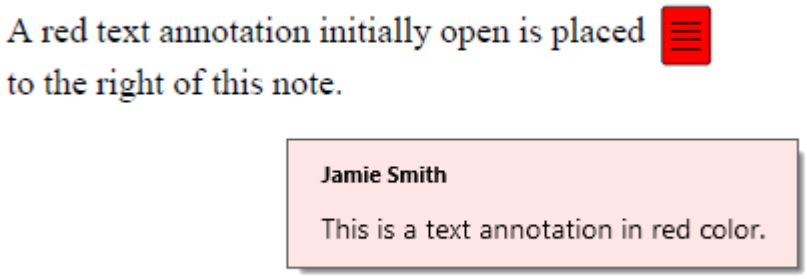
For more information about how to implement annotations using DsPdf, see DsPdf sample browser.

Annotation Types

DsPdf 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. DsPdf provides TextAnnotation class to enable the users to apply text annotations in the PDF document.



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

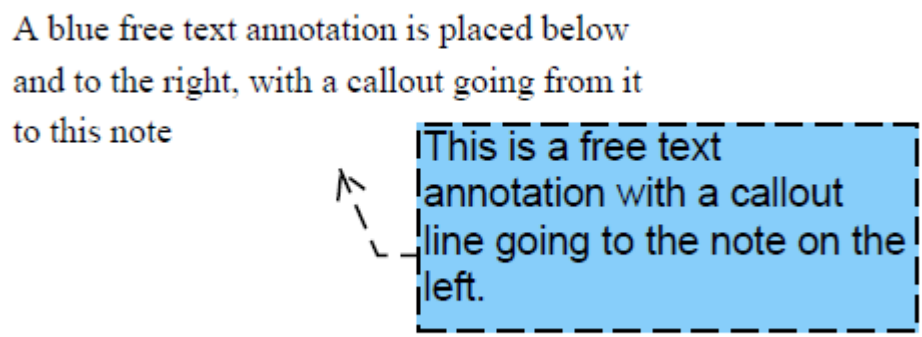
C#

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

    //Create an instance of TextAnnotation class and set its relevant properties
    var textAnnot = new TextAnnotation()
    {
        UserName = "Jamie Smith",
        Contents = "This is a text annotation in red color.",
        PdfRect = 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. DsPdf provides FreeTextAnnotation class to enable the users to apply free text annotations to the PDF file.



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

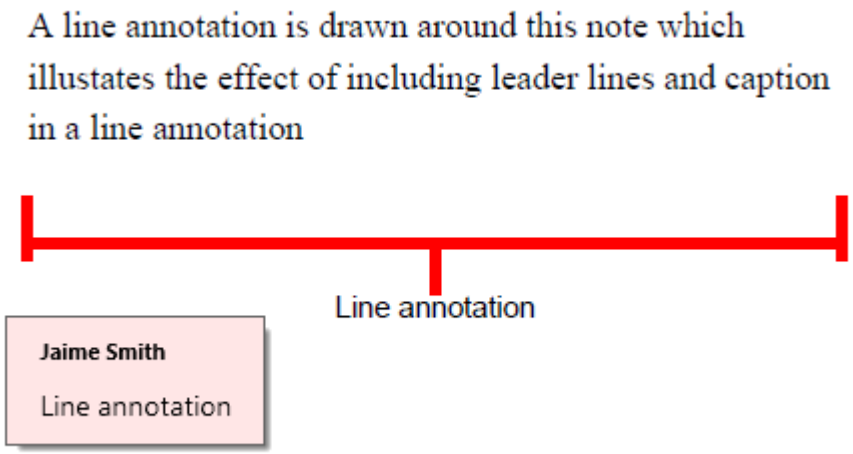
C#

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

    //Create an instance of FreeTextAnnotation class and set its relevant properties
    var freeAnnot = new FreeTextAnnotation()
    {
        PdfRect = 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. DsPdf provides LineAnnotation class to enable the users to apply line annotations to the PDF file.



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

C#

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

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

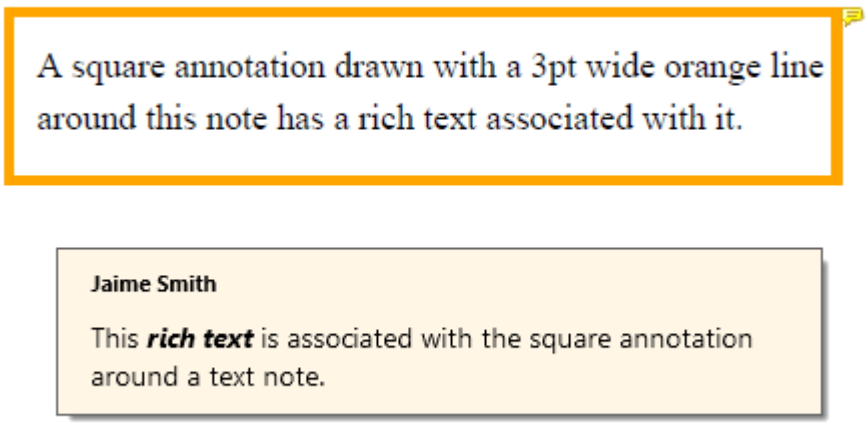
```
        TextPosition = LineAnnotationTextPosition.Inline,
    };

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

Back to Top

Square Annotation

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

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



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

C#

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

    //Create an instance of SquareAnnotation class and set its relevant properties
    var squareAnnot = new SquareAnnotation()
```

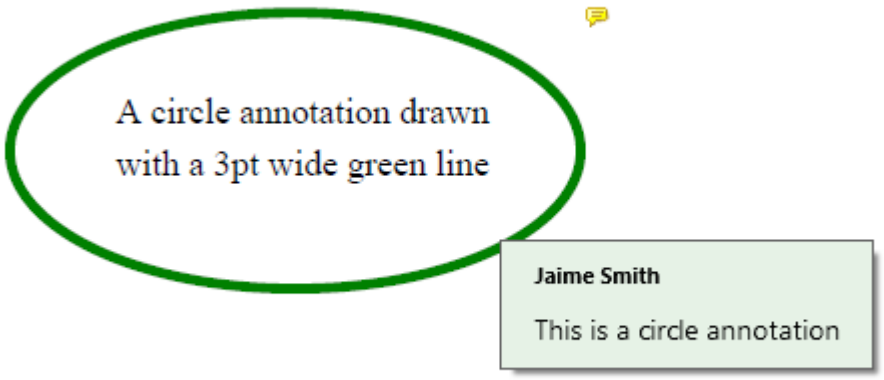
```
    {
        UserName = "Jaime Smith",
        PdfRect = rc,
        LineWidth = 3,
        Color = Color.Orange,
        RichText =
  "<body><p>This <b><i>rich text</i></b> is associated with the square annotation
around a text note.</p></body>"
    };
    page.Annotations.Add(squareAnnot); //Add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

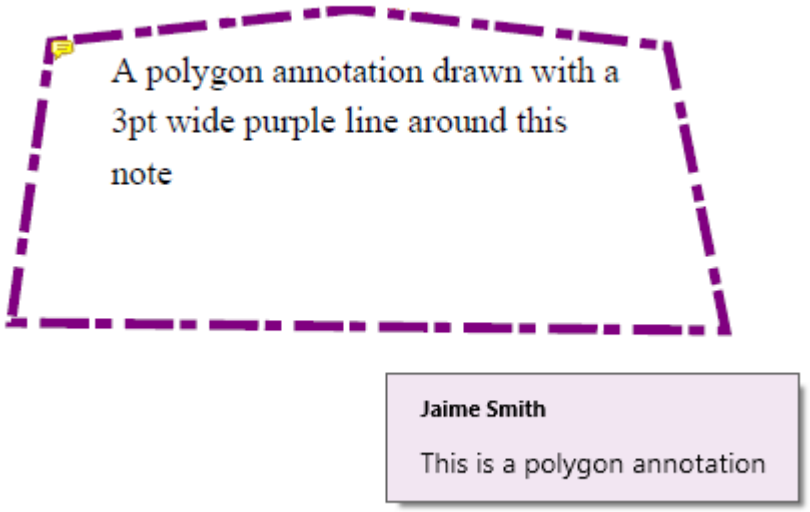
```
public void CreateCircleAnnotation()
{
    GcPdfDocument doc = new GcPdfDocument();
    Page page = doc.NewPage();
    RectangleF rc = new RectangleF(50, 50, 120, 50);
    page.Graphics.DrawString("A circle annotation drawn with a 3pt wide green line",
        new TextFormat() { Font = StandardFonts.Times,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

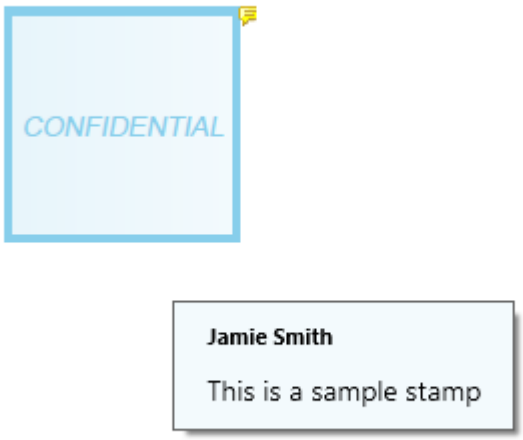
```
public void CreatePolygonAnnotation()
{
    GcPdfDocument doc = new GcPdfDocument();
    Page page = doc.NewPage();
    RectangleF rc = new RectangleF(140, 30, 160, 70);
    page.Graphics.DrawString("A polygon annotation drawn with a 3pt wide purple line
around this note",
```

```
        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. DsPdf 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.

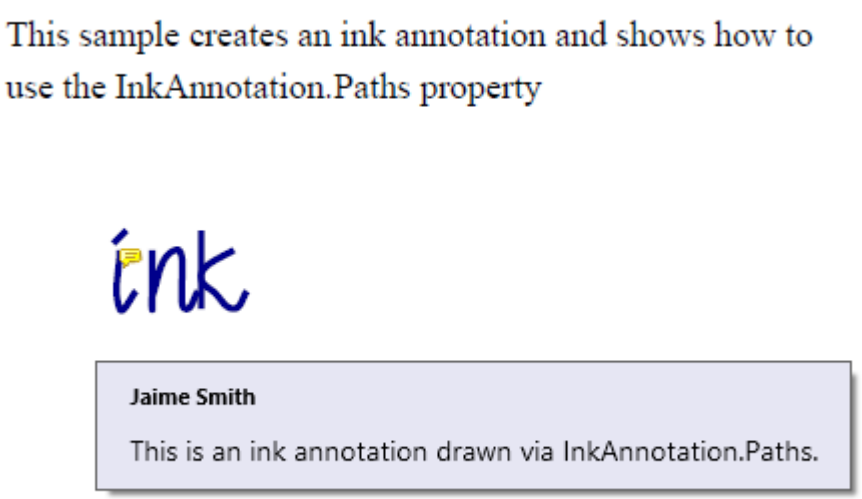
C#

```
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,
        PdfRect = 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. DsPdf provides InkAnnotation class to enable the users to apply ink annotations to the PDF file.



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

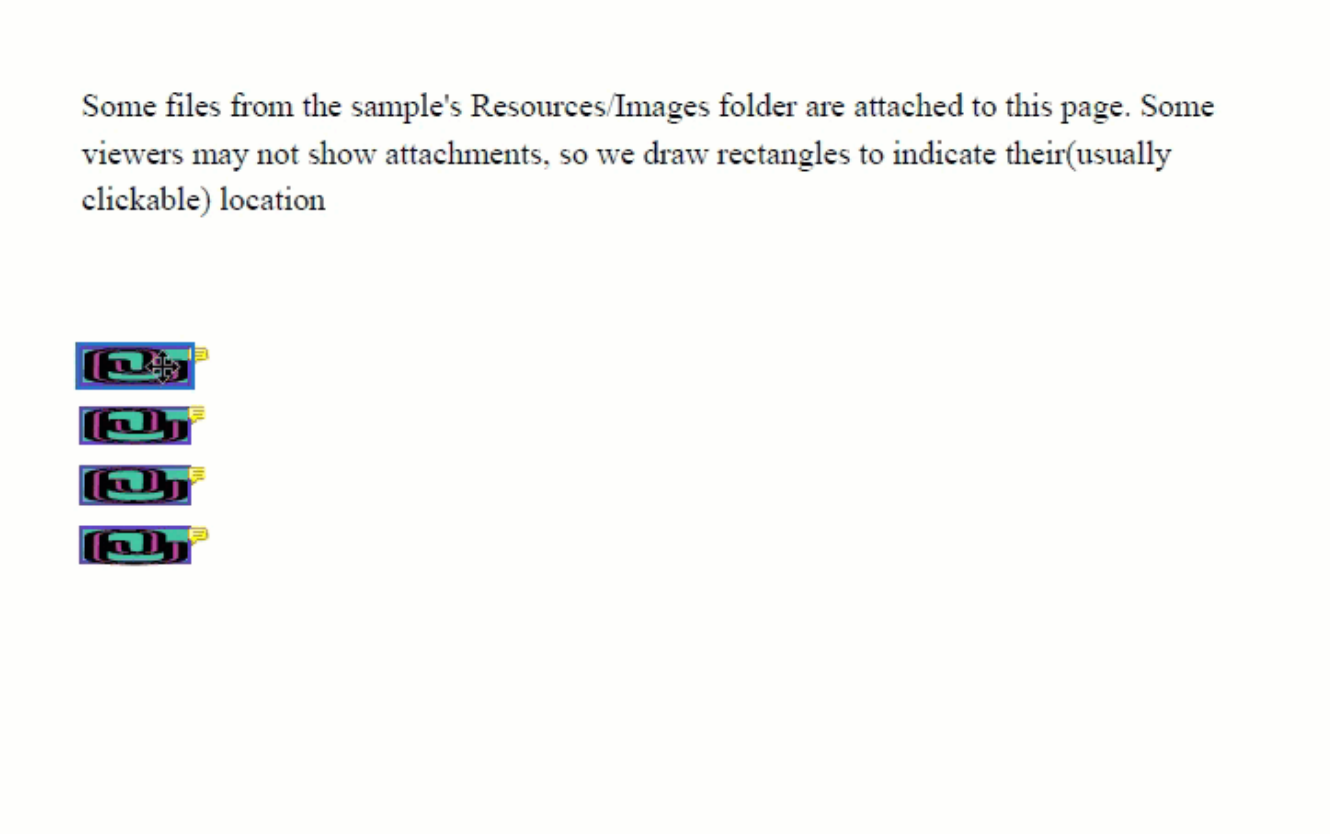
C#

```
public void CreateInkAnnotation()
{
    var doc = new GcPdfDocument();
    var page = doc.NewPage();
    RectangleF rc = new RectangleF(50, 50, 250, 50);
    page.Graphics.DrawString("This sample creates an ink annotation and shows how to
use the " +
        "InkAnnotation.Paths property",
        new TextFormat() { Font = StandardFonts.Times, FontSize = 11 }, rc);

    //Create an instance of InkAnnotation class and set its relevant properties
    var inkAnnot = new InkAnnotation()
    {
        UserName = "Jaime Smith",
        PdfRect = 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
    List<PointF[]> paths = new List<PointF[]>();
    paths.Add(new[] { new PointF(x + w / 2, y), new PointF(x + w / 2, y + h),
                      new PointF(x + w, y + h * .7f) });
    paths.Add(new[] { new PointF(x + w / 2, y), new PointF(x + w / 2, y + h),
                      new PointF(x + w, y + h * .7f) });
    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;
    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;
    paths.Add(new[] { new PointF(x, y - h / 3), new PointF(x, y + h) });
    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) });
    inkAnnot.Paths = paths;

    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. DsPdf provides FileAttachmentAnnotation class to enable the users to apply file attachment annotations to the PDF file.



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

C#

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

    var ip = new PointF(rc.X, rc.Bottom + 9);
    var attSize = new SizeF(36, 12);
    var gap = 8;
```

```
    string[] files = new string[]
    {
        "tudor.jpg",
        "sea.jpg",
        "puffins.jpg",
        "lavender.jpg",
    };
    foreach (string fn in files)
    {
        string file = Path.Combine("Resources", "Images", fn);
        //Create an instance of FileAttachmentAnnotation class and set its relevant
properties
        FileAttachmentAnnotation faa = new FileAttachmentAnnotation()
        {
            Color = Color.FromArgb(unchecked((int)0xFFc540a5)),
            UserName = "Jaime Smith",
            PdfRect = 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

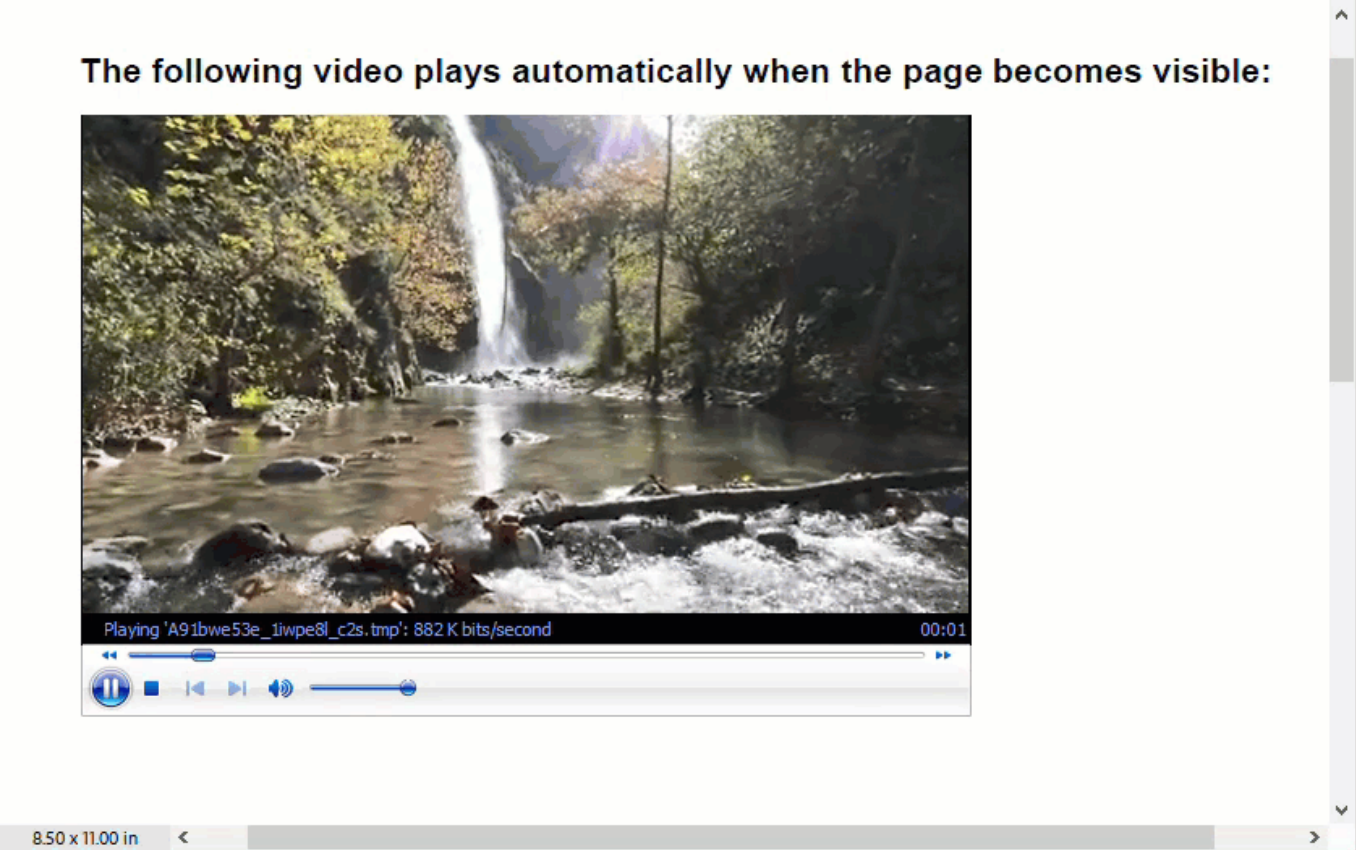
RichMedia Annotation

RichMedia annotation is a reference to the media resources (audio and video) embedded in a PDF document. It will appear as a rectangular box with a play button inside. To add RichMedia annotations to the PDF document, DsPdf provides SetVideo, SetAudio, and SetContent methods of RichMediaAnnotation class.

DsPdf also provides helper classes RichMediaAnnotationActivation, RichMediaAnnotationDeactivation, and RichMediaAnnotationPresentationStyle with constants that define the following properties of RichMediaAnnotation class:

Property	Constants	Description
ActivationCondition	UserAction	The annotation is activated explicitly by a user action or script.
	PageReceivesFocus	The annotation activates as soon as the page that contains it receives focus as the current page.
	PageBecomesVisible	The annotation activates as soon as any part of the page that contains the annotation becomes visible.
DeactivationCondition	UserAction	The annotation is deactivated explicitly by a user action or script.
	PageLosesFocus	The annotation deactivates as soon as the page that contains it loses

Property	Constants	Description
		focus as the current page.
	PageBecomesInvisible	The annotation deactivates as soon as the entire page that contains the annotation is no longer visible.
PresentationStyle	Embedded	The media plays inside the viewer.
	Windowed	The media plays in a separate window.



Refer to the following example code to add a video using RichMediaAnnotation:

C#

```
// Initialize GcPdfDocument.
var doc = new GcPdfDocument();

// Add a blank page to the PDF document.
var page = doc.NewPage();

// Initialize GcPdfGraphics.
var g = page.Graphics;

// Provide video path.
var videoPath = Path.Combine("waterfall.mp4");

const int smallGap = 18;

// Set text format.
```

```
var tf = new TextFormat()
{
    Font = StandardFonts.HelveticaBold,
    FontSize = 14
};

// Add a video that plays when the page becomes visible.
var rc = new RectangleF(new PointF(60, 50), new SizeF(500, 10));

// Draw the introductory string.
g.DrawString("The following video plays automatically when the page becomes
visible:", tf, rc);

// Initialize RichMediaAnnotation and set its properties.
var rma = new RichMediaAnnotation();
var videoEfs = EmbeddedFileStream.FromFile(doc, videoPath);
var videoFileSpec = FileSpecification.FromEmbeddedStream(Path.GetFileName(videoPath),
videoEfs);

// Set the media for the video.
rma.SetVideo(videoFileSpec);

// Set presentation style.
rma.PresentationStyle = RichMediaAnnotationPresentationStyle.Embedded;

// Set activation condition.
rma.ActivationCondition = RichMediaAnnotationActivation.PageBecomesVisible;

// Set deactivation condition.
rma.DeactivationCondition = RichMediaAnnotationDeactivation.PageBecomesInvisible;

// Set behavior of navigation pane.
rma.ShowNavigationPane = true;

// Set page to which RichMediaAnnotation will be added.
rma.Page = page;

// Set rectangle in which RichMediaAnnotation will be added.
rma.Rect = new RectangleF(rc.X, rc.Bottom + smallGap, 72 * 5, 72 * 5 * 0.56f);

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

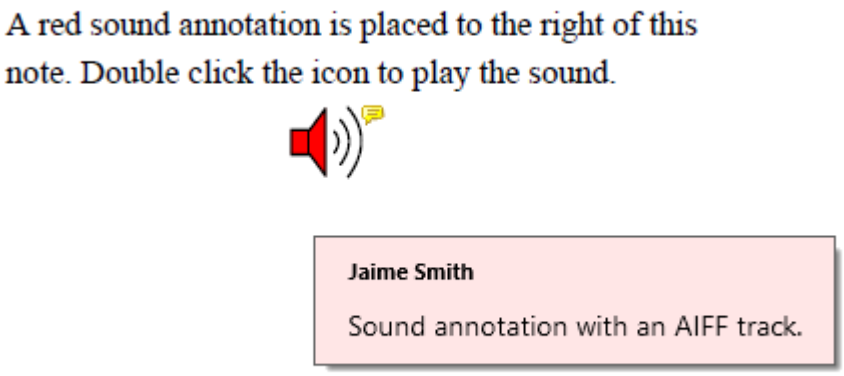
User can also fetch the media from the RichMedia annotation using GetContent method of RichMediaAnnotation class.

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. DsPdf provides SoundAnnotation

class to enable the users to apply sound annotations to the PDF file.



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

C#

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

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

Back to Top

Widget Annotation

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

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

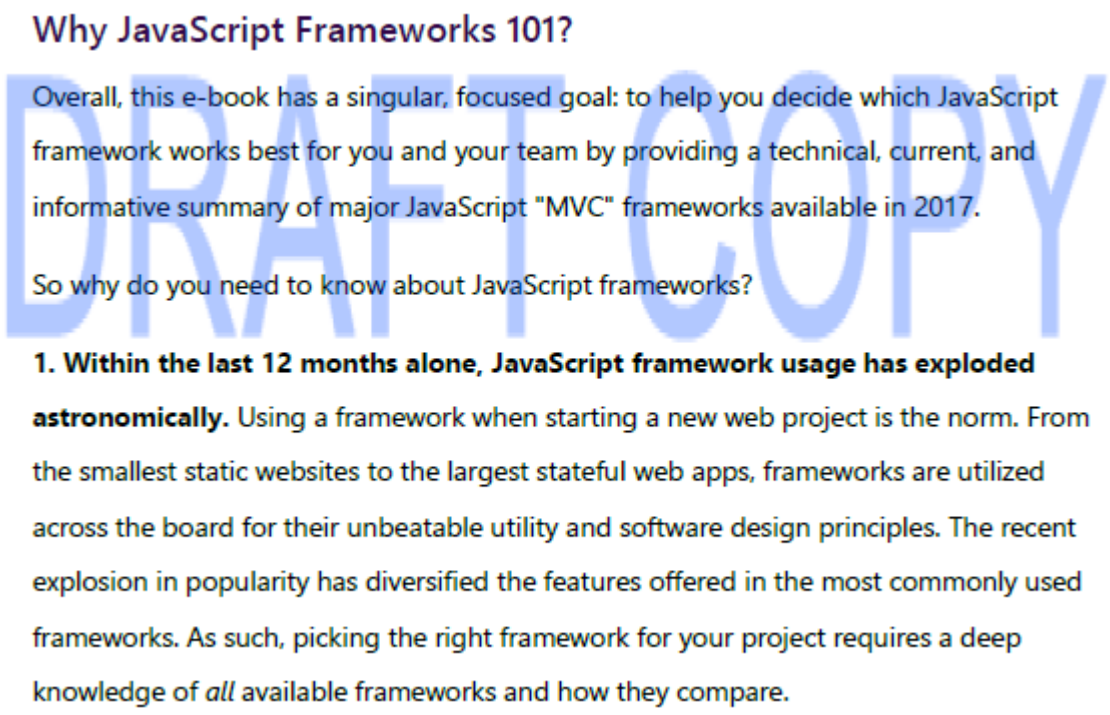
C#

```
public void CreateWidgetAnnotation()
{
    var doc = new GcPdfDocument();
    var page = doc.NewPage();
    var g = page.Graphics;
    TextFormat tf = new TextFormat();
    tf.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.PdfRect = 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. DsPdf provides WatermarkAnnotation class to enable the users to apply watermark annotations to the PDF file.



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

C#

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

    //Loop over pages, add a watermark to each page:
    foreach (var page in doc.Pages)
    {
        //Create an instance of WatermarkAnnotation class and set its relevant
properties
        var watermark = new WatermarkAnnotation()
        {
            PdfRect = 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 DsPdf 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. DsPdf 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
    List<Quadrilateral> area = new List<Quadrilateral>();
    area.Add(l[0].Bounds[0]);
    redactAnnotation.Area = area;
    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.

DsPdf 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#

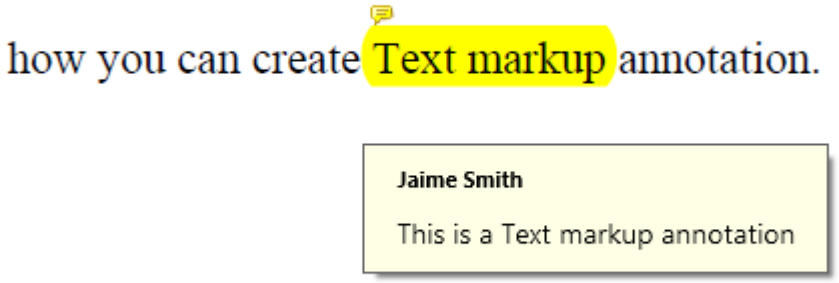
```
var doc = new GcPdfDocument();
using (var fs = new FileStream(Path.Combine("Resources", "PDFs",
"TimeSheet_Redacted.pdf"), FileMode.Open, FileAccess.Read))
{
    // Load the PDF containing redact annotations (areas marked for redaction)
    doc.Load(fs);

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

    var redact = new RedactAnnotation()
    {
        PdfRect = 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. DsPdf provides TextMarkupAnnotation class to enable the users to apply text markup annotations to the PDF file.



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

C#

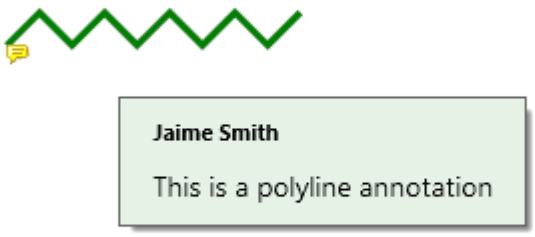
```
public void CreateTextMarkupAnnotation()
{
    GcPdfDocument doc = new GcPdfDocument();
    var page = doc.NewPage();
    var tl = page.Graphics.CreateTextLayout();
    tl.DefaultFormat.Font = StandardFonts.Times;
    tl.DefaultFormat.FontSize = 14;
    tl.Append("This sample demonstrates how you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

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

```
    textMarkupAnnot.Contents = "This is a Text markup annotation";
    //search the text(e.g employee name) which needs to be redacted
    var found = doc.FindText(new FindTextParams("Text markup", true, false), null);
    List<Quadrilateral> area = new List<Quadrilateral>();
    foreach (var f in found)
        area.Add(f.Bounds[0]);
    textMarkupAnnot.Area = area;
    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. DsPdf 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#

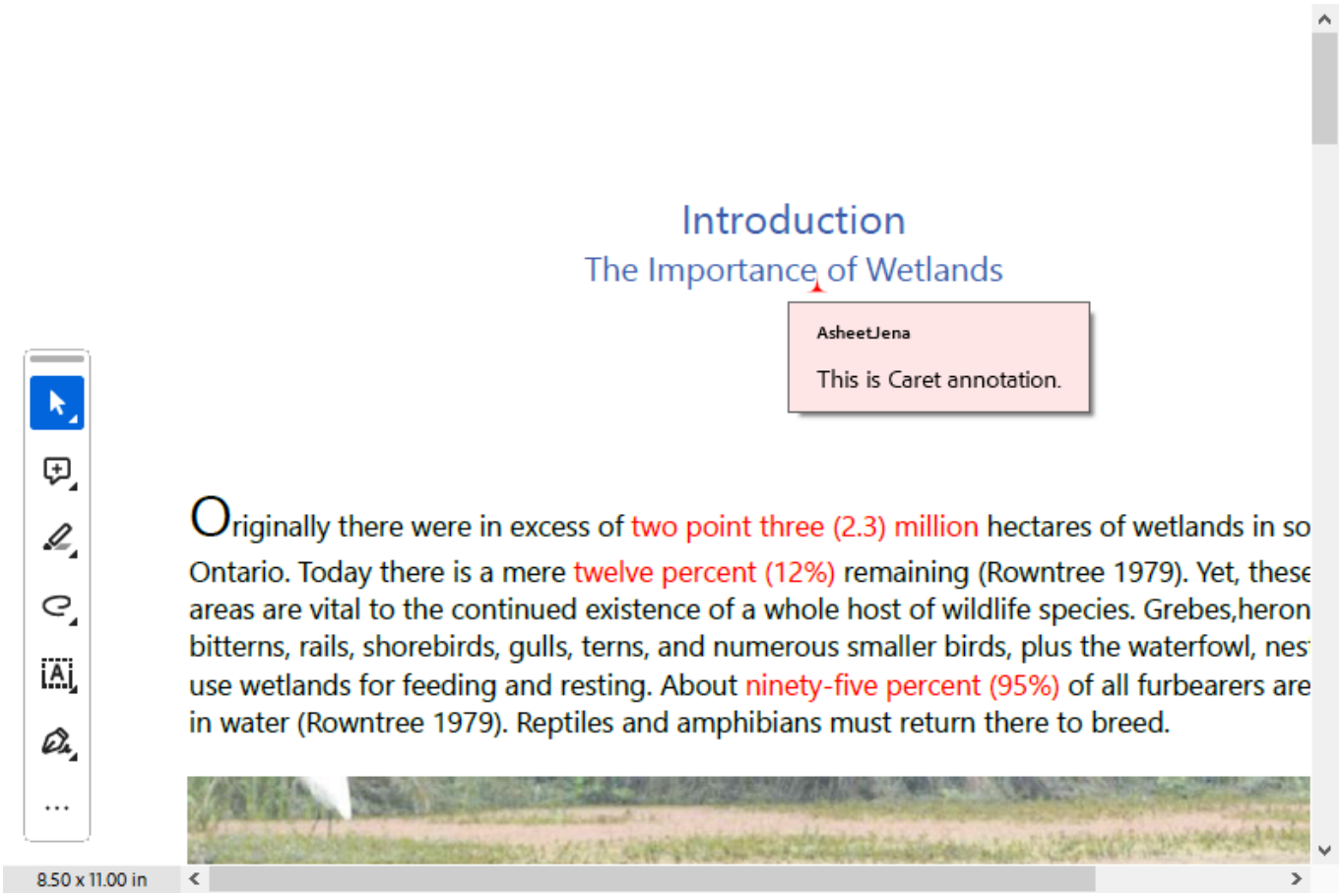
```
public void CreatePolyLineAnnotation()
{
    GcPdfDocument doc = new GcPdfDocument();
    Page page = doc.NewPage();
    RectangleF rc = new RectangleF(50, 50, 400, 40);
    page.Graphics.DrawString("This sample demonstrates how you can create a polyline
annotation.",
        new TextFormat() { Font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

Caret Annotation

Caret annotation is a visual symbol typically used to mark text for some changes or to indicate some missing text. DsPdf provides CaretAnnotation class that allows a user to add Caret annotations to the PDF file.



Refer to the following example code to add a caret annotation to a PDF document:

C#

```
// Initialize GcPdfDocument.
GcPdfDocument doc = new GcPdfDocument();

// Create a file stream.
FileStream fs = new FileStream("Wetlands.pdf", FileMode.Open, FileAccess.Read);

// Load the file stream.
doc.Load(fs);

// Get the first page.
var page = doc.Pages[0];

// Get the text map.
var tm = page.GetTextMap();

// Insert the CaretAnnotation after "The Importance" text.
tm.FindText(new FindTextParams("The Importance", false, true), (fp_) =>
{
    // r is bounds of the found text.
    var r = fp_.Bounds[0].ToRect();
    // Create the CaretAnnotation and add it to the page.
    CaretAnnotation ca = new CaretAnnotation();
```

```
    ca.Page = page;
    // in this code annotation size is hardcoded, you can make a code
    // which will adjust size of the annotation depending on height of the found text
fragment
    ca.Rect = new System.Drawing.RectangleF(r.Right - 4, r.Bottom - 8, 8, 8);
    ca.Opacity = 1f;
    ca.Color = Color.Red;
    ca.Contents = "This is Caret annotation.";
});

doc.Save("CaretAnnotation.pdf");
```

Back to Top

For more information about how to work with annotations using DsPdf, see DsPdf sample browser.

Appearance 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.

DsPdf 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.

However, to handle annotations generating rich text such as FreeTextAnnotation and LineAnnotation, the GcPdfDocument class provides a boolean property BuildRichTextAppearanceStreams which is set to **false**, by default. For such annotations, DsPdf does not generate appearance streams for them by default. When the BuildRichTextAppearanceStreams property is set to **true**, the DsPdf library generates appearance streams for such annotations and renders the rich text of these annotations as plain text.

In addition, the AnnotationBase class provides RemoveAppearance**()** method to remove all appearance streams associated with the annotations. The method disables the generation of appearance streams when the document is saved. This helps in reducing the file size of exported PDF document.

The following code illustrates how to add an appearance stream to a stamp annotation in a PDF document and then, shows how to remove the same.

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;
        }

        // Remove appearance stream for first annotation of the first page
        var p = doc.Pages[0];
        var an = p.Annotations[0];
        an.RemoveAppearance();
        doc.Save(stream);
    }
}
```

The following code illustrates setting BuildRichTextAppearanceStreams to true for FreeTextAnnotation that generates rich text:

C#

```
GcPdfDocument.BuildRichTextAppearanceStreams = true;
var doc = new GcPdfDocument();
var page = doc.NewPage();

// Another free text annotation, with some rich text inside:
var freeRichAnnot = new FreeTextAnnotation()
```

```
{
    Rect = new RectangleF(10, 50, 288, 72),
    LineWidth = 1,
    Color = Color.LightSalmon,
    RichText =
    "<body><p>This is another <i>free text annotation</i>, with <b><i>Rich Text</i>
</b> content.</p>" +
    "<p><br />Even though a <b>free text</b> annotation displays text directly on a
page, " +
    "as other annotations it can be placed outside the page's bounds.</p></body>"
};
page.Annotations.Add(freeRichAnnot);
// Done:
doc.Save("Annotations.pdf");
```

Document

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

DsPdf provides following document properties through GcPdfDocument class:

Compression

DsPdf 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

DsPdf 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

DsPdf 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

DsPdf 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

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

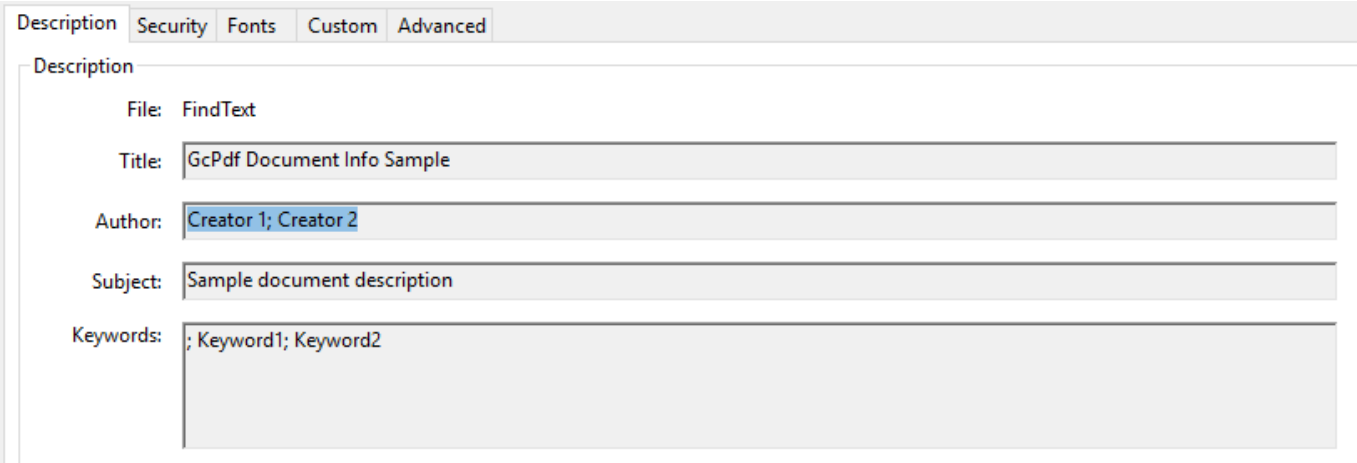
Pdf Version

DsPdf 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

DsPdf 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 = "Document Info Sample";
    doc.DocumentInfo.Author = "John Doe";
    doc.DocumentInfo.Subject = "DsPdfDocument.DocumentInfo";
    doc.DocumentInfo.Producer = "DsPdfWeb Producer";
    doc.DocumentInfo.Creator = "DsPdfWeb 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 = "MESCIUS, 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 DsPdfWeb";
    // 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

Optimize Document Size

DsPdf allows you to reduce the size of a document efficiently using RemoveDuplicateImages method of GcPdfDocument class. This method eliminates redundant instances of identical images internally within the document, retaining only a single instance across multiple locations, hence reducing the size of the document.

Refer to the following example code demonstrating how to optimize the file size of a document using RemoveDuplicateImages method:

C#

```
// Initialize GcPdfDocument.
GcPdfDocument doc = new GcPdfDocument();

// Open PDF document in the file stream.
FileStream fs = File.OpenRead("Invoice.pdf");

// Load the PDF document.
```

```
doc.Load(fs);

// Remove duplicate images.
doc.RemoveDuplicateImages();

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

Note: You can also optimize the file size of a merged PDF document. See **Remove Duplicate Images from Merged Document**.

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.

Refer to the following example code to append one PDF document to 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");
```

Note: DsPdf also provides **ClonePage** method that can be used to clone a particular page from a specified index in the PDF file and insert it into a specified index of the same PDF file. For more information on cloning a page, see Clone a Page.

Remove Duplicate Images from Merged Document

DsPdf also allows you to remove duplicate instances of identical images when merging PDF documents using RemoveDuplicateImages property of MergeDocumentOptions class. Removing the duplicate instances helps reduce the size of the merged PDF document. The default value of RemoveDuplicateImages property is false.

RemoveDuplicateImages property calls RemoveDuplicateImages method, which scans for duplicate instances during the merging, hence affecting the merge performance. To avoid this, you can merge the documents first and then

remove the duplicate instances of the same images using RemoveDuplicateImages method. For more information, see **Optimize Document Size**.

Refer to the following example code to remove duplicate instances of the same images from a merged PDF document using RemoveDuplicateImages method:

C#

```
// Initialize GcPdfDocument for first PDF document.
GcPdfDocument doc1 = new GcPdfDocument();

// Open first PDF document in the file stream.
FileStream fs1 = File.OpenRead("Invoice_Jan.pdf");

// Load the first PDF document.
doc1.Load(fs1);

// Initialize GcPdfDocument for second PDF document.
GcPdfDocument doc2 = new GcPdfDocument();

// Open second PDF document in the file stream.
FileStream fs2 = File.OpenRead("Invoice_Feb.pdf");

// Load the second PDF document.
doc2.Load(fs2);

// Merge PDF document into current document.
doc1.MergeWithDocument(doc2);

// Remove duplicate images.
doc1.RemoveDuplicateImages();

// Save PDF document.
doc1.Save("Merged.pdf");
```

Back to Top

Apply Redact Annotation

DsPdf 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. DsPdf 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, DsPdf 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. DsPdf 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

DsPdf 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

DsPdf 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 DsPdf, see DsPdf sample browser.

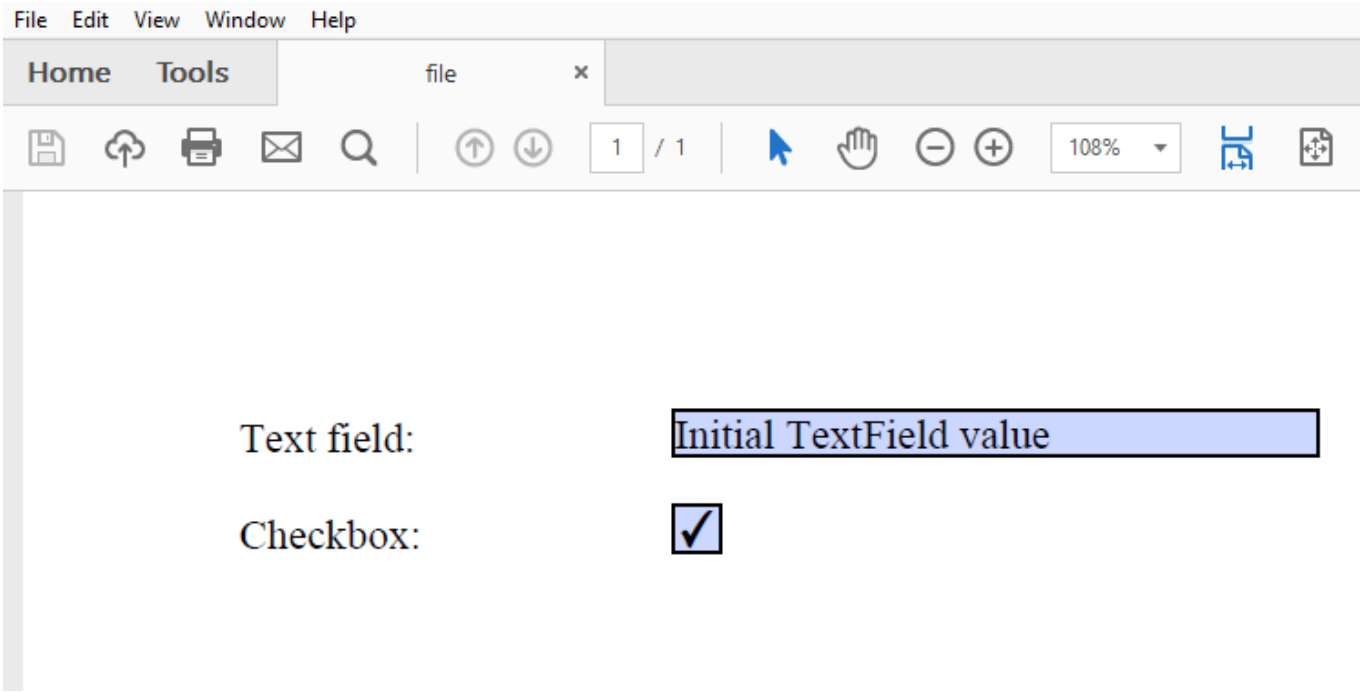
Back to Top

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).

DsPdf 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 DsPdf:

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

Set Field Format

You can set the format of TextField, CombTextField, and ComboBoxField to percent, number, date, time, and special format using the following methods in TextField, CombTextField, and ComboBoxField classes:

Methods	Description
SetPercentFormat	Initializes the field's Format and KeyPress events with ActionJavaScript objects corresponding to the specified format. This method does not affect the field value or appearance streams of the field annotations.
SetNumberFormat	Initializes the field's Format and KeyPress events with ActionJavaScript objects corresponding to the specified format. This method does not affect the field value or appearance streams of the field annotations.
SetDateFormat	Initializes the field's Format and KeyPress events with ActionJavaScript objects corresponding to the specified format. This method does not affect the field value or appearance streams of the field annotations.
SetTimeFormat	Initializes the field's Format and KeyPress events with ActionJavaScript objects corresponding to the specified format. This method does not affect the field value or appearance streams of the field annotations.
SetSpecialFormat	Initializes the field's Format and KeyPress events with ActionJavaScript objects corresponding to the specified format. This method does not affect the field value or appearance streams of the field annotations.
SetPercentValue	Initializes the field's Format and KeyPress events, sets the field value, and generates the appearance stream for field annotations. The passed value will be converted to a string using the corresponding format.
SetNumberValue	Initializes the field's Format and KeyPress events, sets the field value, and generates the appearance stream for field annotations. The passed value will be converted to a string using the corresponding format.
SetDateValue	Initializes the field's Format and KeyPress events, sets the field value, and generates the appearance stream for field annotations. The passed value will be converted to a string using the corresponding format.
SetTimeValue	Initializes the field's Format and KeyPress events, sets the field value, and generates the appearance stream for field annotations. The passed value will be converted to a string using the corresponding format.
SetSpecialFormatValue	Initializes the field's Format and KeyPress events, sets the field value, and generates the appearance stream for field annotations. The passed value will be converted to a string using the corresponding format.

Refer to the following example code to set formats for text fields, comb text fields, and combo box fields:

C#

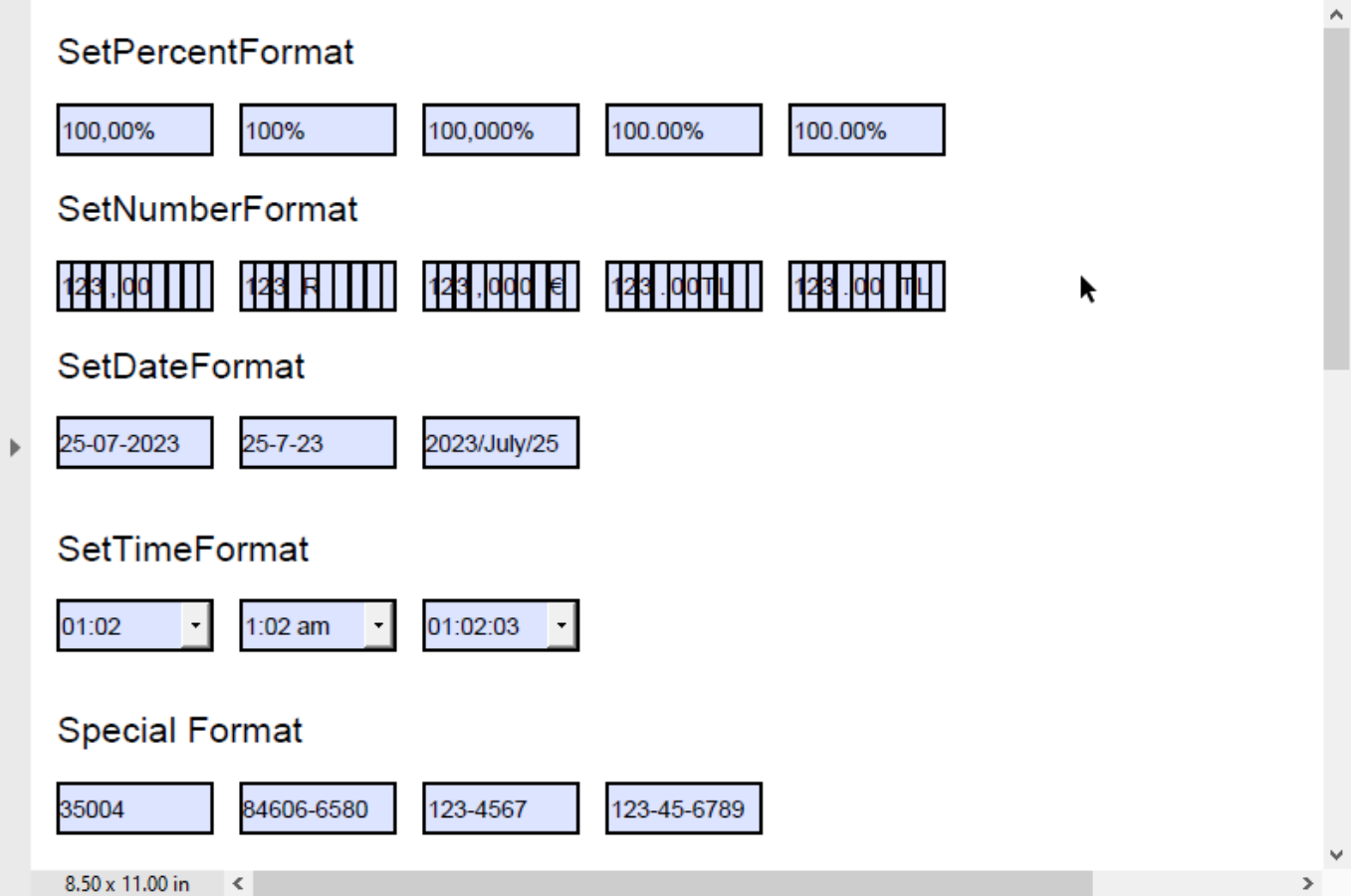
```
// Set percent format for TextField.
private static TextField AddPercentFormat(Page p,
    _Rect rect,
    int decimalPlaces,
    TextField.NumberSeparatorStyle separatorStyle)
{
    TextField result = new TextField();
    p.Doc.AcroForm.Fields.Add(result);
```

```
    result.Widget.Page = p;
    result.Widget.Rect = rect;
    result.Widget.Border.Width = 1;
    result.SetPercentFormat(decimalPlaces, separatorStyle);

    return result;
}

// Set number format for CombTextField.
private static CombTextField AddNumberFormat(Page p,
    _Rect rect,
    int decimalPlaces,
    CombTextField.NumberSeparatorStyle separatorStyle,
    CombTextField.NumberNegativeStyle negativeStyle,
    string currencySymbol,
    CombTextField.CurrencySymbolStyle currencySymbolStyle)
{
    CombTextField result = new CombTextField();
    p.Doc.AcroForm.Fields.Add(result);

    result.Widget.Page = p;
    result.Widget.Rect = rect;
    result.Widget.Border.Width = 1;
    result.SetNumberFormat(decimalPlaces, separatorStyle, negativeStyle,
currencySymbol, currencySymbolStyle);

    return result;
}

// Set date format for TextField.
private static TextField AddDateFormat(Page p,
    _Rect rect,
    DateTime v,
    string format)
{
    TextField result = new TextField();
    p.Doc.AcroForm.Fields.Add(result);

    result.Widget.Page = p;
    result.Widget.Rect = rect;
    result.Widget.Border.Width = 1;
    result.SetDateValue(v, format);

    return result;
}

// Set time format for ComboBoxField.
private static ComboBoxField AddTimeFormat(Page p,
    _Rect rect,
    DateTime v,
```

```
    string format)
{
    ComboBoxField result = new ComboBoxField();
    p.Doc.AcroForm.Fields.Add(result);

    result.Widget.Page = p;
    result.Widget.Rect = rect;
    result.Widget.Border.Width = 1;
    result.SetTimeValue(v, format);

    return result;
}

// Set special format for TextField.
private static TextField AddSpecialFormat(Page p,
    _Rect rect,
    string v,
    TextField.SpecialFormat format)
{
    TextField result = new TextField();
    p.Doc.AcroForm.Fields.Add(result);

    result.Widget.Page = p;
    result.Widget.Rect = rect;
    result.Widget.Border.Width = 1;
    result.SetSpecialFormatValue(v, format);

    return result;
}

static void Main(string[] args)
{
    // Initialize GcPdfDocument.
    GcPdfDocument doc = new GcPdfDocument();

    // Set compression level.
    doc.CompressionLevel = System.IO.Compression.CompressionLevel.NoCompression;

    // Add a blank page to the document.
    var p = doc.NewPage();

    // Initialize GcPdfGraphics.
    var g = p.Graphics;

    // Draw SetPercentFormat string.
    g.DrawString("SetPercentFormat", StandardFonts.Helvetica, 14, Color.Black, new
_Point(10, 10));

    // Set percent format for the text fields.
    AddPercentFormat(p, new _Rect(10, 40, 60, 20), 2,
TextField.NumberSeparatorStyle.Comma);
```

```
    AddPercentFormat(p, new _Rect(80, 40, 60, 20), 0,
TextField.NumberSeparatorStyle.CommaDot);
    AddPercentFormat(p, new _Rect(150, 40, 60, 20), 3,
TextField.NumberSeparatorStyle.DotComma);
    AddPercentFormat(p, new _Rect(220, 40, 60, 20), 2,
TextField.NumberSeparatorStyle.Dot);
    AddPercentFormat(p, new _Rect(290, 40, 60, 20), 2,
TextField.NumberSeparatorStyle.ApostropheDot);

    // Draw SetNumberFormat string.
    g.DrawString("SetNumberFormat", StandardFonts.Helvetica, 14, Color.Black, new
_Point(10, 70));

    // Set number format for the comb text fields.
    AddNumberFormat(p, new _Rect(10, 100, 60, 20), 2,
CombTextField.NumberSeparatorStyle.Comma,
        CombTextField.NumberNegativeStyle.None, null,
CombTextField.CurrencySymbolStyle.BeforeWithSpace);
    AddNumberFormat(p, new _Rect(80, 100, 60, 20), 0,
CombTextField.NumberSeparatorStyle.CommaDot,
        CombTextField.NumberNegativeStyle.UseRedText, "R",
CombTextField.CurrencySymbolStyle.BeforeWithSpace);
    AddNumberFormat(p, new _Rect(150, 100, 60, 20), 3,
CombTextField.NumberSeparatorStyle.DotComma,
        CombTextField.NumberNegativeStyle.ShowParentheses, "\u20ac",
CombTextField.CurrencySymbolStyle.AfterWithSpace);
    AddNumberFormat(p, new _Rect(220, 100, 60, 20), 2,
CombTextField.NumberSeparatorStyle.Dot,
        CombTextField.NumberNegativeStyle.ShowParentheses, "TL",
CombTextField.CurrencySymbolStyle.AfterNoSpace);
    AddNumberFormat(p, new _Rect(290, 100, 60, 20), 2,
CombTextField.NumberSeparatorStyle.ApostropheDot,
        CombTextField.NumberNegativeStyle.ShowParentheses |
CombTextField.NumberNegativeStyle.UseRedText, "TL",
CombTextField.CurrencySymbolStyle.AfterWithSpace);

    // Draw SetDateFormat string.
    g.DrawString("SetDateFormat", StandardFonts.Helvetica, 14, Color.Black, new
_Point(10, 130));

    // Set date format for the text fields.
    AddDateFormat(p, new _Rect(10, 160, 60, 20), DateTime.Now, "dd-mm-yyyy");
    AddDateFormat(p, new _Rect(80, 160, 60, 20), DateTime.Now, "d-m-yy");
    AddDateFormat(p, new _Rect(150, 160, 60, 20), DateTime.Now, "yyyy/mmmm/dd");

    // Draw SetTimeFormat string.
    g.DrawString("SetTimeFormat", StandardFonts.Helvetica, 14, Color.Black, new
_Point(10, 200));

    // Set time format for the combobox fields.
    DateTime dt = new DateTime(2000, 1, 1, 1, 2, 3);
```

```
    AddTimeFormat(p, new _Rect(10, 230, 60, 20), dt, "HH:MM");
    AddTimeFormat(p, new _Rect(80, 230, 60, 20), dt, "h:MM tt");
    AddTimeFormat(p, new _Rect(150, 230, 60, 20), dt, "HH:MM:ss");

    // Draw Special Format string.
    g.DrawString("Special Format", StandardFonts.Helvetica, 14, Color.Black, new
_Point(10, 270));

    // Set special format for the text fields.
    AddSpecialFormat(p, new _Rect(10, 300, 60, 20), "35004",
TextField.SpecialFormat.ZipCode);
    AddSpecialFormat(p, new _Rect(80, 300, 60, 20), "84606-6580",
TextField.SpecialFormat.ZipCode4);
    AddSpecialFormat(p, new _Rect(150, 300, 60, 20), "",
TextField.SpecialFormat.Phone);
    AddSpecialFormat(p, new _Rect(220, 300, 60, 20), "",
TextField.SpecialFormat.SSN);

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



Note: If any field property that affects appearance is changed, then appearance streams generated by SetXXXMethods are lost, so these methods should be called "at the end".

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 DsPdfViewer. Refer Form Editor for more details.

Back to Top

For more information about implementation of AcroForms using DsPdf, see DsPdf 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 DsPdf library supports the import and export of PDF forms data in FDF, XFDF and XML file formats.

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

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

Import or Export Forms Data from FDF

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

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

C#

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

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

}
```

Back to Top

Import or Export Forms Data from XFDF

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

class.

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

C#

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

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

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

Back to Top

Import or Export Forms Data from XML

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

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

C#

```
public void ImportDataFromXML()
{
    var doc = new GcPdfDocument();
    //Load the document
    doc.Load(new FileStream(Path.Combine("Pdf_BlankForm.pdf"), FileMode.Open,
FileAccess.Read));
    //Open the XML file
    FileStream stream = new FileStream(Path.Combine("XML_Data.xml"), FileMode.Open,
```

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

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

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

Back to Top

Form XObjects

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

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

1. Initialize two instances of GcPdfDocument class, one to load the existing PDF file from which the content is to be imported and the other one is the target PDF to which the imported content is to be rendered.
2. Create a list of the FormXObject 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

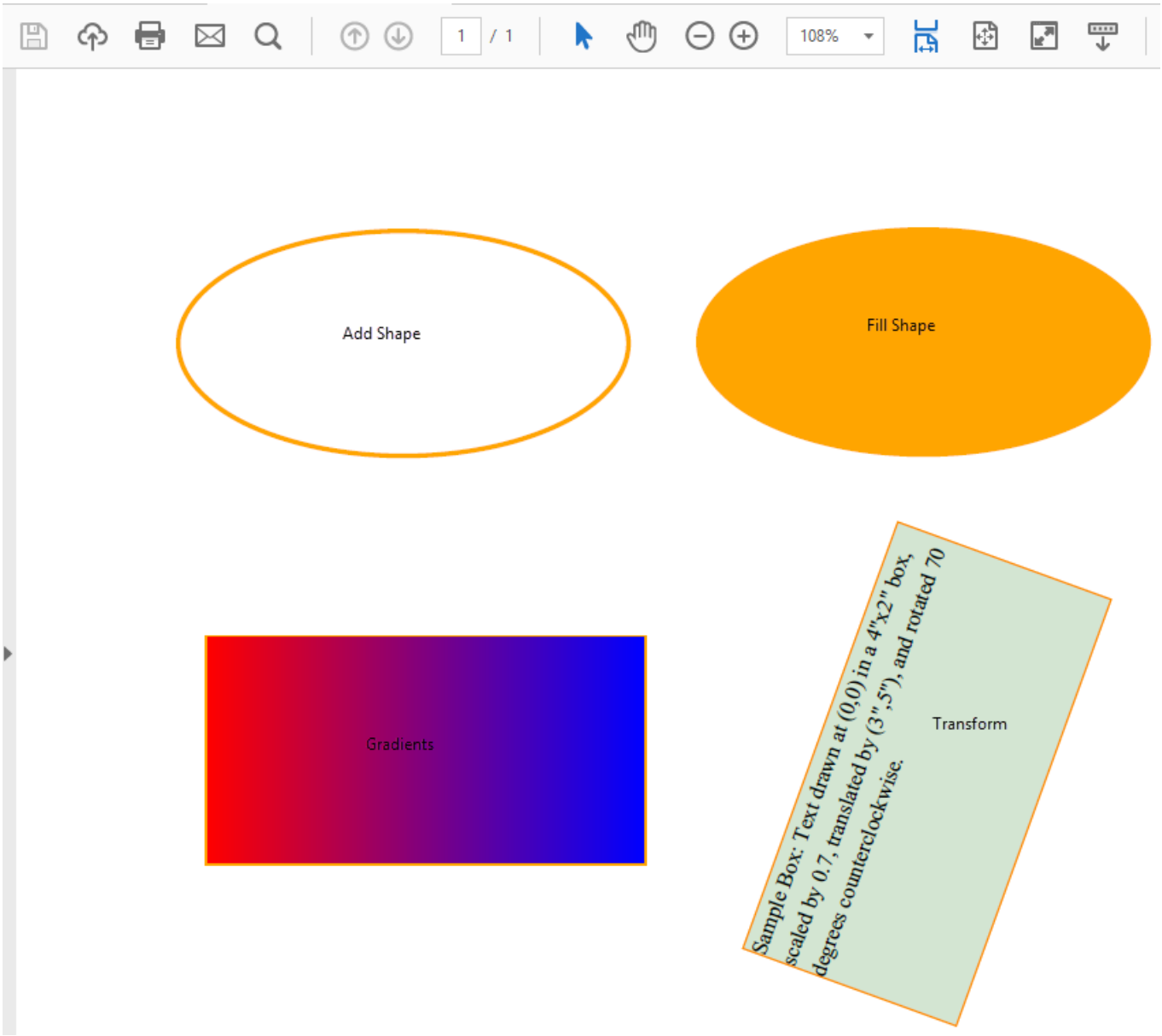
For more information about implementation of Form XObject feature using DsPdf, see DsPdf sample browser.

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.

DsPdf 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 DsPdf:

- 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 DsPdf, 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 DsPdf, see DsPdf 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.DefaultFormat.FontSize = 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(ifore, 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 DsPdf 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.

DsPdf 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

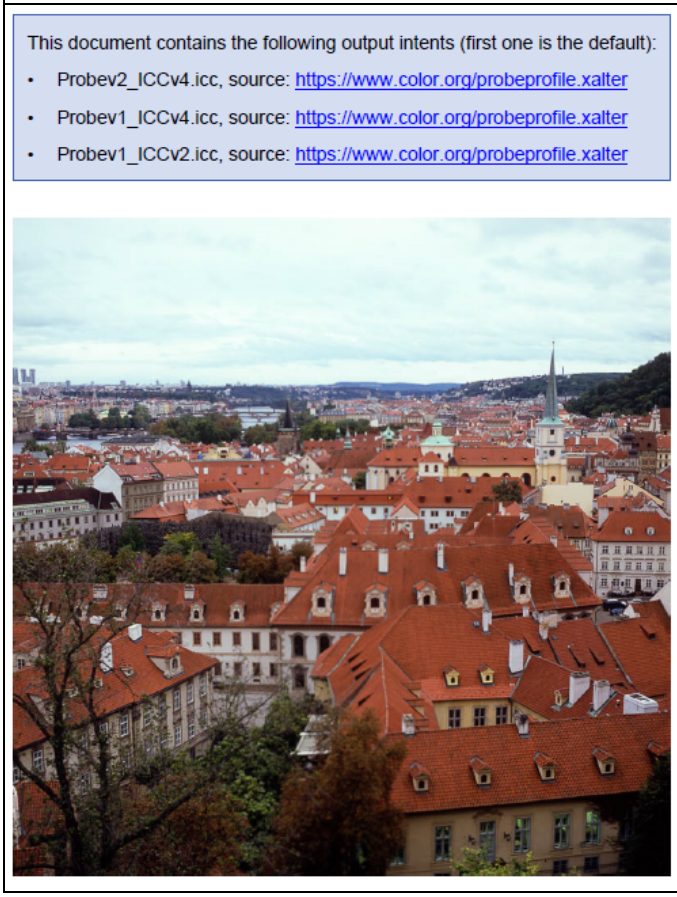
An output intent can be used in conjunction with the ICC profiles to convert the source colors to those required for the intended output. The ICC profiles are used for describing the output intents in PDF/A, PDF/X and PDF/VT standards. DsPdf provides the **ICCProfile** class which represents the ICC 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"),
    ("Probev1_ICCv4.icc", @"https://www.color.org/probeprofile.xalter"),
    ("Probev1_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
	

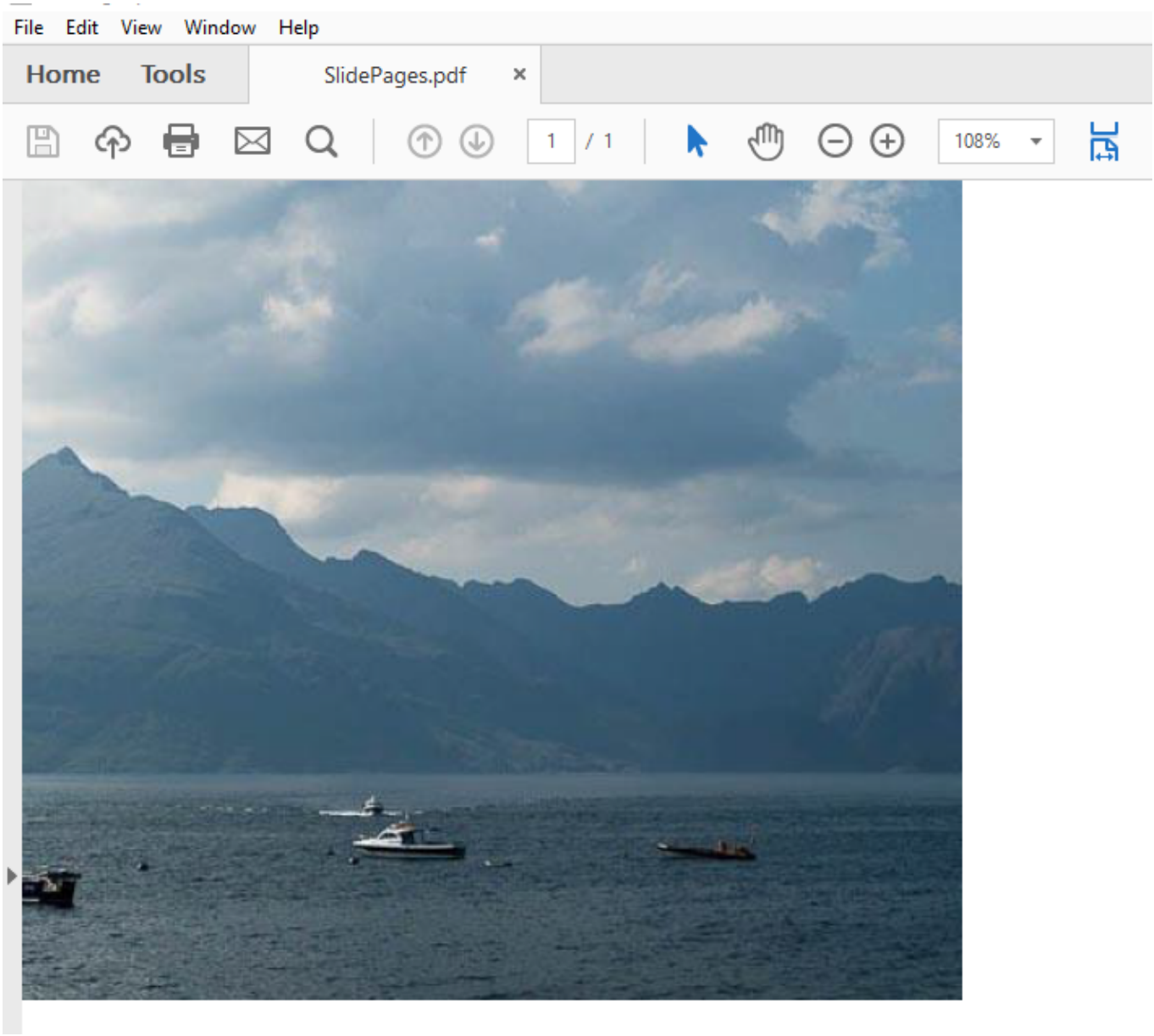
Note: The ICC profiles used in the above code can be downloaded from ICC Profile Registry.

Images

DsPdf allows you to draw an image on a page using DrawImage method of the GcGraphics class. You can load an image from a file or using a stream. 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.

While rendering a single image object repeatedly in a GcPdfDocument instance, DsPdf automatically stores the image in a dictionary and reuses the same object throughout the document. This lets you create PDF files of optimum size at a faster pace as it saves time and uses cache to load the same image multiple times. This feature stands very beneficial when rendering document that contains company logo in its repeating page header.

In addition, DsPdf 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 the 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

> **Note:** When creating a PDF document, adding the same image(s) in multiple places can increase the size of the document. So, to keep the size of the PDF document to a minimum, DsPdf provides RemoveDuplicateImages method in GcPdfDocument class that removes duplicate instances of the same image. For more information, see Optimize Document Size.

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 = Image.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();
```

Note: RawImage class is now obsolete. Instead, use **Image** class.

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 DsPdf, see DsPdf 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 DsPdf, see DsPdf 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 DsImaging 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.

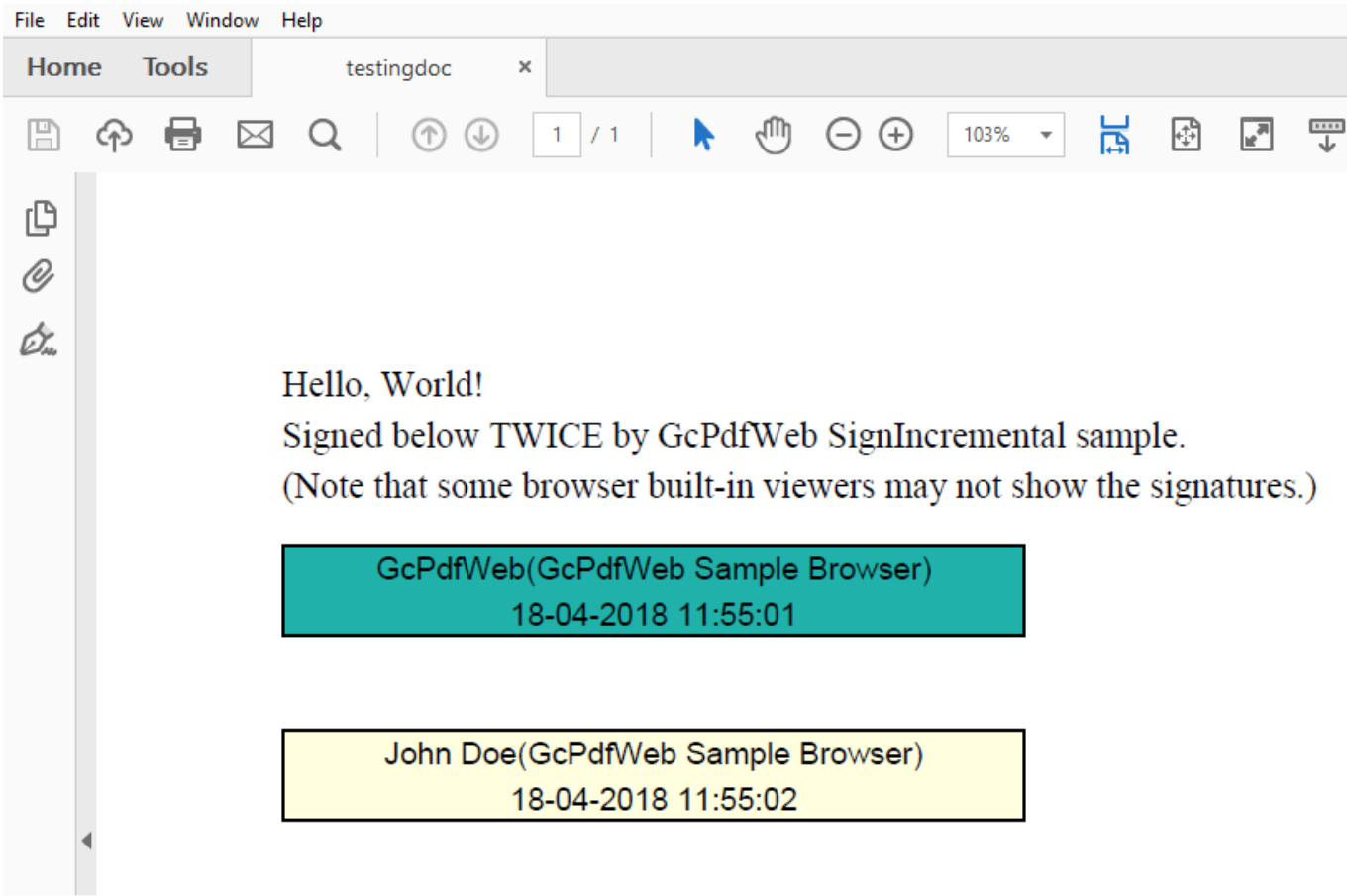
DsPdf allows you to incrementally update and save a modified document using Save method. By default, this method saves the document without an incremental update. However, you can save the document incrementally by setting the incrementalUpdate parameter of the Save method to true. The **Save** method 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, DsPdf 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. DsPdf 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 DsPdf, 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 = "DsPdfWeb Sample Browser",
            SignerName = "John Doe",
        };

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

        // Sign and save the document:
        // NOTES:
        // - Signing and saving is an atomic operation, the two cannot be
separated.
        // - The stream passed to the Sign() method must be readable.
        doc.Sign(sp2, stream);

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

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

    // This method is almost exactly the same as the DigitalSignature sample,
    // but adds a second signature field (does not sign it though):
    private Stream CreateAndSignPdf()
    {
        GcPdfDocument doc = new GcPdfDocument();
        Page page = doc.NewPage();
        TextFormat tf = new TextFormat() { Font = StandardFonts.Times, FontSize
= 14 };
        page.Graphics.DrawString(
            "Hello, World!\r\nSigned below TWICE by DsPdfWeb 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", "DsPdfTest.pfx");
        X509Certificate2 cert = new X509Certificate2(File.ReadAllBytes(pfxPath),
"qq",
        X509KeyStorageFlags.MachineKeySet | X509KeyStorageFlags.PersistKeySet
        | X509KeyStorageFlags.Exportable);
        SignatureProperties sp = new SignatureProperties();
        sp.Certificate = cert;
        sp.Location = "DsPdfWeb Sample Browser";
        sp.SignerName = "DsPdfWeb";

        // 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 DsPdf, see DsPdf 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. DsPdf 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 DsPdf 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(xxxx, 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.

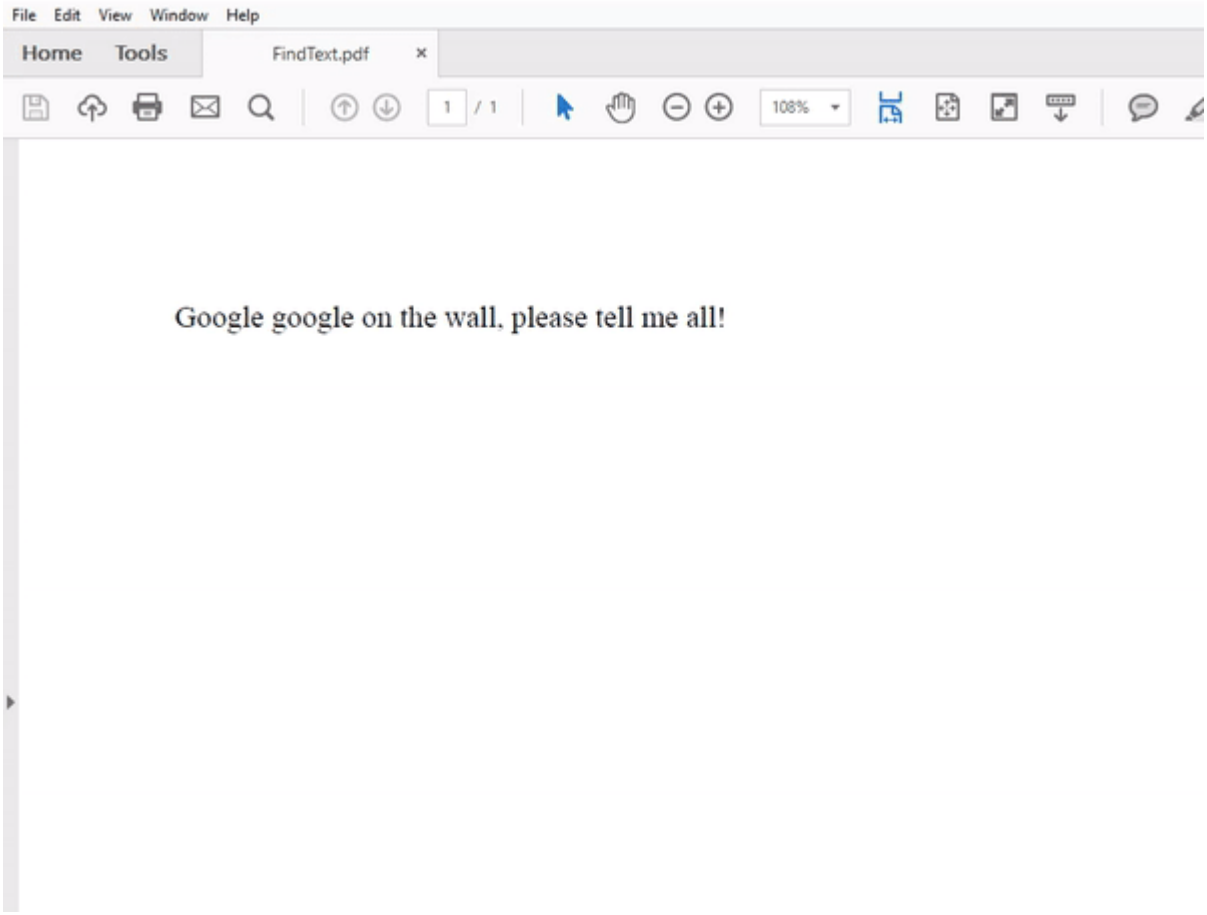
C#

```
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).

DsPdf 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);
}
```

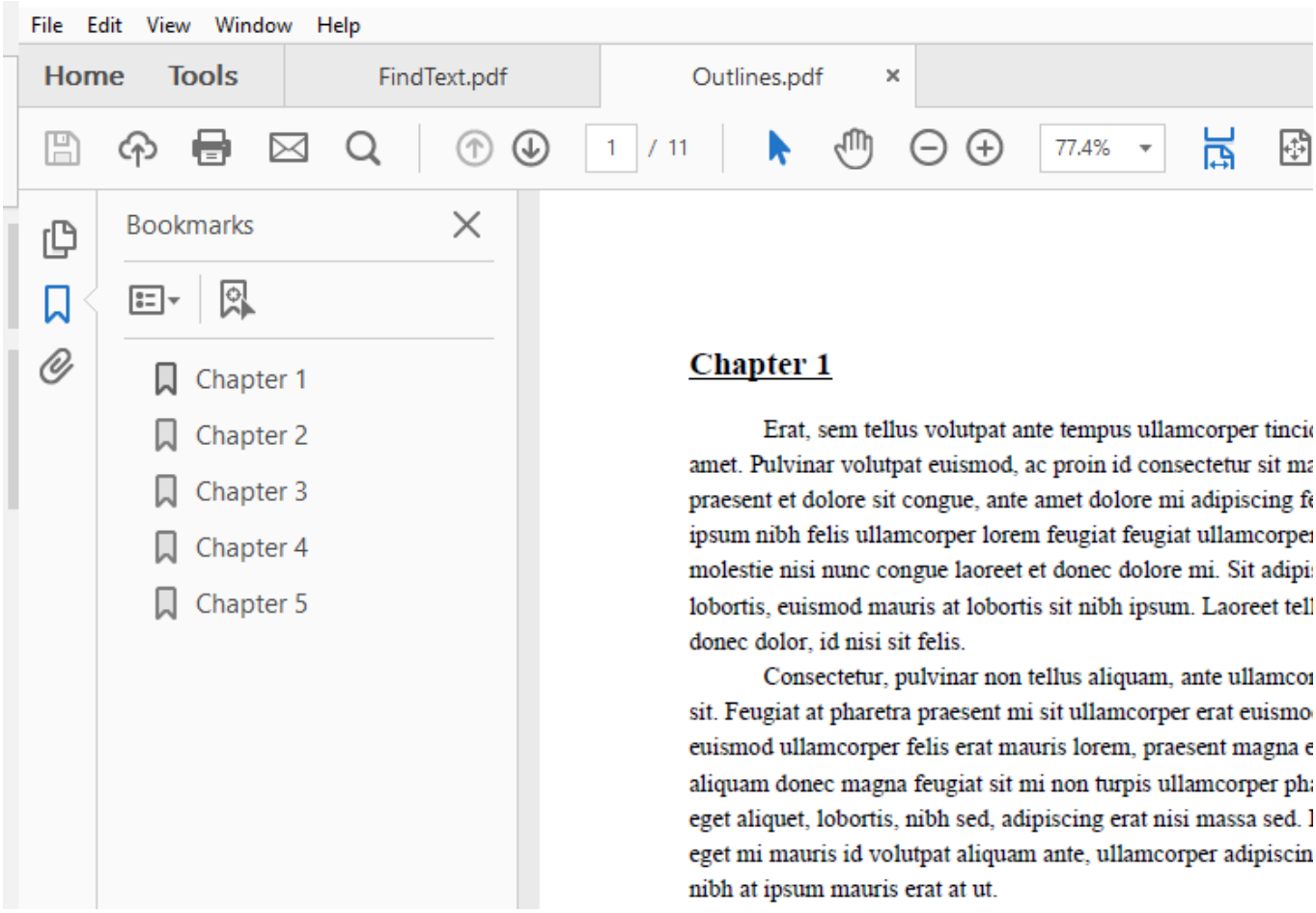
Back to Top

For more information about implementation of links using DsPdf, see DsPdf 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).

DsPdf 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 DsPdf, see DsPdf 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. DsPdf 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

DsPdf 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 DsPdf, 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 DsPdf, see DsPdf sample browser.

Clone a Page

DsPdf provides ClonePage method in PageCollection class to clone a particular page from a specified index in the PDF file and insert it into a specified index of the same PDF file. ClonePage method also supports two additional boolean type parameters: **cloneAnnotations** and **cloneFields**, which enable a user to allow or restrict the cloning of annotations and fields on the page to be cloned.

C#

```
// Initialize GcPdfDocument.
GcPdfDocument doc = new GcPdfDocument();

// Load the PDF file from the stream.
var fs = new FileStream(Path.Combine("digital-signature-sample.pdf"), FileMode.Open,
             FileAccess.Read);
doc.Load(fs);

// Clone page at index 0 and insert it at index 1, copy fields but skip annotations.
doc.Pages.ClonePage(0, 1, false, true);

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

Note: DsPdf also provides **MergeWithDocument** method that can be used to copy pages between different PDF files. For more information on merging documents, see Merge Documents.

Layouts

DsPdf provides LayoutRect and other related classes in the **GrapeCity.Documents.Layout** namespace to place multiple elements on a PDF page or image without having to calculate positions of each element relative to other ones.

The LayoutRect and other related classes implement the flat layout model. There are no chains, barriers, guidelines, biases, or other complications. Certain features of the layout model are:

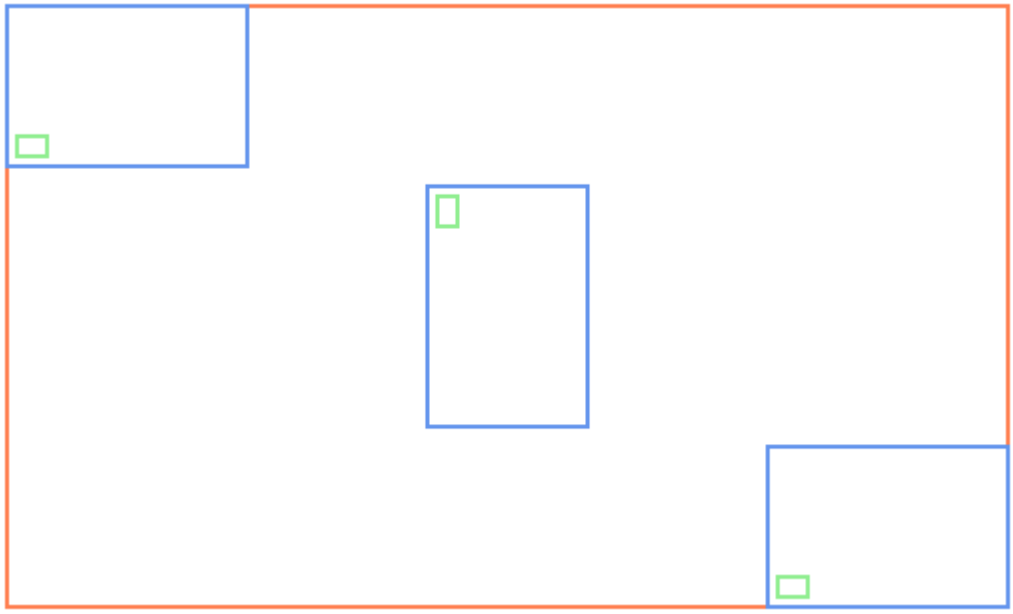
- Rectangles can be rotated by a multiple of 90 degrees.
- Constraints can reference anchor points from other views (with different transformation matrices).
- Rectangle sides can be bound to arbitrary contours.
- Views can be nested, and the inner view's transformation is automatically recalculated when the outer view's transformation changes.

LayoutHost is the main object, which defines the origin of the coordinate system. Also, this object performs layout when all other objects are prepared and linked. LayoutHost can create views. LayoutView defines a rectangular region with some width, height, and transformation, and the units of all sizes and coordinates are floats and can be of arbitrary dimension.

A LayoutHost can create multiple LayoutViews with different sizes and transformations, and each LayoutView can create multiple LayoutRects. A LayoutRect is a rectangle whose sides are parallel to the LayoutView sides. LayoutRect is defined by four points: P0, P1, P2, and P3.

The layout engine calculates the exact positions of the P0, P1, and P2 points for each LayoutRect of each LayoutView within a LayoutHost. The size and position of a LayoutRect can be determined if some of the following parameters are known: Width, Height, AspectRatio, Angle (as a multiple of 90 degrees), Left, Top, Right, Bottom, HorizontalCenter, VerticalCenter. The Width, Height and AspectRatio parameters are assigned directly; however, other parameters are usually defined as an offset or delta from the LayoutView, other rectangles, or the special anchor points.

The transformation matrix is **Matrix** from the **GrapeCity.Documents.Common** namespace. It has double precision vs. single precision Matrix3x2 from System.Numerics. The Matrix can easily be converted to Matrix3x2, or it can be multiplied by a Matrix3x2.



Refer to the following example code to draw a simple layout:

C#

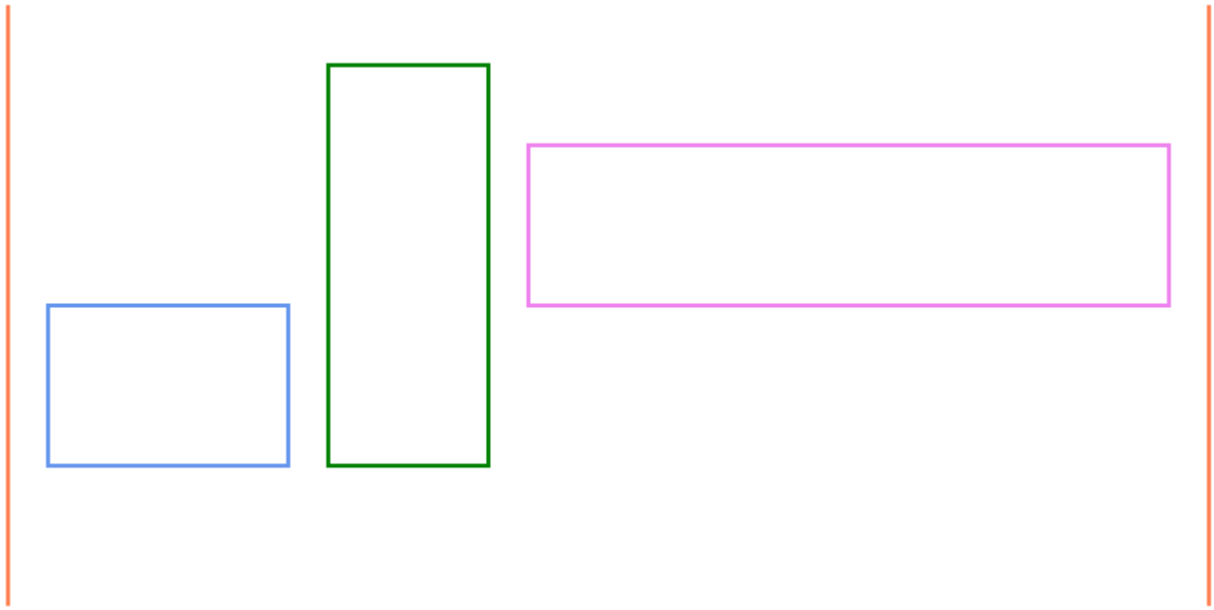
```
// Initialize LayoutHost. This defines the origin of the coordinate system.
var host = new LayoutHost();

// Create LayoutView. This defines a rectangular region with some width, height, and
transformation.
LayoutView view = host.CreateView(500, 300, Matrix.Identity);

// Create lists for blue and green rectangles.
var blueList = new List<LayoutRect>();
var greenList = new List<LayoutRect>();

// Create LayoutRect. LayoutRect is a rectangle whose sides are parallel to the owner
LayoutView sides.
LayoutRect rect = view.CreateRect();

// Set a constraint on the rotation angle of the LayoutRect.
rect.SetAngle(null, 90);

// Set width, height, and center point.
rect.SetWidth(120);
rect.SetHeight(80);
rect.SetHorizontalCenter(null, AnchorParam.VerticalCenter);
rect.SetVerticalCenter(null, AnchorParam.HorizontalCenter);

// Add the LayoutRect to the blue list.
blueList.Add(rect);

// Add a green LayoutRect to the blue LayoutRect.
AddGreenRect(rect);
```

```
// Add a tiny green rectangle at the bottom left corner of the blue rectangle.
void AddGreenRect(LayoutRect r0)
{
    var r1 = view.CreateRect();
    r1.SetWidth(r0, AnchorParam.Width, 0, 0.125f);
    r1.SetHeight(r0, AnchorParam.Height, 0, 0.125f);
    r1.AnchorBottomLeft(r0, 5, 5);
    greenList.Add(r1);
}

// Create two more blue rectangles and place them at different places in the
LayoutView.
// Add green rectangles to the blue rectangles.
rect = view.CreateRect();
rect.AnchorTopLeft(null, 0, 0, 120, 80);
blueList.Add(rect);
AddGreenRect(rect);

rect = view.CreateRect();
rect.AnchorBottomRight(null, 0, 0, 120, 80);
blueList.Add(rect);
AddGreenRect(rect);

// Calculate all rectangle coordinates based on the constraints provided.
host.PerformLayout();

// Initialize GcPdfDocument.
GcPdfDocument doc = new GcPdfDocument();

// Create a new page.
var page = doc.Pages.Add(new SizeF(600, 400));

// Initialize GcPdfGraphics.
var gr = page.Graphics;

var pen = new Pen(Color.Coral, 2);

// Set the transformation matrix of the LayoutView when creating the view.
var m = Matrix3x2.CreateTranslation(20, 20);
gr.Transform = view.Transform.Multiply(m);

// Draw a rectangle with the corresponding values of the LayoutView.
gr.DrawRectangle(view.AsRectF(), pen);

// Draw blue rectangles.
pen.Color = Color.CornflowerBlue;
for (int i = 0; i < blueList.Count; i++)
{
    rect = blueList[i];
    gr.Transform = rect.Transform.Multiply(m);
```

```
    gr.DrawRectangle(rect.AsRectF(), pen);
}

// Draw green rectangles.
pen.Color = Color.LightGreen;
for (int i = 0; i < greenList.Count; i++)
{
    rect = greenList[i];
    gr.Transform = rect.Transform.Multiply(m);
    gr.DrawRectangle(rect.AsRectF(), pen);
}

// Save the PDF file.
doc.Save("Layout.pdf");
```

Simple Position Constraints

Simple position constraints change one of the following LayoutRect parameters: Left, Right, Top, Bottom, HorizontalCenter, and VerticalCenter.



Refer to the following example code to draw a layout with simple position constraints:

C#

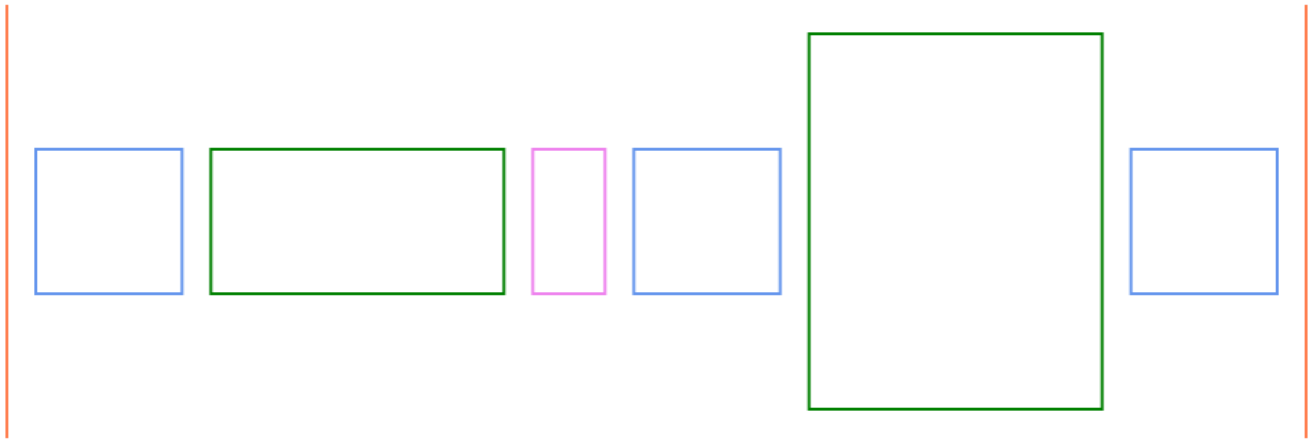
```
// Initialize LayoutHost. This defines the origin of the coordinate system.
var host = new LayoutHost();

// Create LayoutView. This defines a rectangular region with width and height.
LayoutView view = host.CreateView(600, 300);

// Create a left vertical line by setting AnchorParam to left.
var bL = view.CreateRect();
bL.AnchorVerticalLine(null);
```

```
bL.SetLeft(null, AnchorParam.Left);

// Create a right vertical line by setting AnchorParam to right.
var bR = view.CreateRect();
bR.AnchorVerticalLine(null);
bR.SetRight(null, AnchorParam.Right);

// Create a blue rectangle and set its AnchorParam.
var r1 = view.CreateRect();
r1.SetTop(null, AnchorParam.VerticalCenter);
r1.SetWidth(120);
r1.SetHeight(80);
r1.SetLeft(bL, AnchorParam.Right, 20);

// Create a green rectangle (rotated 270 degrees or 90 degrees counterclockwise) and
set its AnchorParam.
var r2 = view.CreateRect();
r2.SetAngle(r1, 270);
r2.SetLeft(r1, AnchorParam.Bottom);
r2.SetWidth(200);
r2.SetHeight(80);
r2.SetTop(r1, AnchorParam.Right, 20);

// Create a violet rectangle and set its AnchorParam.
var r3 = view.CreateRect();
r3.SetAngle(r1, 0);
r3.SetBottom(r1, AnchorParam.Top);
r3.SetHeight(80);
r3.SetLeft(r2, AnchorParam.Bottom, 20);
r3.SetRight(bR, AnchorParam.Left, -20);

// Calculate all rectangle coordinates based on the constraints provided.
host.PerformLayout();

// Initialize GcPdfDocument.
GcPdfDocument doc = new GcPdfDocument();

// Create a new page.
var page = doc.Pages.Add(new SizeF(700, 400));

// Initialize GcPdfGraphics.
var gr = page.Graphics;

// Define transformation matrix.
var m = Matrix3x2.CreateTranslation(20, 20);

// Draw the vertical lines and rectangles.
DrawRect(bL, Color.Coral);
DrawRect(bR, Color.Coral);
DrawRect(r1, Color.CornflowerBlue);
DrawRect(r2, Color.Green);
```

```
DrawRect(r3, Color.Violet);
void DrawRect(LayoutRect r, Color c)
{
    gr.Transform = r.Transform.Multiply(m);
    gr.DrawRectangle(r.AsRectF(), new Pen(c, 2));
}

// Save the PDF file.
doc.Save("Layout.pdf");
```

Chained Position Constraints

Chained position constraints set the parameters of a LayoutRect to fill the whole width or height of some area with multiple rectangles having different proportional sizes (measured in stars). The widths or heights of the rectangles will be proportional to their weights (number of stars). The SetLeftAndOpposite and SetBottomAndOpposite methods are used to set the chained constraints. These methods create two constraints at once.



Refer to the following example code to draw a layout with chained position constraints:

C#

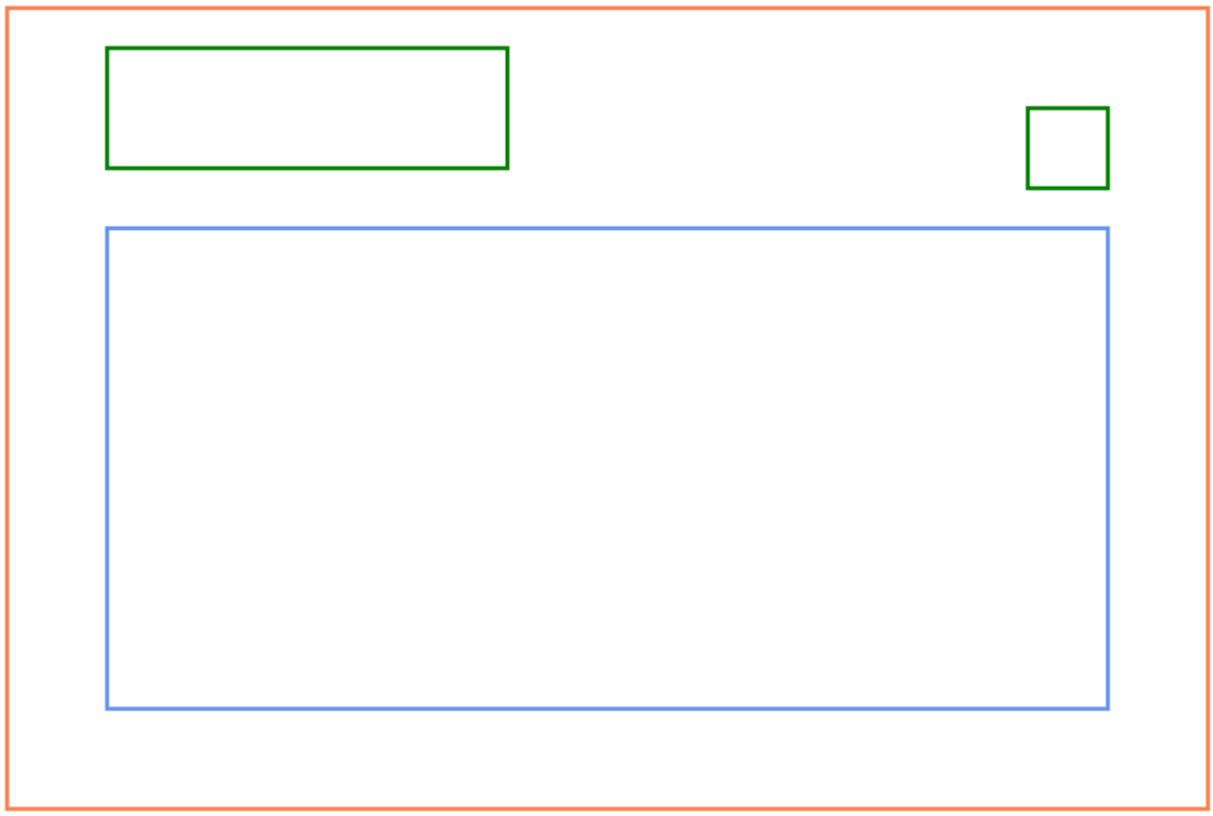
```
// Initialize LayoutHost. This defines the origin of the coordinate system.
var host = new LayoutHost();

// Create LayoutView. This defines a rectangular region with some width and height.
LayoutView view = host.CreateView(900, 300);

// Create a left vertical line by setting AnchorParam to left.
var bL = view.CreateRect();
bL.AnchorVerticalLine(null);
bL.SetLeft(null, AnchorParam.Left);

// Create a right vertical line by setting AnchorParam to right.
var bR = view.CreateRect();
bR.AnchorVerticalLine(null);
bR.SetRight(null, AnchorParam.Right);

// Create blue, green, and violet rectangles.
```

```
var r1 = view.CreateRect();
r1.AnchorTopBottom(null, 100, 100);

var r2 = view.CreateRect();
r2.AnchorTopBottom(null, 100, 100);

var r3 = view.CreateRect();
r3.AnchorTopBottom(null, 100, 100);

var r4 = view.CreateRect();
r4.AnchorTopBottom(null, 100, 100);

var r6 = view.CreateRect();
r6.AnchorTopBottom(null, 100, 100);

// Create a green rectangle rotated 90 degrees.
var r5 = view.CreateRect();
r5.SetAngle(null, 90);
r5.SetLeft(null, AnchorParam.Top, 20);
r5.SetRight(null, AnchorParam.Bottom, -20);

// Create a chain of rectangles by setting the AnchorParam. The SetLeftAndOpposite
and SetBottomAndOpposite methods are used to set chained position constraints.
r1.SetLeft(bL, AnchorParam.Right, 20);
r2.SetLeftAndOpposite(r1, AnchorParam.Right, 20);
r3.SetLeftAndOpposite(r2, AnchorParam.Right, 20);
r4.SetLeftAndOpposite(r3, AnchorParam.Right, 20);
r5.SetBottomAndOpposite(r4, AnchorParam.Right, -20);
r6.SetLeftAndOpposite(r5, AnchorParam.Top, 20);
r6.SetRight(bR, AnchorParam.Left, -20);

// Set the star and fixed widths.
r1.SetStarWidth(2);
r2.SetStarWidth(4);
r3.SetWidth(50);
r4.SetStarWidth(2);
r6.SetStarWidth(2);

// Set the star height as this rectangle is rotated.
r5.SetStarHeight(4);

// Calculate all rectangle coordinates based on the constraints provided.
host.PerformLayout();

// Initialize GcPdfDocument.
GcPdfDocument doc = new GcPdfDocument();

// Create a new page.
var page = doc.Pages.Add(new SizeF(1000, 400));

// Initialize GcPdfGraphics.
```

```
var gr = page.Graphics;

// Define transformation matrix.
var m = Matrix3x2.CreateTranslation(20, 20);

// Draw the vertical lines and rectangles.
DrawRect(bL, Color.Coral);
DrawRect(bR, Color.Coral);
DrawRect(r1, Color.CornflowerBlue);
DrawRect(r2, Color.Green);
DrawRect(r3, Color.Violet);
DrawRect(r4, Color.CornflowerBlue);
DrawRect(r5, Color.Green);
DrawRect(r6, Color.CornflowerBlue);
void DrawRect(LayoutRect r, Color c)
{
    gr.Transform = r.Transform.Multiply(m);
    gr.DrawRectangle(r.AsRectF(), new Pen(c, 2));
}

// Save the PDF file.
doc.Save("Layout.pdf");
```

Minimum or Maximum Position Constraints

Min/Max position constraints bind a single LayoutRect parameter to one or several other LayoutRects.
The AppendMinTop method is used in the following case to define the minimum top gap between the LayoutRect and other LayoutRects.



Refer to the following example code to draw a layout with minimum position constraints:

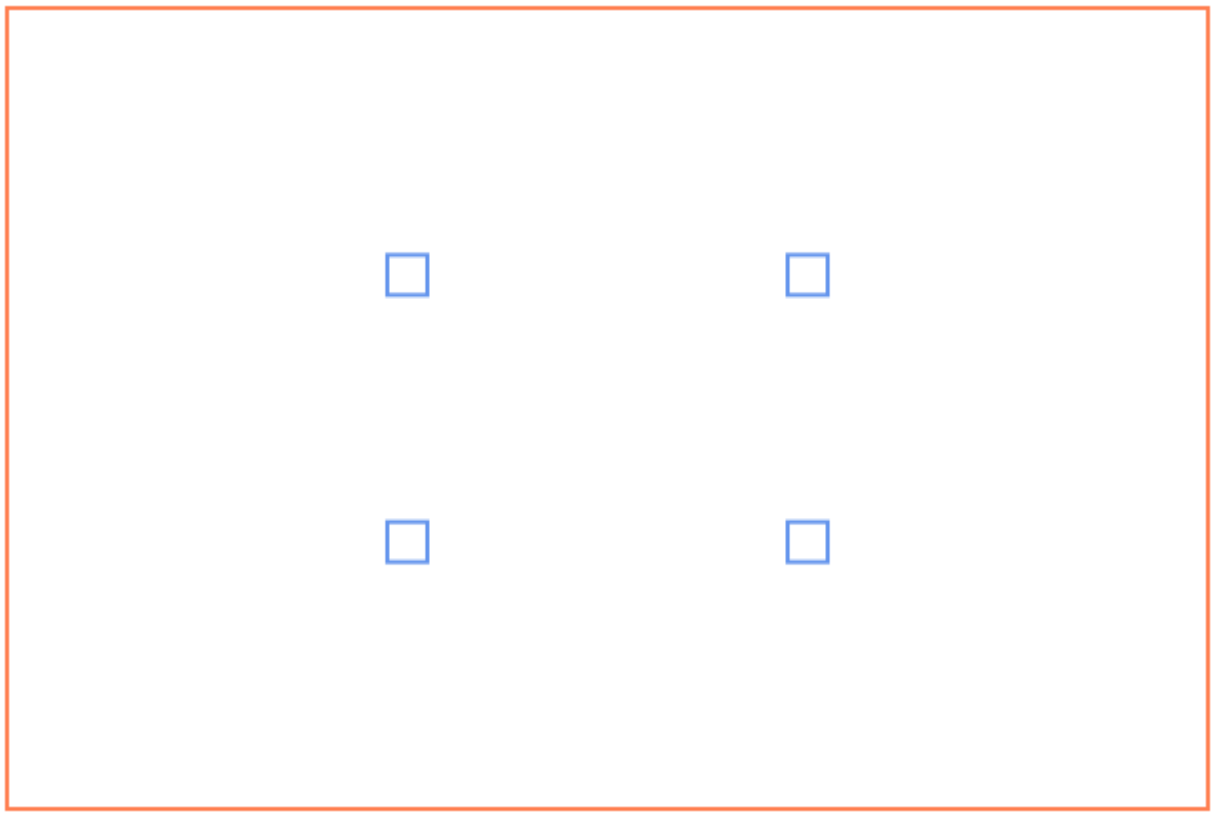
C#

```
// Initialize LayoutHost. This defines the origin of the coordinate system.
var host = new LayoutHost();

// Create LayoutView. This defines a rectangular region with some width and height.
LayoutView view = host.CreateView(600, 400);

// Create an outer rectangle.
var rOuter = view.CreateRect();
rOuter.AnchorExact(null);

// Create an inner rectangle.
var rInner = view.CreateRect();
rInner.AnchorBottomLeftRight(rOuter, 50, 50, 50);

// Create rectangles.
var r1 = view.CreateRect();
r1.AnchorTopLeft(rOuter, 20, 50, 200, 60);
var r2 = view.CreateRect();
r2.AnchorTopRight(rOuter, 50, 50, 40, 40);

// Set the position of the inner rectangle according to the other rectangles.
rInner.AppendMinTop(rOuter, AnchorParam.Top, 50);
rInner.AppendMinTop(r1, AnchorParam.Bottom, 20);
```

```
rInner.AppendMinTop(r2, AnchorParam.Bottom, 20);

// Calculate all rectangle coordinates based on the constraints provided.
host.PerformLayout();

// Initialize GcPdfDocument.
GcPdfDocument doc = new GcPdfDocument();

// Create a new page.
var page = doc.Pages.Add(new SizeF(700, 500));

// Initialize GcPdfGraphics.
var gr = page.Graphics;

// Define transformation matrix.
var m = Matrix3x2.CreateTranslation(20, 20);

// Draw the rectangles.
DrawRect(rOuter, Color.Coral);
DrawRect(rInner, Color.CornflowerBlue);
DrawRect(r1, Color.Green);
DrawRect(r2, Color.Green);
void DrawRect(LayoutRect r, Color c)
{
    gr.Transform = r.Transform.Multiply(m);
    gr.DrawRectangle(r.AsRectF(), new Pen(c, 2));
}

// Save the PDF file.
doc.Save("Layout.pdf");
```

Anchor Points

Anchor points set the parameters of a LayoutRect relative to a LayoutView, other LayoutRects, or the special anchor points. The anchor points can be created with the CreatePoint method of a LayoutView or LayoutRect object.



Refer to the following example code to draw a layout with anchor points:

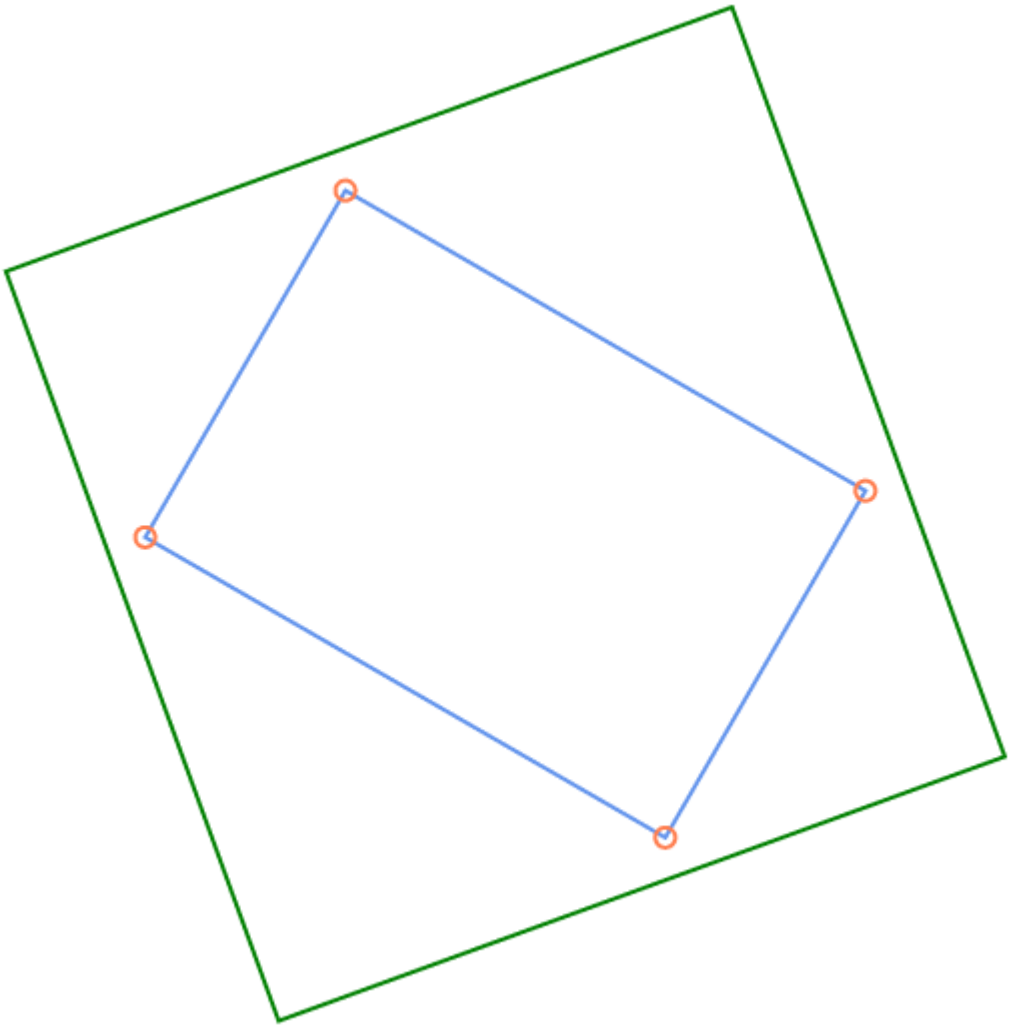
C#

```
// Initialize LayoutHost. This defines the origin of the coordinate system.
var host = new LayoutHost();

// Create LayoutView. This defines a rectangular region with some width and height.
LayoutView view = host.CreateView(600, 400);

// Create main rectangle.
var rMain = view.CreateRect();
rMain.AnchorExact(null);

// Create anchor points.
var ap1 = rMain.CreatePoint(1f / 3, 1f / 3);
var ap2 = rMain.CreatePoint(2f / 3, 1f / 3);
var ap3 = rMain.CreatePoint(1f / 3, 2f / 3);
var ap4 = rMain.CreatePoint(2f / 3, 2f / 3);

// Create four rectangles and position them as per the anchor points defined.
var r1 = view.CreateRect();
AnchorCenter(r1, ap1);
var r2 = view.CreateRect();
AnchorCenter(r2, ap2);
var r3 = view.CreateRect();
AnchorCenter(r3, ap3);
```

```
var r4 = view.CreateRect();
AnchorCenter(r4, ap4);
void AnchorCenter(LayoutRect r, AnchorPoint ap)
{
    r.SetHorizontalCenter(ap);
    r.SetVerticalCenter(ap);
    r.SetWidth(20);
    r.SetHeight(20);
}

// Calculate all rectangle coordinates based on the constraints provided.
host.PerformLayout();

// Initialize GcPdfDocument.
GcPdfDocument doc = new GcPdfDocument();

// Create a new page.
var page = doc.Pages.Add(new SizeF(700, 500));

// Initialize GcPdfGraphics.
var gr = page.Graphics;

// Define transformation matrix.
var m = Matrix3x2.CreateTranslation(20, 20);

// Draw the rectangles.
DrawRect(rMain, Color.Coral);
DrawRect(r1, Color.CornflowerBlue);
DrawRect(r2, Color.CornflowerBlue);
DrawRect(r3, Color.CornflowerBlue);
DrawRect(r4, Color.CornflowerBlue);
void DrawRect(LayoutRect r, Color c)
{
    gr.Transform = r.Transform.Multiply(m);
    gr.DrawRectangle(r.AsRectF(), new Pen(c, 2));
}

// Save the PDF file.
doc.Save("Layout.pdf");
```

Constraints based on other LayoutView

The LayoutRect parameters cannot be bound to a LayoutRect from another LayoutView. However, it is possible to bind parameters to any anchor point within the same LayoutHost.



Refer to the following example code to draw a layout circumscribed in a layout from another LayoutView:

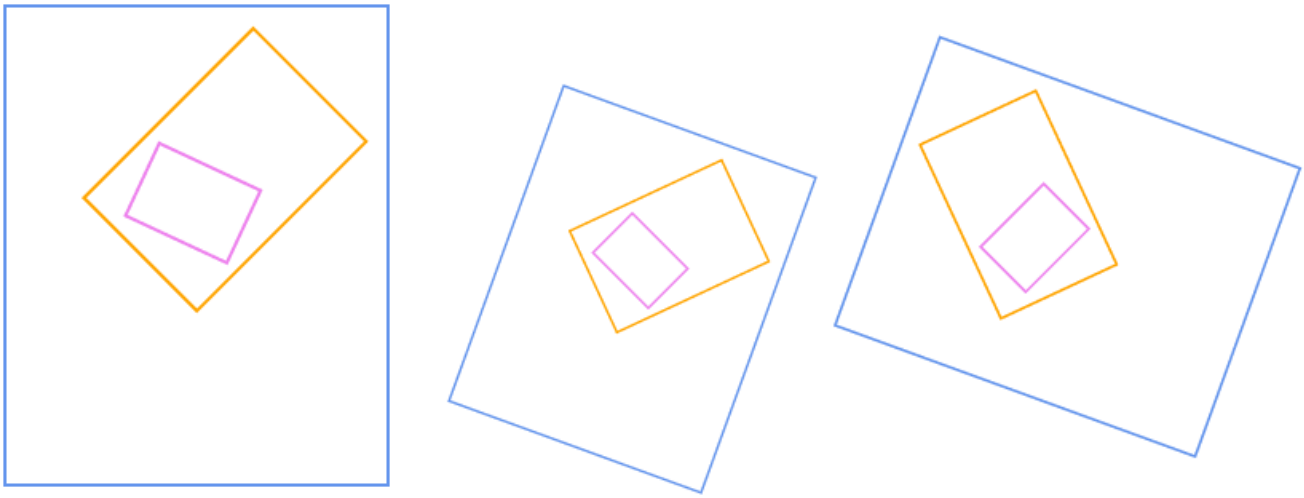
C#

```
// Initialize LayoutHost. This defines the origin of the coordinate system.
var host = new LayoutHost();

//Create rotation.
const double DegToRad = Math.PI / 180;
var m1 = Matrix.CreateRotation(DegToRad * 30);
m1 = m1.Translate(190, -50);

// Create first view and rectangle.
var view1 = host.CreateView(10, 10, m1);
var rc1 = view1.CreateRect();
rc1.AnchorTopLeft(null, 30, 30, 300, 200);

// Create second view and rectangle.
var m2 = Matrix.CreateRotation(DegToRad * -20);
var view2 = host.CreateView(10, 10, m2);
var rc2 = view2.CreateRect();
```

```
// Create anchor points.
var ap1 = rc1.CreatePoint(0, 0);
var ap2 = rc1.CreatePoint(1, 0);
var ap3 = rc1.CreatePoint(1, 1);
var ap4 = rc1.CreatePoint(0, 1);

// Add constraints relative to the anchor points.
rc2.SetTop(ap1, -20);
rc2.SetBottom(ap3, 20);
rc2.SetLeft(ap4, -20);
rc2.SetRight(ap2, 20);

// Calculate all rectangle coordinates based on the constraints provided.
host.PerformLayout();

// Initialize GcPdfDocument.
GcPdfDocument doc = new GcPdfDocument();

// Create a new page.
var page = doc.Pages.Add(new SizeF(700, 600));

// Initialize GcPdfGraphics.
var gr = page.Graphics;

// Define transformation matrix.
var m = Matrix3x2.CreateTranslation(20, 20);

// Draw the rectangles and ellipses.
DrawRect(rc1, Color.CornflowerBlue);
DrawRect(rc2, Color.Green);
DrawPoint(ap1);
DrawPoint(ap2);
DrawPoint(ap3);
DrawPoint(ap4);
void DrawRect(LayoutRect r, Color c)
{
    gr.Transform = r.Transform.Multiply(m);
    gr.DrawRectangle(r.AsRectF(), new Pen(c, 2));
}
void DrawPoint(AnchorPoint ap)
{
    gr.Transform = ap.Transform.Multiply(m);
    gr.DrawEllipse(new RectangleF(-5, -5, 10, 10), new Pen(Color.Coral, 2));
}

// Save the PDF file.
doc.Save("Layout.pdf");
```

Dependent Views and Transformations

The hierarchy of LayoutViews is not necessarily flat within the same LayoutHost. Some views can be nested in other views. When the transformation matrix of the parent LayoutView is updated, all child view transformations are recalculated.



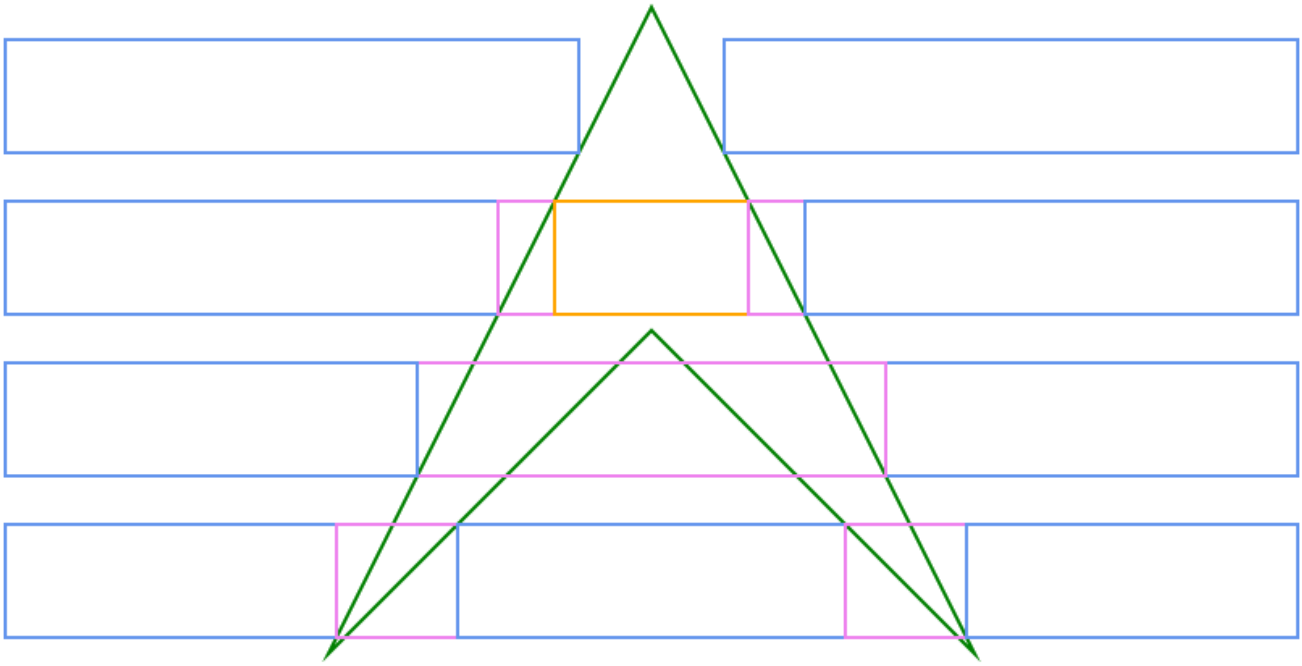
C#

```
// Initialize LayoutHost. This defines the origin of the coordinate system.
var host = new LayoutHost();

// Create first view and rectangle.
var view1 = host.CreateView(240, 300);
var rc1 = view1.CreateRect();
rc1.AnchorExact(null);

// Create second view and rectangle.
var view2 = host.CreateView(100, 150);
var rc2 = view2.CreateRect();
rc2.AnchorExact(null);

// Create third view and rectangle.
var view3 = host.CreateView(70, 50);
var rc3 = view3.CreateRect();
rc3.AnchorExact(null);

//Create rotation.
const double DegToRad = Math.PI / 180;
var m2 = Matrix.CreateRotation(DegToRad * 45);
view2.SetRelativeTransform(view1, m2.Translate(120, -100));
var m3 = Matrix.CreateRotation(DegToRad * -20);
view3.SetRelativeTransform(view2, m3.Translate(-23, 90));

// Calculate all rectangle coordinates based on the constraints provided.
host.PerformLayout();

// Initialize GcPdfDocument.
GcPdfDocument doc = new GcPdfDocument();
```

```
// Create a new page.
var page = doc.Pages.Add(new SizeF(850, 350));

// Initialize GcPdfGraphics.
var gr = page.Graphics;

// Draw the first set of rectangles according to the first transformation matrix.
var m = Matrix3x2.CreateTranslation(20, 20);
DrawRect(rc1, Color.CornflowerBlue);
DrawRect(rc2, Color.Orange);
DrawRect(rc3, Color.Violet);

// Draw the second set of rectangles according to the second transformation matrix.
view1.Transform = Matrix.CreateTranslation(350, 50).Scale(0.7).Rotate(DegToRad * 20);
host.PerformLayout();
DrawRect(rc1, Color.CornflowerBlue);
DrawRect(rc2, Color.Orange);
DrawRect(rc3, Color.Violet);

// Draw the third set of rectangles according to the third transformation matrix.
view1.Transform = Matrix.CreateTranslation(520, 200).Scale(0.8).Rotate(DegToRad * -
70);
host.PerformLayout();
DrawRect(rc1, Color.CornflowerBlue);
DrawRect(rc2, Color.Orange);
DrawRect(rc3, Color.Violet);
void DrawRect(LayoutRect r, Color c)
{
    gr.Transform = r.Transform.Multiply(m);
    gr.DrawRectangle(r.AsRectF(), new Pen(c, 2));
}

// Save the PDF file.
doc.Save("Layout.pdf");
```

Contours

Contour is a closed figure drawn through anchor points. One side of a LayoutRect can be bound to one or several contours. From LayoutRect's point of view, contours consist of the outer area, the inner area, and the border area. Rectangles can be bound to positions where one area changes to another. The CreateContour method of LayoutView class creates an object of Contour class, which is used to draw a contour.



C#

```
// Initialize LayoutHost. This defines the origin of the coordinate system.
var host = new LayoutHost();

// Create LayoutView. This defines a rectangular region with some width and height.
LayoutView view = host.CreateView(800, 400);

// Create a contour.
var contour = view.CreateContour();
contour.AddPoints(new AnchorPoint[]
{
    view.CreatePoint(0, 0, 400, 0),
    view.CreatePoint(0, 0, 600, 400),
    view.CreatePoint(0, 0, 400, 200),
    view.CreatePoint(0, 0, 200, 400)
});

// Create first row of rectangles.
var rc11 = view.CreateRect();
rc11.AnchorLeftTopBottom(null, 0, 20, 310);
rc11.AppendMaxRight(contour, ContourPosition.FirstInOutside);
var rc12 = view.CreateRect();
rc12.AnchorRightTopBottom(null, 0, 20, 310);
rc12.AppendMinLeft(contour, ContourPosition.FirstInOutside);

// Create second row of rectangles.
var rc21 = view.CreateRect();
rc21.AnchorLeftTopBottom(null, 0, 120, 210);
rc21.AppendMaxRight(contour, ContourPosition.FirstInOutside);
var rc22 = view.CreateRect();
```

```
rc22.AnchorTopBottom(null, 120, 210);
rc22.SetLeft(rc21, AnchorParam.Right);
rc22.AppendMaxRight(contour, ContourPosition.FirstInInside);
var rc23 = view.CreateRect();
rc23.AnchorTopBottom(null, 120, 210);
rc23.SetLeft(rc22, AnchorParam.Right);
rc23.AppendMaxRight(contour, ContourPosition.NextOutInside);
var rc24 = view.CreateRect();
rc24.AnchorTopBottom(null, 120, 210);
rc24.SetLeft(rc23, AnchorParam.Right);
rc24.AppendMaxRight(contour, ContourPosition.NextOutOutside);
var rc25 = view.CreateRect();
rc25.SetLeft(rc24, AnchorParam.Right);
rc25.AnchorRightTopBottom(null, 0, 120, 210);

// Create third row of rectangles.
var rc31 = view.CreateRect();
rc31.AnchorRightTopBottom(null, 0, 220, 110);
rc31.AppendMinLeft(contour, ContourPosition.FirstInOutside);
var rc32 = view.CreateRect();
rc32.AnchorTopBottom(null, 220, 110);
rc32.SetRight(rc31, AnchorParam.Left);
rc32.AppendMinLeft(contour, ContourPosition.LastOutOutside);
var rc33 = view.CreateRect();
rc33.SetRight(rc32, AnchorParam.Left);
rc33.AnchorLeftTopBottom(null, 0, 220, 110);

// Create fourth row of rectangles.
var rc41 = view.CreateRect();
rc41.AnchorLeftTopBottom(null, 0, 320, 10);
rc41.AppendMaxRight(contour, ContourPosition.FirstInOutside);
var rc42 = view.CreateRect();
rc42.AnchorTopBottom(null, 320, 10);
rc42.SetLeft(rc41, AnchorParam.Right);
rc42.AppendMaxRight(contour, ContourPosition.NextOutOutside);
var rc43 = view.CreateRect();
rc43.AnchorTopBottom(null, 320, 10);
rc43.SetLeft(rc42, AnchorParam.Right);
rc43.AppendMaxRight(contour, ContourPosition.FirstInOutside);
var rc44 = view.CreateRect();
rc44.AnchorTopBottom(null, 320, 10);
rc44.SetLeft(rc43, AnchorParam.Right);
rc44.AppendMaxRight(contour, ContourPosition.NextOutOutside);
var rc45 = view.CreateRect();
rc45.SetLeft(rc44, AnchorParam.Right);
rc45.AnchorRightTopBottom(null, 0, 320, 10);

// Calculate all rectangle coordinates based on the constraints provided.
host.PerformLayout();

// Initialize GcPdfDocument.
```

```
GcPdfDocument doc = new GcPdfDocument();

// Create a new page.
var page = doc.Pages.Add(new SizeF(900, 500));

// Initialize GcPdfGraphics.
var gr = page.Graphics;

// Define transformation matrix.
var m = Matrix3x2.CreateTranslation(20, 20);
gr.Transform = m;

// Draw the rectangles and contour.
DrawContour(contour);
DrawRect(rc11, Color.CornflowerBlue);
DrawRect(rc12, Color.CornflowerBlue);
DrawRect(rc21, Color.CornflowerBlue);
DrawRect(rc22, Color.Violet);
DrawRect(rc23, Color.Orange);
DrawRect(rc24, Color.Violet);
DrawRect(rc25, Color.CornflowerBlue);
DrawRect(rc31, Color.CornflowerBlue);
DrawRect(rc32, Color.Violet);
DrawRect(rc33, Color.CornflowerBlue);
DrawRect(rc41, Color.CornflowerBlue);
DrawRect(rc42, Color.Violet);
DrawRect(rc43, Color.CornflowerBlue);
DrawRect(rc44, Color.Violet);
DrawRect(rc45, Color.CornflowerBlue);
void DrawContour(Contour co)
{
    var pts = co.Points.Select(ap => ap.TransformedLocation).ToArray();
    gr.DrawPolygon(pts, new Pen(Color.Green, 2));
}
void DrawRect(LayoutRect r, Color c)
{
    gr.Transform = r.Transform.Multiply(m);
    gr.DrawRectangle(r.AsRectF(), new Pen(c, 2));
}

// Save the PDF file.
doc.Save("Layout.pdf");
```

Complex Graphic Layouts

DsPdf provides Surface, Layer, View, Space, and Visual classes in GrapeCity.Documents.Layout.Composition namespace that acts as a medium between the layout engine and the drawing surface, allowing you to draw complex graphics, text, and images. Furthermore, these classes also enable you to customize the z-order and clipping of the drawn graphics.

Surface is the main class in the Composition engine. Surface class contains a LayoutHost (the layout engine's root

object) and one or more views (layers). Layers consist of visuals (drawable elements) and nested layers. The Render method of Surface class calls PerformLayout method of LayoutHost class to calculate the surface layout and then it draws all the layers, including nested ones, from the bottom to the top layer on the specified **GcGraphics** object

Layers are of two types: Layer and View class objects (derived from Layer objects). The View object encapsulates the LayoutView object, which represents a separate coordinate system with its own transformation matrix. The Layer object functions as a lightweight View with its own list of visuals, nested layers, and possible clipping area. The Surface object can only create Views, not Layers. However, each View object (as well as the Layer object) can create both nested Layers and nested Views. You must create at least one View on the Surface then use that View to create nested Layers (with the same transformation) or nested Views (with different transformation matrices).

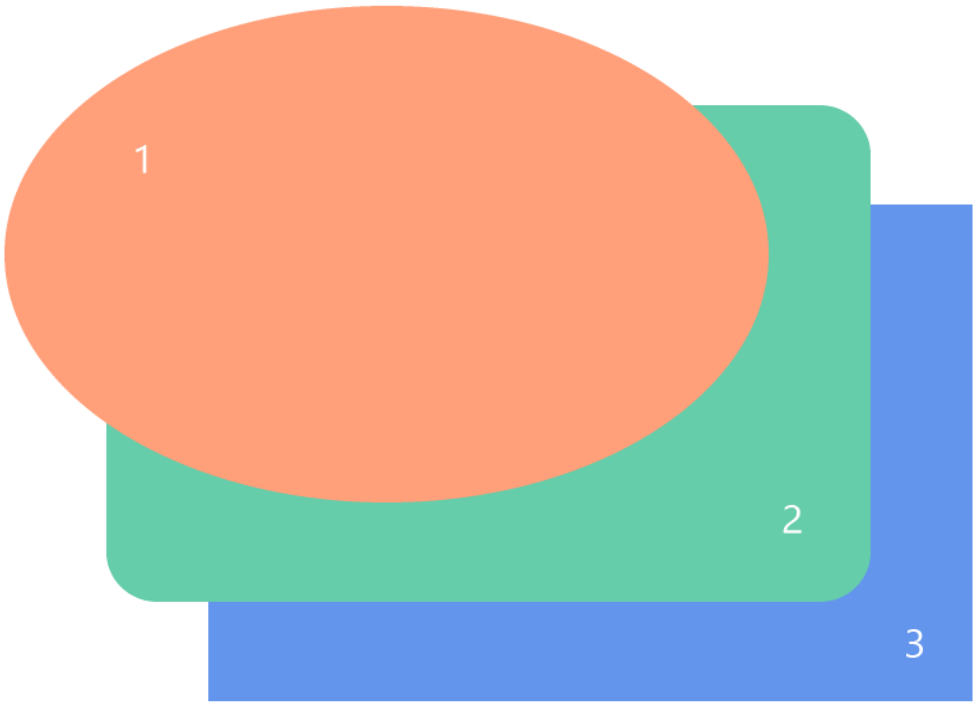
Layers contain Visuals and Spaces. The Space object represents a LayoutRect that may affect the layout of other elements but is never drawn itself. Spaces are not part of the z-hierarchy of visual elements. The Visual class derives from the Space class. Visual class represents an element that will be drawn on the target GcGraphics. The Render method of the Surface class calls the special **Draw** delegate of the Visual and Layer classes (with the **Visible** property set to True) and passes the GcGraphics object and the current item (Layer or Visual) as parameters to the Draw delegate.

Refer to the following example code to draw a complex graphic with some text:

C#

```
// Initialize GcPdfDocument.
var doc = new GcPdfDocument();

// Add a page to the PDF document.
float pageW = 400;
float pageH = 300;
var page = doc.Pages.Add(new SizeF(pageW, pageH));

// Set text format.
var fmt = new TextFormat
{
    FontName = "Segoe UI",
    FontSize = 12f,
    ForeColor = Color.White
};

// Initialize Surface.
var sf = new Surface();

// Create LayoutView.
var view = sf.CreateView(300, 200);

// Create first figure.
var fig1 = view.CreateVisual();
fig1.LayoutRect.AnchorTopLeft(null, 10, 10, 300, 200);
fig1.Draw = (g, v) => {
    g.FillEllipse(v.AsRectF(), Color.LightSalmon);
    g.DrawString("1", fmt, new PointF(50, 50));
};

// Create second figure.
var fig2 = view.CreateVisual((g, v) => {
    g.FillRoundRect(v.AsRectF(), 20, Color.MediumAquamarine);
```

```
    g.DrawString("2", fmt, new PointF(v.Width - 35, v.Height - 45));
});
fig2.LayoutRect.AnchorTopLeft(null, 50, 50, 300, 200);

// Create third figure.
view.CreateVisual((g, v) => {
    g.FillRectangle(v.AsRectF(), Color.CornflowerBlue);
    g.DrawString("3", fmt, new PointF(v.Width - 27, v.Height - 35));
}).LayoutRect.AnchorTopLeft(null, 90, 90, 300, 200);

// Bring the first and second figures to the front.
fig2.BringToFront();
fig1.BringToFront();

// Initialize GcPdfGraphics.
var g = page.Graphics;

// Render the surface.
sf.Render(g);

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



Clipping

Any clipping specified on a Layer object applies to the layer's visuals and the nested layers. The Layer class provides two properties that allow you to define clipping: ClipRect and CreateClipRegion. You can specify just one of these two properties or both. The behavior is different in the three cases:

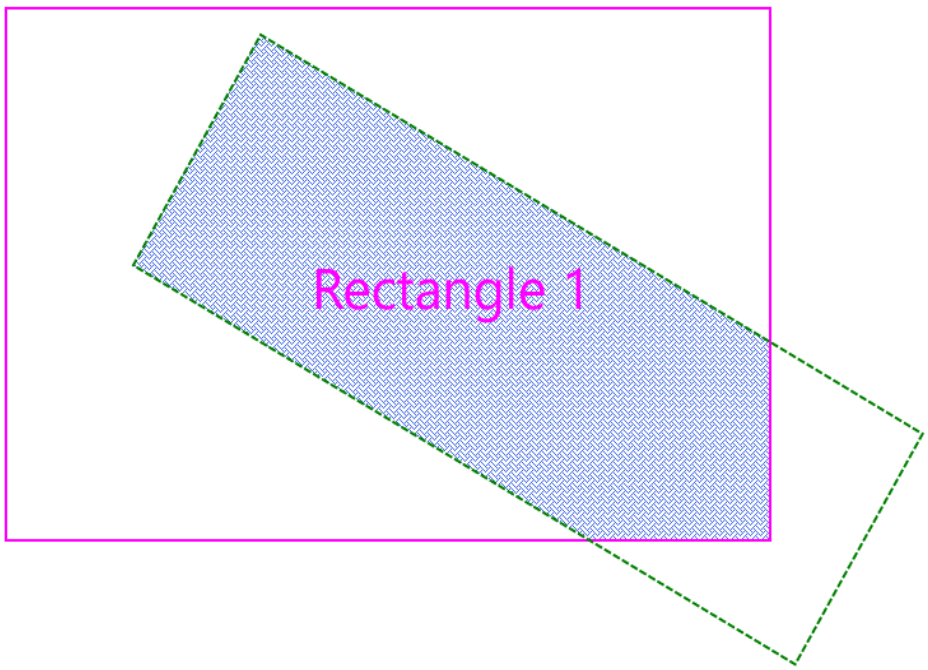
1. If only ClipRect is specified, then LayoutRect value of that property defines the clipping. Note that it can be a

LayoutRect in any View on the same Surface and can have its transformation applied to the corresponding View.

C#

```
// Initialize GcPdfDocument.
var doc = new GcPdfDocument();

// Add a page to the PDF document.
float pageW = 400;
float pageH = 300;
var page = doc.Pages.Add(new SizeF(pageW, pageH));

// Initialize Surface.
var sf = new Surface();

// Create first LayoutView.
var view = sf.CreateView(1, 1);

// Create first sub-layer.
var nestedLayer1 = view.CreateSubLayer();

// Create first figure.
var rect = nestedLayer1.CreateVisual((g, v) => {
    g.DrawRectangle(v.AsRectF(), new Pen(Color.Magenta, 1));
    g.DrawString("Rectangle 1", new TextFormat
    {
        FontName = "Segoe UI",
        FontSize = 16f,
        ForeColor = Color.Magenta
    }, new PointF(120, 90));
});
rect.LayoutRect.AnchorTopLeft(null, 20, 20, 300, 200);

// Create second sub-layer.
var nestedLayer2 = view.CreateSubLayer();

// Create second figure.
nestedLayer2.CreateVisual((g, v) => {
    g.FillRectangle(v.AsRectF(), new HatchBrush(HatchStyle.Weave)
    {
        BackColor = Color.White,
        ForeColor = Color.RoyalBlue
    });
}).LayoutRect.AnchorExact(rect.LayoutRect);

// Create second LayoutView.
var view2 = sf.CreateView(1, 1).Translate(120, 30).Rotate(30);

// Create clipping region.
var clipRect = view2.CreateVisual((g, v) => {
    g.DrawRectangle(v.AsRectF(), Color.Green, 1, DashStyle.Dash);
}).LayoutRect;
```

```
clipRect.AnchorTopLeft(null, 0, 0, 300, 100);

// Set clipping region.
nestedLayer2.ClipRect = clipRect;

// Bring first sub-layer to front.
nestedLayer1.BringToFront();

// Initialize GcPdfGraphics.
var g = page.Graphics;

// Render the surface.
sf.Render(g);

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

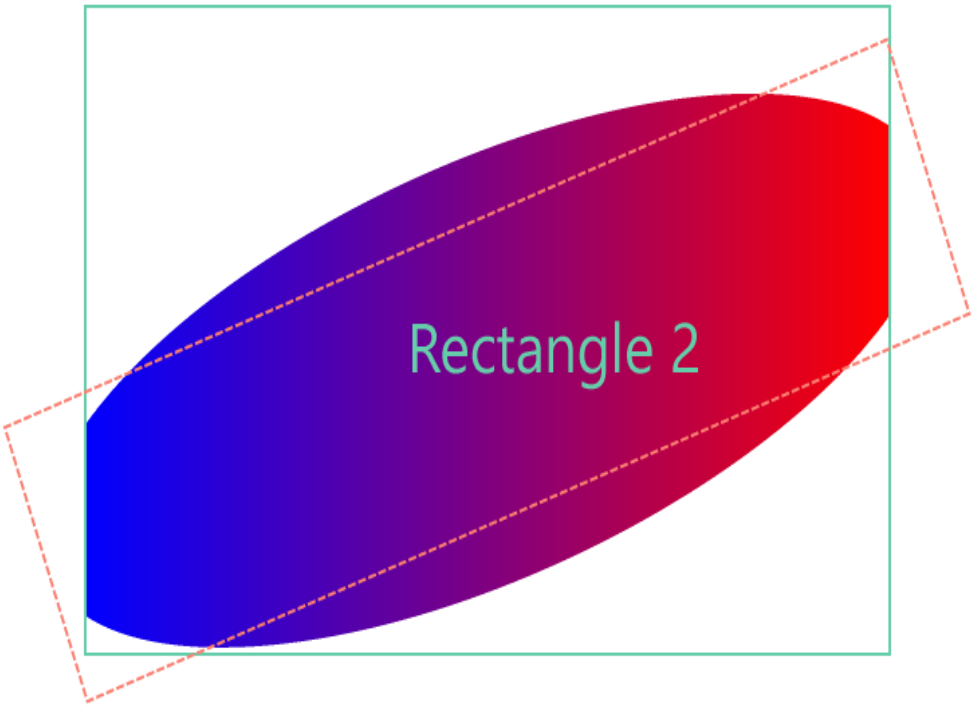


2. If only CreateClipRegion delegate is specified, then GcGraphics.PushClip(clipRegion) applies the clip region returned by the delegate to the graphics before drawing the layer. In this case, the clip region is defined in the layer's own coordinate system without additional transformations. Using CreateClipRegion delegate allows you to set a non-rectangular clipping area. You can create an arbitrary path, then a clipping region based on that path, and return it from the delegate.
3. If both ClipRect and CreateClipRegion properties are specified, then the clip region is defined in the coordinate system of the LayoutRect specified by the ClipRect property. The top left corner of LayoutRect becomes the origin, with axes directed right and down along its sides. Similar to the first case, LayoutRect can be from any View, and its transformation does not depend on the transformation of the layer to be clipped. Then the returned clip region is applied in the coordinate system defined by ClipRect by calling CreateClipRegion delegate. After applying the clip region, objects on the layer are drawn in the layer's coordinate system, while the clipping remains transformed by the ClipRect and CreateClipRegion. This approach simplifies creating complex clipping scenarios. For example, to create a rotated elliptical clipping, you can return an unrotated elliptical region from the CreateClipRegion delegate and rotate it using the ClipRect defined transformation.

C#

```
// Initialize GcPdfDocument.
var doc = new GcPdfDocument();

// Add a page to the PDF document.
float pageW = 600;
float pageH = 600;
var page = doc.Pages.Add(new SizeF(pageW, pageH));

// Initialize Surface.
var sf = new Surface();

// Create first LayoutView.
var view = sf.CreateView(1, 1);

// Create first sub-layer.
var nestedLayer1 = view.CreateSubLayer();

// Create first figure.
var rect = nestedLayer1.CreateVisual((g, v) =>
{
    g.DrawRectangle(v.AsRectF(), new Pen(Color.MediumAquamarine, 1));
    g.DrawString("Rectangle 2", new TextFormat
    {
        FontName = "Segoe UI",
        FontSize = 16f,
        ForeColor = Color.MediumAquamarine
    }, new PointF(120, 90));
});
rect.LayoutRect.AnchorTopLeft(null, 10, 40, 300, 200);

// Create second sub-layer.
var nestedLayer2 = view.CreateSubLayer();

// Create second figure.
nestedLayer2.CreateVisual((g, v) =>
{
    var lgb = new LinearGradientBrush(Color.Blue, Color.Red);
    g.FillRectangle(v.AsRectF(), lgb);
}).LayoutRect.AnchorExact(rect.LayoutRect);

// Create second LayoutView.
var view2 = sf.CreateView(1, 1).Translate(10, 140).Rotate(-20);
var clipRect = view2.CreateVisual((g, v) =>
{
    g.DrawRectangle(v.AsRectF(), Color.Salmon, 1, DashStyle.Dash);
}).LayoutRect;
clipRect.AnchorTopLeft(null, 0, 0, 350, 90);

// Create clipping region.
nestedLayer2.ClipRect = clipRect;
```

```
nestedLayer2.CreateClipRegion = (g, layer) =>
{
    var path = (GcBitmapGraphics.Path)g.CreatePath();
    var rc = layer.ClipRect.AsRectF();
    rc.Inflate(0, 20);
    path.PathBuilder.AddFigure(new EllipticFigure(rc));
    return g.CreateClipRegion(path);
};
nestedLayer2.SendToBack();

// Initialize GcBitmap.
using var bmp = new GcBitmap(380 * 2, 230 * 2, true);
using (var g = bmp.CreateGraphics(Color.White))
{
    g.Transform = Matrix3x2.CreateScale(2);

    // Render the surface.
    sf.Render(g);
}

// Initialize GcPdfGraphics.
var gr = page.Graphics;

// Draw the image.
var image = bmp;
gr.DrawImage(image, new RectangleF(30.6F, 30.7F, 40.8F, 100.9F), null,
ImageAlign.Default);

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



Anchor Points

Layer class provides CreateVisual method that creates a Visual that is not associated with a LayoutRect. The position and size of the Visual are calculated based on one or several anchor points.

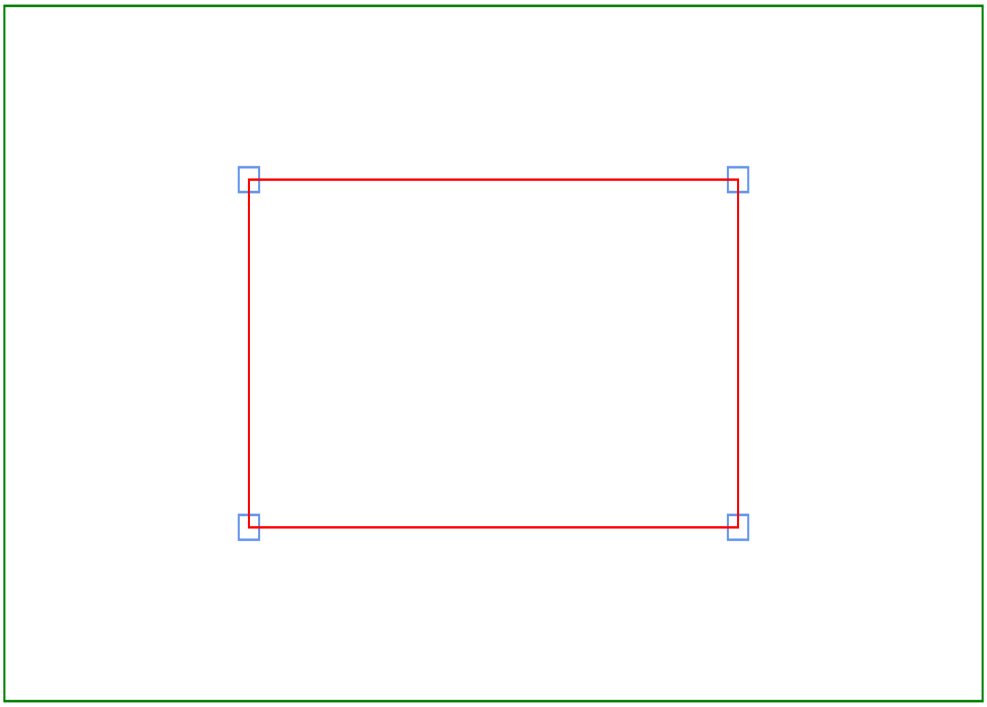
The anchor points assigned to AnchorPoints property of Visual class are recalculated to the View coordinate system and saved to Points property of Visual class before executing the Draw delegate of Visual class.

Refer to the following example code to draw a rectangle relative to anchor points:

C#

```
// Initialize GcPdfDocument.
var doc = new GcPdfDocument();

// Add a page to the PDF document.
float pageW = 400;
float pageH = 300;
var page = doc.Pages.Add(new SizeF(pageW, pageH));

// Initialize Surface.
var sf = new Surface();

// Create LayoutView.
var view = sf.CreateView(pageW, pageH);

// Create margin rectangle.
var marginRect = view.CreateVisual((g, v) => {
    g.DrawRectangle(v.AsRectF(), new Pen(Color.Green));
}).LayoutRect;
marginRect.AnchorDeflate(null, 10);
```

```
// Create points.
var ap1 = marginRect.CreatePoint(0.25f, 0.25f);
var ap2 = marginRect.CreatePoint(0.75f, 0.25f);
var ap3 = marginRect.CreatePoint(0.75f, 0.75f);
var ap4 = marginRect.CreatePoint(0.25f, 0.75f);

var bluePen = new Pen(Color.CornflowerBlue);

// Create anchor points.
view.CreateVisual(new AnchorPoint[] { ap1 }, DrawAP);
view.CreateVisual(new AnchorPoint[] { ap2 }, DrawAP);
view.CreateVisual(new AnchorPoint[] { ap3 }, DrawAP);
view.CreateVisual(new AnchorPoint[] { ap4 }, DrawAP);

// Draw anchor points.
void DrawAP(GcGraphics g, Visual v)
{
    var pt = v.Points[0];
    g.DrawRectangle(new RectangleF(pt.X - 5, pt.Y - 5, 10, 10), bluePen);
}

// Draw polygon through the anchor points.
view.CreateVisual(new AnchorPoint[] { ap1, ap2, ap3, ap4 },
(g, v) => {
    g.DrawPolygon(v.Points, new Pen(Color.Red));
});

// Initialize GcPdfGraphics.
var g = page.Graphics;

// Render the surface.
sf.Render(g);

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



Contours

A Contour can be visualized similarly to anchor points. The Contour and **AnchorPoints** properties of Visual class are mutually exclusive; assigning both properties to non-empty values causes an exception when drawing the Surface. The Contour points are converted to regular points via Points property of Visual class before executing the Draw delegate. Then, you can use methods such as DrawPolygon of GcGraphics class to render the contour from the Draw delegate.

You may encounter a situation where there are several curves and you need to fill the space between them. The Draw delegate of Layer class enables you to fill the space between several contours. Each Contour can be associated with a separate Visual object on the same View or Layer. You can obtain an array of all visuals from the Draw delegate using GetVisuals method of Layer class. You can create a graphics path and add multiple contours as separate figures using the values of Points property of Visual class.

Refer to the following example code to draw multiple contours and fill the space between them by drawing multiple rectangles:

C#

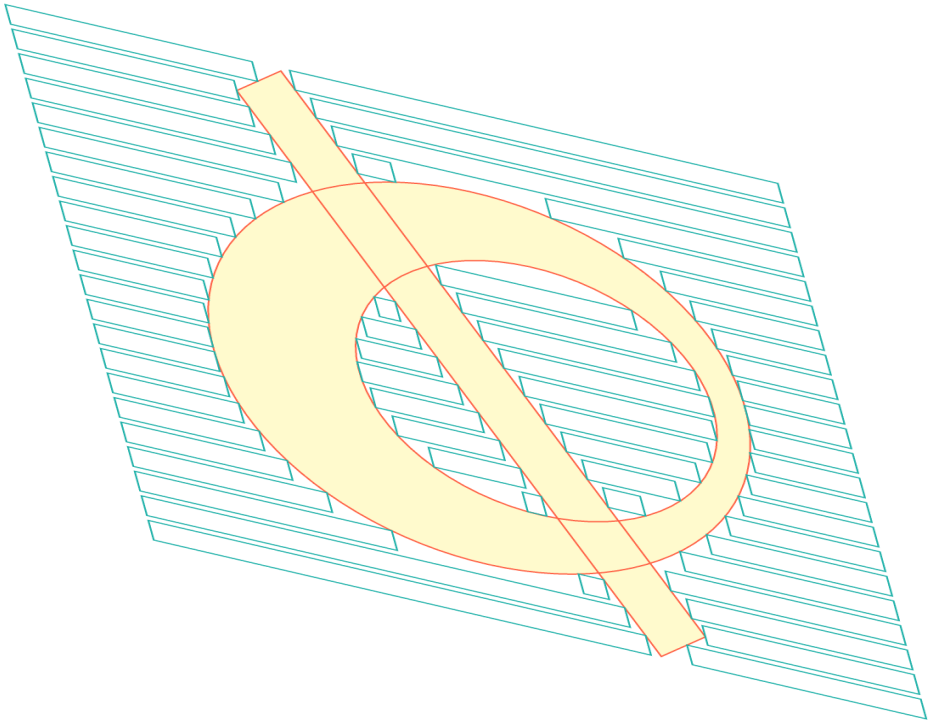
```
    // Initialize GcPdfDocument.
    var doc = new GcPdfDocument();

    // Add a page to the PDF document.
    float pageW = 600;
    float pageH = 570;
    var page = doc.Pages.Add(new SizeF(pageW, pageH));

    // Initialize Surface.
    var sf = new Surface();

    // Create bar contour.
    var c1 = CreateBarContour(sf);
```

```
// Create donut contour.
var (c2, c3) = CreateDonutContours(sf);

// Create first LayoutView.
var view = sf.CreateView(1, 1, (g, l) =>
{
    using var path = g.CreatePath();
    path.SetFillMode(FillMode.Winding);
    var figures = l.GetVisuals();
    for (int i = 0; i < figures.Length; i++)
    {
        var pts = figures[i].Points;
        path.BeginFigure(pts[0]);
        for (int j = 1; j < pts.Length; j++)
            path.AddLine(pts[j]);
        path.EndFigure(FigureEnd.Closed);
    }
    g.FillPath(path, Color.LemonChiffon);
    g.DrawPath(path, Color.Tomato, 1f);
});

// Create Visuals.
view.CreateVisual(c1, false);
view.CreateVisual(c2, false);
view.CreateVisual(c3, false);

// Create second LayoutView.
var view2 = sf.CreateView(1, 1).Translate(-90, -20).Skew(30, 0).Rotate(20);

// Draw rectangles.
float top = 0f;
var pen = new Pen(Color.LightSeaGreen);
for (int i = 0; i < 22; i++)
{
    DrawRects(view2, pen, top, c1, c2, c3);
    top += 20f;
}

// Initialize GcPdfGraphics.
var g = page.Graphics;

// Render the surface.
sf.Render(g);

// Save the PDF document.
doc.Save("Contours.pdf");

// Create bar contour.
static Contour CreateBarContour(Surface surf)
{
    // Create layout view for bar contour.
```

```
        var fig = surf.CreateView(1, 1).Translate(160, 80).Rotate(-30);

        // Create rectangular space.
        var sp = fig.CreateSpace().LayoutRect;
        sp.AnchorTopLeft(null, 0f, 0f, 30, 500);

        // Create contour.
        var c = fig.LayoutView.CreateContour();

        // Create points to anchor.
        c.AddPoints(new AnchorPoint[]
        {
sp.CreatePoint(0, 0),
sp.CreatePoint(1, 0),
sp.CreatePoint(1, 1),
sp.CreatePoint(0, 1)
        });
        return c;
    }

    // Create donut contour.
    static (Contour, Contour) CreateDonutContours(Surface surf)
    {
        // Create layout view for donut contour.
        var fig = surf.CreateView(1, 1).Translate(30, 100).Skew(20, 0);

        // Set dimensions of the donut contour.
        float rMax = 150;
        float xOffsetMax = 200;
        float yOffsetMax = 200;
        float rMin = 100;
        float xOffsetMin = 230;
        float yOffsetMin = 210;
        int nMax = 100;
        int nMin = 70;

        var maxPoints = new List<AnchorPoint>(nMax);
        var minPoints = new List<AnchorPoint>(nMin);
        double deltaMax = Math.PI * 2 / nMax;
        double deltaMin = Math.PI * (-2) / nMin;
        var lv = fig.LayoutView;

        for (int i = 0; i < nMax; i++)
        {
            double alpha = deltaMax * i;
            float x = (float)(Math.Cos(alpha) * rMax + xOffsetMax);
            float y = (float)(Math.Sin(alpha) * rMax + yOffsetMax);
            maxPoints.Add(lv.CreatePoint(0f, 0f, x, y));
        }
        for (int i = 0; i < nMin; i++)
        {
```

```
            double alpha = deltaMin * i;
            float x = (float)(Math.Cos(alpha) * rMin + xOffsetMin);
            float y = (float)(Math.Sin(alpha) * rMin + yOffsetMin);
            minPoints.Add(lv.CreatePoint(0f, 0f, x, y));
        }

        // Create contours.
        var c1 = lv.CreateContour();
        c1.AddPoints(maxPoints);

        var c2 = lv.CreateContour();
        c2.AddPoints(minPoints);

        return (c1, c2);
    }

    // Draw rectangles.
    static void DrawRects(View view, Pen pen, float top, params Contour[] contours)
    {
        LayoutRect? rPrev = null;
        while (true)
        {
            var r0 = view.CreateVisual((g, v) => {
                g.DrawRectangle(v.AsRectF(), pen);
            }).LayoutRect;
            if (rPrev is null)
                r0.AnchorTopLeft(null, top, 100);
            else
            {
                r0.SetTop(null, AnchorParam.Top, top);
                r0.SetLeft(rPrev, AnchorParam.Right);
            }
            r0.SetHeight(16);
            r0.AppendMaxRight(null, AnchorParam.Left, 500);

            var r1 = view.CreateSpace().LayoutRect;
            r1.SetTop(null, AnchorParam.Top, top);
            r1.SetHeight(16);
            r1.SetLeft(r0, AnchorParam.Right);
            r1.AppendMaxRight(null, AnchorParam.Left, 500);

            foreach (var c in contours)
            {
                r0.AppendMaxRight(c, ContourPosition.FirstInOutside);
                r1.AppendMaxRight(c, ContourPosition.NextOutOutside);
            }
            view.Surface.PerformLayout();
            if (r1.Width > 0f)
                rPrev = r1;
            else
            {
```

```
                ((Space)r1.Tag).Detach();
                break;
            }
        }
    }
```



Tables

Drawing tables is a common task when creating documents in PDF, JPEG, SVG, and other formats. Creating a table requires calculating the position of the cells, size of the cells, position of the table, size of the table, etc., but calculating all these parameters manually consumes a lot of effort and time. DsPdf provides a TableRenderer class in **GrapeCity.Documents.Drawing namespace** that allows you to draw the table without having to think much about the size of table columns, merged cells, the layout of rotated text auto-fitting, etc.

DsPdf's layout engine handles the task of automatically calculating all the complex details of cell and table resizing and positioning; you just need to provide information about the desired layout, style, and content. The LayoutHost is the layout engine's core object that is required to instantiate views and calculate the layout. A LayoutHost creates LayoutView which can be defined as a fixed-width and fixed-height rectangle. Each view has its own transformation matrix. The LayoutView can be thought of as a transformed surface with an origin (zero point), two axes (X and Y), and base dimensions (Width and Height). LayoutView places rectangles (LayoutRect objects) whose sides are always parallel to the LayoutView's X or Y axis. Based on the constraints specified, the layout engine calculates the size and position of those rectangles relative to the owner LayoutView.

A LayoutView is created to place a table in it, and the LayoutView is the same size as the PDF page to remove the necessity of any additional transformations. A table is always contained within a rectangle (LayoutRect). The exact size of that rectangle is unknown and may vary depending on the contents of the table. However, we must fix at least one table corner (or two or four corners if necessary).

The TableRenderer class is built on top of the layout engine described in Layouts. All table rows, columns, vertical and horizontal grid lines, cells, and cell text have the associated LayoutRect objects available through the object model. The grid lines are individual rectangles with their own width and height. A table cell is also more than just the intersection of a table column and row. Table cells are added to the grid as separate objects on top of the columns

and rows. The same grid cell may contain regular, background, and foreground table cells. Hence, TableRenderer is a useful tool for drawing tables of any complexity by combining the described flexibility with the power of the layout engine.

TableRenderer represents an immutable table. You cannot add more rows to an existing table or split it into two parts; however, you can create a new TableRenderer instance or more instances with the desired number of rows. You can use an instance of TableRenderer for measuring a table without actually drawing it.

Create Table

The TableRenderer constructor accepts multiple parameters. The following table lists the parameters accepted by TableRenderer constructor:

Parameter	Description
graphics	This parameter specifies the GcGraphics object that will be used to draw the table after it has been constructed.
tableRect	This parameter is the table's LayoutRect.
fixedSides	This parameter specifies which sides of a table are fixed.
rowCount	This parameter specifies the overall number of rows in the table. The table will have at least one row.
columnCount	This parameter specifies the overall number of columns in the table. The table will have at least one column.
gridLineColor	This parameter specifies the color of grid lines.
gridLineWidth	This parameter specifies the thickness of the table grid lines by default. Use the SetVerticalGridLineWidth and SetHorizontalGridLineWidth methods for applying custom thickness to individual grid lines).

The constructor also has optional parameters to specify row minimum height and column minimum width, in case they are not defined manually. The constructor can also set padding for all sides relative to the outer table frame.

The column width and row height do not have to be integers. You can apply star sizes to table columns if both the left and right sides of the table are fixed. It is possible to mix star width, fixed width, and auto width columns in the same table. To set the explicit column width, use the SetWidth method of LayoutRect. Alternatively, you can add a "minimal width" constraint for auto-sized columns by using the AppendMinWidth method of LayoutRect. Applying minimal width constraints is optional for auto-sized columns with horizontal text (and FixedWidth of CellStyle is set to false). Similar constraints can be applied to table rows. You can assign star heights to table rows if the top and bottom table sides are both fixed.

To create a simple table:

1. Initialize LayoutHost, LayoutView, and LayoutRect class instances to define table size and position. The layout engine automatically calculates the position of the table and cells.
2. Create an instance of the TableRenderer class and set the parameters of the table.
3. Set the star width (proportional width) of the columns using SetStarWidth method.
4. Use PerformLayout method on the LayoutHost, which will calculate the coordinates based on the constraints provided.
5. Use Render method on TableRenderer object, which will draw the table.

C#

```
// Initialize GcPdfDocument.
var doc = new GcPdfDocument();
```

```
// Add a page to the PDF document.
float pageW = 400;
float pageH = 230;
var page = doc.Pages.Add(new SizeF(pageW, pageH));

// Initialize LayoutHost.
var host = new LayoutHost();

// Create LayoutView.
var view = host.CreateView(pageW, pageH);

// Create LayoutRect. Add anchor points.
var rt = view.CreateRect();
rt.AnchorTopLeftRight(null, 36, 36, 36);

// Create an instance of TableRenderer.
var ta = new TableRenderer(
    page.Graphics,
    rt, FixedTableSides.TopLeftRight,
    rowCount: 5,
    columnCount: 4,
    gridLineColor: Color.Coral,
    gridLineWidth: 3,
    rowMinHeight: 30);

// Set the star width (proportional width) of the columns.
var columns = ta.ColumnRects;
columns[0].SetStarWidth(1);
columns[1].SetStarWidth(5);
columns[2].SetStarWidth(2);
columns[3].SetStarWidth(3);

// Calculate the coordinates based on the constraints provided.
host.PerformLayout();

// Draw the table.
ta.Render();

// Save the PDF document.
doc.Save("simple-table.pdf");
```

The output of the above-mentioned example code is shown in the image below:

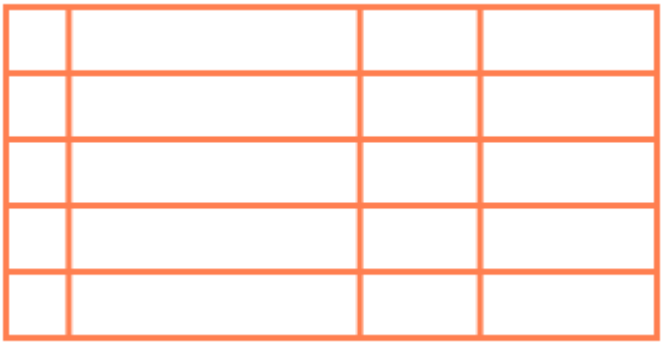


Table Cells

The table cells are separate objects added to the table grid, and one cell can spread to several rows and/or columns. The AddCell method adds a table cell containing text or custom content in the specified position with the default or specified style. The AddCell method also allows you to specify row and column spans. There are three types of table cells: normal (regular), background, and foreground.

The regular cells appear on top of the grid and cannot overlap. For example, if such a cell merges two rows, there will be no grid line between the rows in the cell's column. These cells are added at the specified row or column index using the TableRenderer class's indexer property. If there is at least one regular cell, the grid lines will only be drawn around such cells. The grid lines are not drawn around the gaps that are not covered by regular cells.
The AddMissingCells method ensures that there are no gaps in the grid.

The background cells always appear behind the grid and behind the filling of the regular and foreground cells (if such a fill exists). Background cells can be overlapped by other cells. You can use background cells to highlight some regions in the table. For example, the odd rows may have a different background color. Also, sometimes you may want to display two or more TextLayout objects in the same table cell. One TextLayout will belong to a regular cell, and the others will belong to background cells. If background cells have some text content, their size can be adjusted automatically, as for the regular cells.

The foreground cells appear on top of the background and regular cells. You can use such cells to draw additional elements on top of the grid. Foreground cells can overlap each other and other cells.

Add Cells to Table

To add cells to the table:

1. Add cells to the table using AddCell method which takes CellStyle, row, and column indexes as its arguments.
2. Use AddMissingCells method to fill the gaps in the table with empty regular cells.

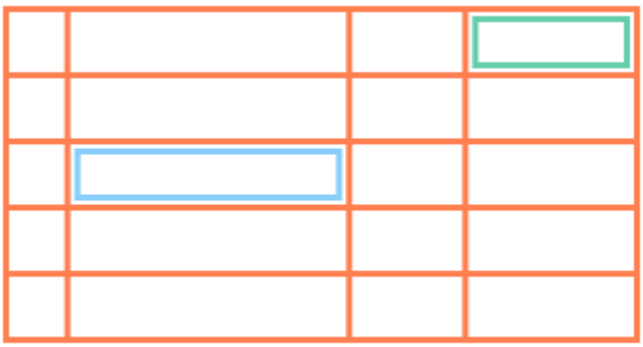
C#

```
// Initialize GcPdfDocument.
var doc = new GcPdfDocument();

// Add a page to the PDF document.
float pageW = 400;
float pageH = 230;
var page = doc.Pages.Add(new SizeF(pageW, pageH));

// Initialize LayoutHost.
var host = new LayoutHost();
```

```
// Create LayoutView.
var view = host.CreateView(pageW, pageH);

// Create LayoutRect. Add anchor points.
var rt = view.CreateRect();
rt.AnchorTopLeftRight(null, 36, 36, 36);

/* Create an instance of TableRenderer.
Pass paddingAll paramater to avoid overlapping of grid lines with the outer table
frame. */
var ta = new TableRenderer(
    page.Graphics,
    rt, FixedTableSides.TopLeftRight,
    rowCount: 5,
    columnCount: 4,
    gridLineColor: Color.Coral,
    gridLineWidth: 3,
    rowMinHeight: 30,
    paddingAll: 5);

// Set the star width (proportional width) of the columns.
var columns = ta.ColumnRects;
columns[0].SetStarWidth(1);
columns[1].SetStarWidth(5);
columns[2].SetStarWidth(2);
columns[3].SetStarWidth(3);

// Set style of the new cell.
var csBlue = new CellStyle
{
    LineColor = Color.LightSkyBlue,
    LineWidth = 3f,
    LinePaddingAll = 2f
};

// Add cell to the table.
ta.AddCell(csBlue, 2, 1);

// Set style of the second new cell.
var csGreen = new CellStyle(csBlue)
{
    LineColor = Color.MediumAquamarine
};

// Add the second cell to the table.
ta.AddCell(csGreen, 0, 3);

// Fill gaps in the table with empty regular cells.
ta.AddMissingCells();

// Draw the table.
```

```
ta.Render();

// Save the PDF document.
doc.Save("add-cells.pdf");
```

The output of the above-mentioned example code is shown in the image below:



Add Data to Cells

DsPdf allows you to add data to the cells by defining the TextFormat property. To add data to the cells:

1. Create a text format using the TextFormat class, then a cell style for normal cells using the CellStyle class. Create a cell style for header text using the CellStyle class, and a header format using the TextFormat class.
2. To add data to cells, use AddCell method and pass the data to be added to cells as one of its parameters.

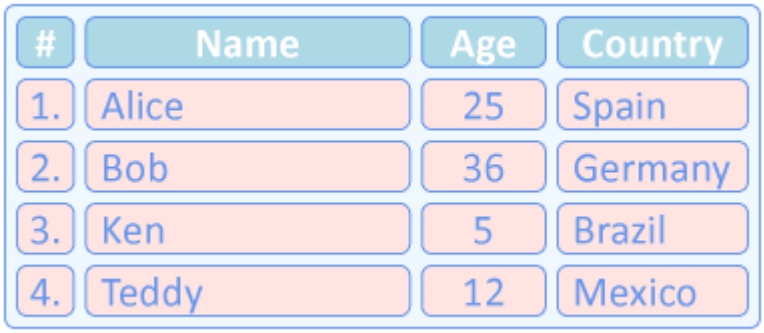
C#

```
// Initialize GcPdfDocument.
var doc = new GcPdfDocument();

// Add a page to the PDF document.
float pageW = 420;
float pageH = 230;
var page = doc.Pages.Add(new SizeF(pageW, pageH));

// Initialize GcPdfGraphics.
var g = page.Graphics;

// Initialize LayoutHost.
var host = new LayoutHost();

// Create LayoutView.
var view = host.CreateView(pageW, pageH);

// Create LayoutRect. Add anchor points.
var rt = view.CreateRect();
rt.AnchorTopLeftRight(null, 36, 36, 36);

// Create an instance of TableRenderer.
var ta = new TableRenderer(g,
    rt, FixedTableSides.TopLeftRight,
```

```
    rowCount: 5,
    columnCount: 4,
    gridLineColor: Color.Empty,
    gridLineWidth: 1,
    rowMinHeight: 30,
    paddingAll: 3)

// Add table frame style.
{
    TableFrameStyle = new FrameStyle
    {
        FillColor = Color.AliceBlue,
        LineColor = Color.CornflowerBlue,
        LineWidth = 1,
        CornerRadius = 5
    }
};

// Set the star width (proportional width) of the columns.
var columns = ta.ColumnRects;
columns[0].SetStarWidth(1);
columns[1].SetStarWidth(5);
columns[2].SetStarWidth(2);
columns[3].SetStarWidth(3);

// Set text format.
var fmt = new TextFormat
{
    FontName = "Calibri",
    ForeColor = Color.CornflowerBlue,
    FontSize = 16
};

// Set cell style for normal text.
var csNormal = new CellStyle
{
    TextFormat = fmt,
    ParagraphAlignment = ParagraphAlignment.Center,
    PaddingLeftRight = 10,
    FillColor = Color.MistyRose,
    LineColor = Color.CornflowerBlue,
    LinePaddingAll = 2,
    LineWidth = 1,
    CornerRadius = 5
};

// Set text alignment to center.
var csCenter = new CellStyle(csNormal)
{
    TextAlignment = TextAlignment.Center,
    PaddingLeftRight = 0,
```

```
};

// Set cell style for table header.
var csHeader = new CellStyle(csCenter)
{
    TextFormat = new TextFormat(fmt)
    {
        ForeColor = Color.White,
        FontBold = true
    },
    FillColor = Color.LightBlue
};

// Add cells to the table with data and cell style.
ta.AddCell(csHeader, 0, 0, "#");
ta.AddCell(csHeader, 0, 1, "Name");
ta.AddCell(csHeader, 0, 2, "Age");
ta.AddCell(csHeader, 0, 3, "Country");

ta.AddCell(csCenter, 1, 0, "1.");
ta.AddCell(csNormal, 1, 1, "Alice");
ta.AddCell(csCenter, 1, 2, "25");
ta.AddCell(csNormal, 1, 3, "Spain");

ta.AddCell(csCenter, 2, 0, "2.");
ta.AddCell(csNormal, 2, 1, "Bob");
ta.AddCell(csCenter, 2, 2, "36");
ta.AddCell(csNormal, 2, 3, "Germany");

ta.AddCell(csCenter, 3, 0, "3.");
ta.AddCell(csNormal, 3, 1, "Ken");
ta.AddCell(csCenter, 3, 2, "5");
ta.AddCell(csNormal, 3, 3, "Brazil");

ta.AddCell(csCenter, 4, 0, "4.");
ta.AddCell(csNormal, 4, 1, "Teddy");
ta.AddCell(csCenter, 4, 2, "12");
ta.AddCell(csNormal, 4, 3, "Mexico");

// Draw the table.
ta.Render();

// Save the PDF document.
doc.Save("add-data-to-cells.pdf");
```

The output of the above-mentioned example code is shown in the image below:



#	Name	Age	Country
1.	Alice	25	Spain
2.	Bob	36	Germany
3.	Ken	5	Brazil
4.	Teddy	12	Mexico

Merge Cells

DsPdf allows you to merge cells by defining the column and row span in the AddCell method. To merge cells:

1. Use DefaultCellStyle property to set the default cell style.
2. Use AddCell method to add the cells to the table and also set the column and row spans while adding cells.
3. Use AddMissingCells method to fill the gaps in the table with empty regular cells.
4. Use ApplyCellConstraints method to calculate the layout of the table and cells.

C#

```
// Initialize GcPdfDocument.
var doc = new GcPdfDocument();

// Add a page to the PDF document.
float pageW = 400;
float pageH = 230;
var page = doc.Pages.Add(new SizeF(pageW, pageH));

// Initialize LayoutHost.
var host = new LayoutHost();

// Create LayoutView.
var view = host.CreateView(pageW, pageH);

// Create LayoutRect. Add anchor points.
var rt = view.CreateRect();
rt.AnchorTopLeftRight(null, 36, 36, 36);

// Create an instance of TableRenderer.
var ta = new TableRenderer(
    page.Graphics,
    rt, FixedTableSides.TopLeftRight,
    rowCount: 5,
    columnCount: 4,
    gridLineColor: Color.Empty,
    gridLineWidth: 1,
    rowMinHeight: 30,
    paddingAll: 3)

// Add table frame style.
```

```
{
    TableFrameStyle = new FrameStyle
    {
        LineColor = Color.CornflowerBlue,
        LineWidth = 1,
        CornerRadius = 5,
        FillColor = Color.AliceBlue
    }
};

// Set the star width (proportional width) of the columns.
var columns = ta.ColumnRects;
columns[0].SetStarWidth(1);
columns[1].SetStarWidth(5);
columns[2].SetStarWidth(2);
columns[3].SetStarWidth(3);

// Set default cell style.
ta.DefaultCellStyle = new CellStyle
{
    LinePaddingAll = 2,
    LineColor = Color.CornflowerBlue,
    LineWidth = 1,
    CornerRadius = 5,
    FillColor = Color.MistyRose
};

// Add cells and set row and column spans.
ta.AddCell(0, 1, 3, 1);
ta.AddCell(3, 0, 1, 4);
ta.AddCell(1, 2, 2, 2);

// Fill gaps in the table with empty regular cells.
ta.AddMissingCells();

// Calculate layout of the grid and cells.
ta.ApplyCellConstraints();

// Draw the table.
ta.Render();

// Save the PDF document.
doc.Save("merge-cells.pdf");
```

The output of the above-mentioned example code is shown in the image below:

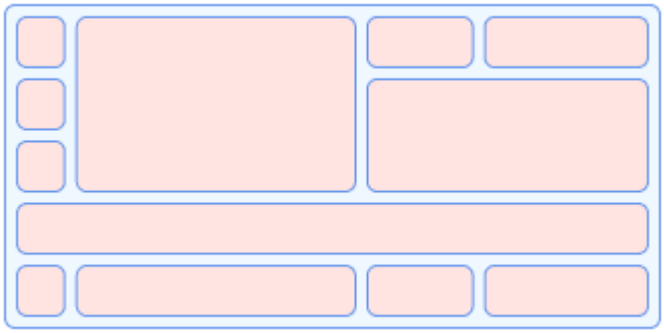


Table and Cell Styling

Each table cell has an associated style (CellStyle) that describes the appearance and behavior of the cell. For example, the Background property in the CellStyle class defines whether the cell is in the background. The FrameStyle class provides several appearance properties, such as FillColor, LineColor, LineWidth, **LinePadding**, CornerRadius, etc., and it is also used to describe the appearance of the outer table frame.

DsPdf allows you to customize the table and cells by adding an outer table frame and inner cell borders. To customize the table and cells:

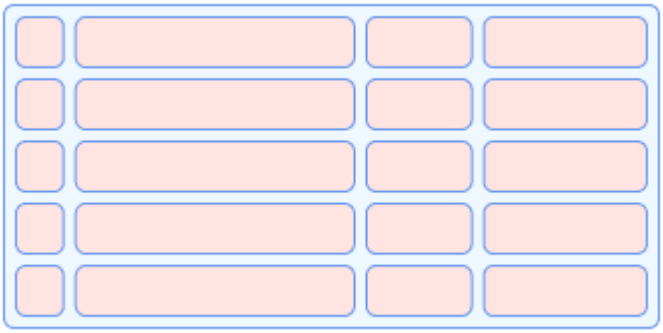
1. Create an outer table frame using TableFrameStyle property.
2. Use AddMissingCells method to add empty cells to the table. Set the padding of the empty cells in the table to make spaces equal using LinePaddingAll property, and also set LineColor and LineWidth to draw inner cell borders.
3. Use ApplyCellConstraints method to calculate the layout of the table and the cells.

C#

```
// Initialize GcPdfDocument.
var doc = new GcPdfDocument();

// Add a page to the PDF document.
float pageW = 400;
float pageH = 230;
var page = doc.Pages.Add(new SizeF(pageW, pageH));

// Initialize LayoutHost.
var host = new LayoutHost();

// Create LayoutView.
var view = host.CreateView(pageW, pageH);

// Create LayoutRect. Add anchor points.
var rt = view.CreateRect();
rt.AnchorTopLeftRight(null, 36, 36, 36);

// Create an instance of TableRenderer.
var ta = new TableRenderer(
    page.Graphics,
    rt, FixedTableSides.TopLeftRight,
    rowCount: 5,
    columnCount: 4,
```

```
    gridLineColor: Color.Empty,
    gridLineWidth: 1,
    rowMinHeight: 30,
    paddingAll: 3)

// Add table frame style.
{
    TableFrameStyle = new FrameStyle
    {
        LineColor = Color.CornflowerBlue,
        LineWidth = 1,
        CornerRadius = 5,
        FillColor = Color.AliceBlue
    }
};

// Set the star width (proportional width) of the columns.
var columns = ta.ColumnRects;
columns[0].SetStarWidth(1);
columns[1].SetStarWidth(5);
columns[2].SetStarWidth(2);
columns[3].SetStarWidth(3);

// Create empty cells.
ta.AddMissingCells(new CellStyle
{
    LinePaddingAll = 2,
    LineColor = Color.CornflowerBlue,
    LineWidth = 1,
    CornerRadius = 5,
    FillColor = Color.MistyRose
});

// Calculate layout of the grid and cells.
ta.ApplyCellConstraints();

// Draw the table.
ta.Render();

// Save the PDF document.
doc.Save("table-cell-customization.pdf");
```

The output of the above-mentioned example code is shown in the image below:



Text Customizations in Cells

The RightToLeft, TextAlignment, ParagraphAlignment, MaxWidth, and MaxHeight properties of CellStyle class resemble the properties of the TextLayout class that set the style for text in a cell. The CellStyle class also has RotationAngle property, which specifies the flow direction of the cell text. If cell content is rotated, then other properties of CellStyle are also defined relative to the current text direction.

The FixedWidth and FixedHeight properties of CellStyle class fix the width and height of the cell. The FixedWidth property is set to true by default, while the FixedHeight property is set to false. These properties work for merged and rotated cells as well.

Position	Nation	Games				Points				Table points
		Played	Won	Drawn	Lost	For	Against	Difference	Tries	
1	England	38	36	26	26	27	40	8	19	53
2	France	20	49	42	32	28	48	35	42	91
3	Ireland	14	39	45	29	18	36	35	40	148
4	Italy	45	37	45	38	29	2	25	5	140
5	Scotland	28	24	49	41	16	5	36	40	116
6	Wales	37	40	36	7	26	49	10	38	85

Refer to the following example code to add text customizations in the cells:

C#

```
// Initialize Team.
var teams = new Team[]
{
    new Team("England"),
    new Team("France"),
    new Team("Ireland"),
    new Team("Italy"),
    new Team("Scotland"),
    new Team("Wales"),
```

```
};

int pageW = 410;
int pageH = 320;

// Initialize GcPdfDocument.
var doc = new GcPdfDocument();

// Add a page to the PDF document.
var page = doc.Pages.Add(new SizeF(pageW, pageH));

// Initialize GcPdfGraphics.
var g = page.Graphics;

// Initialize LayoutHost.
var host = new LayoutHost();

// Create LayoutView.
var view = host.CreateView(pageW, pageH);

// Create LayoutRect. Add anchor points.
var rt = view.CreateRect();
rt.AnchorTopRight(null, 50, 50);

// Create an instance of TableRenderer.
var ta = new TableRenderer(g,
    rt, FixedTableSides.TopRight,
    rowCount: teams.Length + 2,
    columnCount: 11,
    gridLineColor: Color.FromArgb(173, 223, 252),
    gridLineWidth: 1,
    rowMinHeight: 10,
    columnMinWidth: 10);

// Set text format.
var fmt = new TextFormat
{
    FontName = "Tahoma",
    FontSize = 12
};

// Set cell style.
var cs = new CellStyle
{
    TextFormat = fmt,
    FixedWidth = false,
    PaddingAll = 4
};

// Set horizonal cell style for table header.
var csHeaderH = new CellStyle(cs)
```

```
{
    TextFormat = new TextFormat(fmt)
    {
        ForeColor = Color.White,
        FontBold = true
    },
    FillColor = Color.FromArgb(17, 93, 140),
    TextAlignment = TextAlignment.Center,
    ParagraphAlignment = ParagraphAlignment.Center,
};

// Set vertical cell style for table header.
var csHeaderV = new CellStyle(csHeaderH)
{
    RotationAngle = 270,
    TextAlignment = TextAlignment.Leading,
    PaddingLeft = 3
};

// Set cell style for numbers.
var csNumber = new CellStyle(cs)
{
    TextAlignment = TextAlignment.Center
};

// Set cell style for Nation.
var csNation = new CellStyle(cs)
{
    TextFormat = new TextFormat(fmt)
    {
        ForeColor = Color.FromArgb(50, 123, 197)
    },
};

// Add the header cells.
ta.AddCell(csHeaderV, 0, 0, 2, 1, "Position");
ta.AddCell(csHeaderH, 0, 1, 2, 1, "Nation");
ta.AddCell(csHeaderH, 0, 2, 1, 4, "Games");
ta.AddCell(csHeaderV, 1, 2, "Played");
ta.AddCell(csHeaderV, 1, 3, "Won");
ta.AddCell(csHeaderV, 1, 4, "Drawn");
ta.AddCell(csHeaderV, 1, 5, "Lost");
ta.AddCell(csHeaderH, 0, 6, 1, 4, "Points");
ta.AddCell(csHeaderV, 1, 6, "For");
ta.AddCell(csHeaderV, 1, 7, "Against");
ta.AddCell(csHeaderV, 1, 8, "Difference");
ta.AddCell(csHeaderV, 1, 9, "Tries");
ta.AddCell(csHeaderH, 0, 10, 2, 1, "Table\npoints");

// Add the data cells.
for (int i = 0; i < teams.Length; i++)
```

```
{
    var team = teams[i];
    int rowIndex = i + 2;
    ta.AddCell(csNumber, rowIndex, 0, $"{i + 1}");
    ta.AddCell(csNation, rowIndex, 1, team.Nation);
    ta.AddCell(csNumber, rowIndex, 2, $"{team.Played}");
    ta.AddCell(csNumber, rowIndex, 3, $"{team.Won}");
    ta.AddCell(csNumber, rowIndex, 4, $"{team.Drawn}");
    ta.AddCell(csNumber, rowIndex, 5, $"{team.Lost}");
    ta.AddCell(csNumber, rowIndex, 6, $"{team.For}");
    ta.AddCell(csNumber, rowIndex, 7, $"{team.Against}");
    ta.AddCell(csNumber, rowIndex, 8, $"{team.Diff}");
    ta.AddCell(csNumber, rowIndex, 9, $"{team.Tries}");
    ta.AddCell(csNumber, rowIndex, 10, $"{team.TablePoints}");
}

// Change background for odd rows.
ta.DefaultCellStyle = new CellStyle
{
    Background = true,
    FillColor = Color.FromArgb(238, 238, 238)
};
for (int i = 0; i < teams.Length; i += 2)
{
    ta.AddCell(i + 2, 0, 1, 11);
}

// Draw the table.
ta.Render();

// Save the PDF document.
doc.Save("text-customizations.pdf");

// Create the class.
class Team
{
    static readonly Random _rnd = new(24323429);

    public string Nation;
    public int Played, Won, Drawn, Lost;
    public int For, Against, Diff, Tries;
    public int TablePoints;

    internal Team(string nation)
    {
        Nation = nation;
        Played = _rnd.Next(0, 50);
        Won = _rnd.Next(0, 50);
        Drawn = _rnd.Next(0, 50);
        Lost = _rnd.Next(0, 50);
        For = _rnd.Next(0, 50);
```

```
        Against = _rnd.Next(0, 50);
        Diff = _rnd.Next(0, 50);
        Tries = _rnd.Next(0, 50);
        TablePoints = _rnd.Next(0, 150);
    }
}
```

Draw Custom Content in Cells

DsPdf allows you to draw custom content in a cell using CustomDraw property of type delegate from CellStyle class. When a delegate is assigned to this property, it is executed from the Render method of TableRenderer class. The delegate assigned to CustomDraw property accepts two parameters: a GcGraphics object (passed to the Render method) and a TableCell object.

Shape / Color	Circle	Triangle	Rectangle	Oval	Square
Red	Red Circle 	Red Triangle 	Red Rectangle 	Red Oval 	Red Square 
Green	Green Circle 	Green Triangle 	Green Rectangle 	Green Oval 	Green Square 
Blue	Blue Circle 	Blue Triangle 	Blue Rectangle 	Blue Oval 	Blue Square 
Orange	Orange Circle 	Orange Triangle 	Orange Rectangle 	Orange Oval 	Orange Square 
Purple	Purple Circle 	Purple Triangle 	Purple Rectangle 	Purple Oval 	Purple Square 

Refer to the following example code to add custom content in the cells:

C#

```
        // Initialize GcPdfDocument.
        var doc = new GcPdfDocument();

        float pageW = 800;
        float pageH = 500;

        // Add a page to the PDF document.
        var page = doc.Pages.Add(new SizeF(pageW, pageH));

        // Initialize GcPdfGraphics.
        var g = page.Graphics;

        // Initialize LayoutHost.
        var host = new LayoutHost();
```

```
        // Create LayoutView.
        var view = host.CreateView(pageW, pageH);

        // Create LayoutRect. Add anchor points.
        var rt = view.CreateRect();
        rt.AnchorTopLeft(null, 36, 36);

        // Create an instance of TableRenderer.
        var ta = new TableRenderer(g,
            rt, FixedTableSides.TopLeft,
            rowCount: 6, columnCount: 6,
            gridLineColor: Color.Black,
            gridLineWidth: 1);

        // Set height and width of the the rows and columns.
        ta.RowRects[0].SetHeight(50);
        ta.ColumnRects[0].SetWidth(120);

        // Set the cell style.
        var cs = new CellStyle
        {
            TextAlignment = TextAlignment.Center,
            ParagraphAlignment = ParagraphAlignment.Center,

            // Set text format.
            TextFormat = new TextFormat
            {
                FontName = "Calibri",
                FontSize = 16,
                FontSizeInGraphicUnits = true,
                FontBold = true
            }
        };

        // Add a background style for displaying shapes with a custom drawn element
(diagonal line).
        var csCornerTopRight = new CellStyle(cs)
        {
            Background = true,
            LineWidth = 1f,
            Borders = FrameBorders.MainDiagonal,
            TextAlignment = TextAlignment.Trailing,
            ParagraphAlignment = ParagraphAlignment.Near,
            PaddingRight = 10,
            PaddingTop = 5
        };

        // Add a regular style for displaying color at the bottom left corner of the
same cell.
        var csCornerBottomLeft = new CellStyle(cs)
        {
```

```
    TextAlignment = TextAlignment.Leading,
    ParagraphAlignment = ParagraphAlignment.Far,
    PaddingLeft = 10,
    PaddingBottom = 5
};

// Add a background cell at the top left corner.
ta.AddCell(csCornerTopRight, 0, 0, "Shape");

// Add row header cells.
ta.AddCell(cs, 0, 1, "Circle");
ta.AddCell(cs, 0, 2, "Triangle");
ta.AddCell(cs, 0, 3, "Rectangle");
ta.AddCell(cs, 0, 4, "Oval");
ta.AddCell(cs, 0, 5, "Square");

// Add a regular cell at the top left corner.
ta.AddCell(csCornerBottomLeft, 0, 0, "Color");

// Add column header cells.
ta.AddCell(cs, 1, 0, "Red");
ta.AddCell(cs, 2, 0, "Green");
ta.AddCell(cs, 3, 0, "Blue");
ta.AddCell(cs, 4, 0, "Orange");
ta.AddCell(cs, 5, 0, "Purple");

// Add default cell style.
ta.DefaultCellStyle = new CellStyle
{
    PaddingTop = 3,
    PaddingLeftRight = 20,
    PaddingBottom = 55,
    FixedWidth = false,
    TextAlignment = TextAlignment.Center,
    TextFormat = new TextFormat
    {
        FontName = "Calibri",
        FontSize = 14
    },

    // Set text layout.
    CreateTextLayout = (g, cs, data) =>
    {
        var tl = g.CreateTextLayout();
        tl.Append(((Figure)data).Title, cs.TextFormat);
        return tl;
    },

    // Draw the custom content into the cells.
    CustomDraw = (g, tc) =>
    {
```

```
                ((Figure)tc.Data).Draw(g, tc.Width, tc.Height);
            }
        };

        // Add data cells.
        ta.AddCell(1, 1, new Figure("Red Circle", Shape.Circle, Color.Red));
        ta.AddCell(1, 2, new Figure("Red Triangle", Shape.Triangle, Color.Red));
        ta.AddCell(1, 3, new Figure("Red Rectangle", Shape.Rectangle, Color.Red));
        ta.AddCell(1, 4, new Figure("Red Oval", Shape.Oval, Color.Red));
        ta.AddCell(1, 5, new Figure("Red Square", Shape.Square, Color.Red));

        ta.AddCell(2, 1, new Figure("Green Circle", Shape.Circle, Color.Green));
        ta.AddCell(2, 2, new Figure("Green Triangle", Shape.Triangle, Color.Green));
        ta.AddCell(2, 3, new Figure("Green Rectangle", Shape.Rectangle,
Color.Green));
        ta.AddCell(2, 4, new Figure("Green Oval", Shape.Oval, Color.Green));
        ta.AddCell(2, 5, new Figure("Green Square", Shape.Square, Color.Green));

        ta.AddCell(3, 1, new Figure("Blue Circle", Shape.Circle, Color.Blue));
        ta.AddCell(3, 2, new Figure("Blue Triangle", Shape.Triangle, Color.Blue));
        ta.AddCell(3, 3, new Figure("Blue Rectangle", Shape.Rectangle, Color.Blue));
        ta.AddCell(3, 4, new Figure("Blue Oval", Shape.Oval, Color.Blue));
        ta.AddCell(3, 5, new Figure("Blue Square", Shape.Square, Color.Blue));

        ta.AddCell(4, 1, new Figure("Orange Circle", Shape.Circle, Color.Orange));
        ta.AddCell(4, 2, new Figure("Orange Triangle", Shape.Triangle,
Color.Orange));
        ta.AddCell(4, 3, new Figure("Orange Rectangle", Shape.Rectangle,
Color.Orange));
        ta.AddCell(4, 4, new Figure("Orange Oval", Shape.Oval, Color.Orange));
        ta.AddCell(4, 5, new Figure("Orange Square", Shape.Square, Color.Orange));

        ta.AddCell(5, 1, new Figure("Purple Circle", Shape.Circle, Color.Purple));
        ta.AddCell(5, 2, new Figure("Purple Triangle", Shape.Triangle,
Color.Purple));
        ta.AddCell(5, 3, new Figure("Purple Rectangle", Shape.Rectangle,
Color.Purple));
        ta.AddCell(5, 4, new Figure("Purple Oval", Shape.Oval, Color.Purple));
        ta.AddCell(5, 5, new Figure("Purple Square", Shape.Square, Color.Purple));

        // Draw the table.
        ta.Render();

        // Save the PDF document.
        doc.Save("custom-content.pdf");
    }
    // Create enum for Shape.
    enum Shape
    {
        Circle,
        Triangle,
```

```
    Rectangle,
    Oval,
    Square
}

// Create a class for Figure.
class Figure
{
    public string Title;
    public Shape Shape;
    public Color Color;

    public Figure(string title, Shape shape, Color color)
    {
        Title = title;
        Shape = shape;
        Color = color;
    }

    public void Draw(GcGraphics g, float w, float h)
    {
        RectangleF rc;
        var pen = new Pen(Color.Black, 1);
        switch (Shape)
        {
            case Shape.Circle:
                rc = new RectangleF(w / 2 - 20, h - 50, 40, 40);
                g.FillEllipse(rc, Color);
                g.DrawEllipse(rc, pen);
                break;
            case Shape.Triangle:
                var points = new PointF[]
                {
            new(w / 2, h - 50),
            new(w / 2 + 25, h - 10),
            new(w / 2 - 25, h - 10)
                };
                g.FillPolygon(points, Color);
                g.DrawPolygon(points, pen);
                break;
            case Shape.Rectangle:
                rc = new RectangleF(w / 2 - 35, h - 50, 70, 40);
                g.FillRectangle(rc, Color);
                g.DrawRectangle(rc, pen);
                break;
            case Shape.Oval:
                rc = new RectangleF(w / 2 - 35, h - 50, 70, 40);
                g.FillEllipse(rc, Color);
                g.DrawEllipse(rc, pen);
                break;
            case Shape.Square:
```

```
                    rc = new RectangleF(w / 2 - 20, h - 50, 40, 40);
                    g.FillRectangle(rc, Color);
                    g.DrawRectangle(rc, pen);
                    break;
                }
            }
        }
}
```

Nested Tables

Nested tables are those in which a table (or tables) is drawn inside another table, where the larger table works as a container for the smaller table. The nested tables work as a way for you to organize images or text in evenly spaced cells. The nested tables can also be helpful in grouping different sets of data. DsPdf allows you to create nested tables and add and customize the data in the nested tables by creating different objects of LayoutView and LayoutRect.

Fire Diamond

Standard Representation	Tabular Representation
	**Risk levels of hazardous materials in this facility**

Health Risk	Flammability	Reactivity	Special
Level 3	Level 2	Level 1	

Refer to the following example code to draw nested tables:

C#

```
// Initialize page width and height.
int pageW = 800;
int pageH = 300;

// Initialize GcPdfDocument.
var doc = new GcPdfDocument();

// Add a page to the PDF document.
var page = doc.Pages.Add(new SizeF(pageW, pageH));

// Initialize GcPdfGraphics.
var g = page.Graphics;

// Initialize LayoutHost.
var host = new LayoutHost();

// Create first table. The view is rotated 45 degrees counterclockwise.
```

```
var view1 = host.CreateView(100, 100, Matrix.CreateRotation(-Math.PI / 4));
var rect1 = view1.CreateRect();

// Coincide the rect1 with the view.
rect1.AnchorExact(null);

// Create an instance of TableRenderer for the first table.
var ta1 = new TableRenderer(g, rect1, FixedTableSides.All,
    rowCount: 2, columnCount: 2, Color.Black, gridLineWidth: 2);

// Set height and width of the the rows and columns.
var columns = ta1.ColumnRects;
columns[0].SetStarWidth(1);
columns[1].SetStarWidth(1);

ta1.RowRects[0].SetStarHeight(1);
ta1.RowRects[1].SetStarHeight(1);

// Set text format.
var fmt1 = new TextFormat
{
    FontName = "Arial",
    FontSize = 32,
    FontSizeInGraphicUnits = true
};

// Set cell style.
var cs1 = new CellStyle
{
    CustomDraw = (g, tc) =>
    {
        var tl = g.CreateTextLayout();
        tl.Append((string)tc.Data, fmt1);
        tl.PerformLayout();
        var rc = tl.ContentRectangle;
        var m = g.Transform;

        // display the number centered and rotated 45 degrees clockwise
        m = Matrix3x2.CreateTranslation(tc.Width * 0.5f, tc.Height * 0.5f) * m;
        g.Transform = Matrix3x2.CreateRotation((float)Math.PI / 4) * m;
        g.DrawTextLayout(tl, new PointF(-rc.Width * 0.5f, -rc.Height * 0.5f));
    }
};

// Add the cells.
ta1.AddCell(new CellStyle(cs1)
{
    FillColor = Color.FromArgb(102, 145, 255)
}, 0, 0).Data = "3";

ta1.AddCell(new CellStyle(cs1)
```

```
{
    FillColor = Color.FromArgb(255, 102, 102)
}, 0, 1).Data = "2";

ta1.AddCell(new CellStyle(cs1)
{
    FillColor = Color.FromArgb(252, 255, 102)
}, 1, 1).Data = "1";

ta1.AddCell(new CellStyle { FillColor = Color.White }, 1, 0);

// Calculate the layout of the table and the cells.
ta1.ApplyCellConstraints();

/* Shift the view down. The Y-coordinate of the top right point (P1)
is zero, and all other coordinates are not negative.*/
var p = view1.Transform.Transform(rect1.P1);
view1.ApplyOffset(null, 0f, -p.Y);

/* The bottom right point (P3) is at the far right,
which can be used to calculate the maximum width.*/
p = view1.Transform.Transform(rect1.P3);
float w1 = p.X;

/* The bottom left point (P2) is at the bottommost position,
   which can be used to calculate the maximum height.*/
p = view1.Transform.Transform(rect1.P2);
float h1 = p.Y;

// Create second table.
var view2 = host.CreateView(0, 0);
var rect2 = view2.CreateRect();

// Anchor the second table to top left corner.
rect2.AnchorTopLeft(null, 0, 0);

// Create an instance of TableRenderer for the second table.
var ta2 = new TableRenderer(g, rect2, FixedTableSides.TopLeft,
    rowCount: 2, columnCount: 4,
    gridLineColor: Color.FromArgb(162, 169, 177),
    gridLineWidth: 2);

// Set text format.
var fmt = new TextFormat
{
    FontName = "Calibri",
    FontSize = 24,
    FontSizeInGraphicUnits = true
};
var fmtBold = new TextFormat(fmt)
{
```

```
    FontBold = true
};

// Set cell style.
var cs = new CellStyle
{
    TextFormat = fmt,
    PaddingAll = 6,
    TextAlignment = TextAlignment.Center,
    FixedWidth = false
};

// Set default cell style.
ta2.DefaultCellStyle = new CellStyle(cs)
{
    TextFormat = fmtBold,
    FillColor = Color.FromArgb(234, 236, 240)
};

// Add cells to the second table with data.
ta2.AddCell(0, 0, "Health Risk");
ta2.AddCell(0, 1, "Flammability");
ta2.AddCell(0, 2, "Reactivity");
ta2.AddCell(0, 3, "Special");

ta2.AddCell(cs, 1, 0, "Level 3");
ta2.AddCell(cs, 1, 1, "Level 2");
ta2.AddCell(cs, 1, 2, "Level 1");
ta2.AddCell(cs, 1, 3);

// Calculate the layout of the table and the cells.
ta2.ApplyCellConstraints();

float w2 = rect2.Width;
float h2 = rect2.Height;

// Create a third table, which will be larger and the outer table.
var view3 = host.CreateView(1, 1);
var rect3 = view3.CreateRect();

// Anchor the third table to top left corner.
rect3.AnchorTopLeft(null, 5, 20);

// Create an instance of TableRenderer for the third table.
var ta3 = new TableRenderer(g, rect3, FixedTableSides.TopLeft,
    rowCount: 3, columnCount: 3,
    gridLineColor: Color.FromArgb(162, 169, 177),
    gridLineWidth: 2);

const float CellPaddingX = 6f;
const float CellPaddingY = 20f;
```

```
const float TextTableGap = 6f;

// Set the width of the columns.
columns = ta3.ColumnRects;
columns[0].AppendMinWidth(CellPaddingX * 2 + w1);
columns[1].SetWidth(12);
columns[2].AppendMinWidth(CellPaddingX * 2 + w2);

// Set the height of the row.
ta3.RowRects[2].AppendMinHeight(CellPaddingY * 2 + h1);

// Add cells to the table with data.
ta3.AddCell(new CellStyle(cs)
{
    TextFormat = fmtBold,
    Background = true
}, 0, 0, 1, 3, "Fire Diamond");

ta3.AddCell(cs, 1, 0, "Standard Representation");
ta3.AddCell(cs, 1, 2, "Tabular Representation");

// Add the first table to the outer table.
ta3.AddCell(new CellStyle
{
    // Draw the first table.
    CustomDraw = (g, tc) =>
    {
        float x = (tc.Width - w1) * 0.5f;
        float y = (tc.Height - h1) * 0.5f;
        g.Transform = Matrix3x2.CreateTranslation(x, y) * g.Transform;
        ta1.Render(g);
    }
}, 2, 0);

// Add the second table to the outer table.
ta3.AddCell(new CellStyle(cs)
{
    TextFormat = fmtBold,
    ParagraphAlignment = ParagraphAlignment.Center,
    PaddingTop = CellPaddingY,
    PaddingBottom = TextTableGap + h2 + CellPaddingY,
    // Draw the second table.
    CustomDraw = (g, tc) =>
    {
        var tl = tc.TextLayout;
        float y = CellPaddingY + tl.ContentY + tl.ContentHeight + TextTableGap;
        float x = (tc.Width - w2) * 0.5f;
        g.Transform = Matrix3x2.CreateTranslation(x, y) * g.Transform;
        ta2.Render(g);
    }
}, 2, 2, "Risk levels of hazardous materials in this facility");
```

```
// Fill gaps in the table with empty regular cells.
ta3.AddMissingCells(1, 0, 2, 3);

ta3.AddCell(new CellStyle
{
    FillColor = Color.FromArgb(248, 249, 250),
    Background = true
}, 1, 0, 2, 3);

// Draw the third table.
ta3.Render();

// Save the PDF document.
doc.Save("nested-table.pdf");
```

Limitations

A table created with TableRenderer is immutable. The number of rows and columns must be known before calling the TableRenderer constructor. To split the table between multiple pages, calculate the layout for the huge table first, then recreate the layout for a subset of rows to fit the available space on each page. Moreover, text having an "East Asian" font and aligned vertically can only be displayed in cells with both FixedWidth and FixedHeight set to true in the CellStyle.

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 DsPdf 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. DsPdf 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 EncyptionAlgorithm
    std.EncryptionAlgorithm = EncryptionAlgorithm.RC4;
    std.EncryptionKeyLength = 128;
    // Set the EncryptHandler property.
    doc.Security.EncryptHandler = std;
    // Render text using DrawString method
    g.DrawString("Welcome", new TextFormat()
    { Font = StandardFonts.TimesBold, FontSize = 12 }, new PointF(In, In));
    // Save document
    doc.Save(stream);
}
```

Back to Top

DsPdf 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", 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 DsPdf 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", new TextFormat()
    { Font = StandardFonts.TimesBold, FontSize = 12 }, new PointF(In, In));

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

Back to Top

Open Encrypted PDF Without Password

Sometimes you might want to load encrypted PDF files without specifying the password. DsPdf allows you to open a password-protected PDF file without specifying the password using Load**(Stream stream, DecryptionOptions decryptionOptions)** overload of GcPdfDocument class after setting the ThrowExceptionIfInvalidPassword property (flag) of DecryptionOptions class to false (by default, it is true). While there are many limitations to what you can do with a PDF opened in this way, you can:

- Read and write properties that are not based on PDF strings or streams.
- Add new objects if they can be defined without using PDF strings or streams.

The PDF format has the "object stream" feature. An object stream is a stream object in which a sequence of indirect objects is stored, rather than at the outermost file level. In the case of encrypted PDF, the object stream is also encrypted so that the PDF objects it contains are not accessible. As a result, DsPdf cannot open the PDF files with the object stream function without knowing the password, and raises an exception. When working with password-protected PDFs without specifying the password, you cannot:

- Read or write PDF strings.

- Change the security handler.
- Read or write properties based on PDF string objects. Typically, those are string properties, such as DocumentInfo.Creator.
- Read or write stream objects, which means you cannot:
  - Render PDF
  - Add pages with content.
  - Change the content of existing pages.
  - Embed fonts
  - Create or edit annotation appearance streams.

The following sections demonstrate some useful scenarios when working with encrypted PDF files without specifying the password.

Add Annotation

C#

```
using (FileStream fs = new FileStream("Sample.pdf", FileMode.Open, FileAccess.Read,
FileShare.Read))
{
    // Initialize GcPdfDocument.
    GcPdfDocument doc = new GcPdfDocument();

    // Load the PDF file and set the decryption options to false.
    doc.Load(fs, new DecryptionOptions() { ThrowExceptionIfInvalidPassword = false,
ThrowExceptionIfUnsupportedSecurityOptions = false });

    // Set first page of PDF file as active page.
    var page = doc.Pages[0];

    // Get page size.
    var pageSize = page.Size;

    // Add square annotation.
    SquareAnnotation sa = new SquareAnnotation();
    sa.Page = page;

    /* Remove the UserName it is initialized by default and will cause an exception
       when the document is saved because strings cannot be encrypted. */
    sa.UserName = null;

    // Add square to the page.
    sa.Rect = new RectangleF(10, 10, pageSize.Width - 20, pageSize.Height - 20);

    // Set color of the square.
    sa.Color = Color.Red;

    // Save the PDF file.
    doc.Save("Annotation.pdf");
}
```

Get Metadata Values

C#

```
using (FileStream fs = new FileStream("Sample.pdf", FileMode.Open, FileAccess.Read,
FileShare.Read))
{
    // Initialize GcPdfDocument.
    GcPdfDocument doc = new GcPdfDocument();

    // Load the PDF file and set the decryption options to false.
    doc.Load(fs, new DecryptionOptions() { ThrowExceptionIfInvalidPassword = false,
ThrowExceptionIfUnsupportedSecurityOptions = false });

    // Check if the metadata is encrypted or not, and if not, change the metadata.
    bool encryptMetadata = true;
    if (doc.Security.EncryptHandler is StandardSecurityHandlerRev4 ssh)
        encryptMetadata = ssh.EncryptMetadata;

    if (!encryptMetadata)
    {
        // Metadata is not encrypted.
        Metadata m = doc.Metadata;
        Console.WriteLine("The document has not encrypted metadata:");
        Console.WriteLine($"CreatorTool: {m.CreatorTool}");
        Console.WriteLine($"CreateDate: {m.CreateDate}");
    }
    else
    {
        Console.WriteLine("The document metadata is ENCRYPTED");
    }
}
```

Change Metadata Values

C#

```
using (FileStream fs = new FileStream("Sample.pdf", FileMode.Open, FileAccess.Read,
FileShare.Read))
{
    // Initialize GcPdfDocument.
    GcPdfDocument doc = new GcPdfDocument();

    // Load the PDF file and set the decryption options to false.
    doc.Load(fs, new DecryptionOptions() { ThrowExceptionIfInvalidPassword = false,
ThrowExceptionIfUnsupportedSecurityOptions = false });

    // Check if the metadata is encrypted or not, and if not, change the metadata.
    bool encryptMetadata = true;
    if (doc.Security.EncryptHandler is StandardSecurityHandlerRev4 ssh)
        encryptMetadata = ssh.EncryptMetadata;

    if (!encryptMetadata)
    {
```

```
        // Metadata is not encrypted.
        Metadata m = doc.Metadata;
        Console.WriteLine("The document has not encrypted metadata:");
        Console.WriteLine($"CreatorTool: {m.CreatorTool}");
        Console.WriteLine($"CreateDate: {m.CreateDate}");

        // Change the creator tool value.
        m.CreatorTool = "New value of CreatorTool";

        doc.Save(@"MetaData.pdf");
    }
    else
    {
        Console.WriteLine("The document metadata is ENCRYPTED");
    }
}
```

Change Page Order

C#

```
using (FileStream fs = new FileStream("Sample.pdf", FileMode.Open, FileAccess.Read,
FileShare.Read))
{
    // Initialize GcPdfDocument.
    GcPdfDocument doc = new GcPdfDocument();

    // Load the PDF file and set the decryption options to false.
    doc.Load(fs, new DecryptionOptions() { ThrowExceptionIfInvalidPassword = false,
ThrowExceptionIfUnsupportedSecurityOptions = false });

    // Swap the first and second pages.
    doc.Pages.Swap(0, 1);

    // Save the PDF file.
    doc.Save("PageOrder.pdf");
}
```

Change Value of CheckBoxField and RadioButtonField

C#

```
using (FileStream fs = new FileStream("Sample.pdf", FileMode.Open, FileAccess.Read,
FileShare.Read))
{
    // Initialize GcPdfDocument.
    GcPdfDocument doc = new GcPdfDocument();

    // Load the PDF file and set the decryption options to false.
    doc.Load(fs, new DecryptionOptions() { ThrowExceptionIfInvalidPassword = false,
ThrowExceptionIfUnsupportedSecurityOptions = false });
```

```
    // Change the value of CheckBoxField.
    var cbf = (CheckBoxField)doc.AcroForm.Fields[0];
    cbf.Checked = true;

    // Change the value of RadioButtonField.
    var rbf = (RadioButtonField)doc.AcroForm.Fields[1];
    var values = rbf.GetCheckedAppearanceStreamNames();
    rbf.Value = values[0];

    // Save the PDF file.
    doc.Save("FieldValue.pdf");
}
```

Change Values of TextField and CombTextField

C#

```
using (FileStream fs = new FileStream("Sample.pdf", FileMode.Open, FileAccess.Read,
FileShare.Read))
{
    // Initialize GcPdfDocument.
    GcPdfDocument doc = new GcPdfDocument();

    // Load the PDF file and set the decryption options to false.
    doc.Load(fs, new DecryptionOptions() { ThrowExceptionIfInvalidPassword = false,
ThrowExceptionIfUnsupportedSecurityOptions = false });

    // Change the value of CheckBoxField.
    var tf = (TextField)doc.AcroForm.Fields[2];
    tf.PdfValue = new PdfName("New Value");

    // Change the value of CombTextField.
    var ctf = (CombTextField)doc.AcroForm.Fields[3];
    ctf.PdfValue = new PdfName("NEW");

    // Set the NeedAppearances entry of AcroForm.
    doc.AcroForm.Set(PdfName.Std.NeedAppearances, PdfBool.True);

    // Save the PDF file.
    doc.Save("TextField.pdf");
}
```

Get Statistics

C#

```
using (FileStream fs = new FileStream("Sample.pdf", FileMode.Open, FileAccess.Read,
FileShare.Read))
{
    // Initialize GcPdfDocument.
    GcPdfDocument doc = new GcPdfDocument();
```

```
    // Load the PDF file and set the decryption options to false.
    doc.Load(fs, new DecryptionOptions() { ThrowExceptionIfInvalidPassword = false,
ThrowExceptionIfUnsupportedSecurityOptions = false });

    // Get the page count and size of the pages.
    Console.WriteLine($"Page count: {doc.Pages.Count}");
    Console.WriteLine();
    foreach (Page page in doc.Pages)
    {
        var sz = page.GetRenderSize();
        Console.WriteLine($"Size of {page.Index} page: {sz.Width}x{sz.Height}");
    }
}
```

Note: When using DsPdf without a license key, working with a password-protected PDF without specifying the password has many additional limitations (e.g., any attempt to change and save the PDF will fail due to the addition of the license nag text). You can ask for an evaluation license key by sending an email to us.sales@mescius.com and evaluate the product for 30 days without any limitations.

Back to Top

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

Digital Signature

DsPdf 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. DsPdf supports legal stamps created by trustworthy authority like the Time Stamp Authority (TSA). DsPdf 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. DsPdf 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 DsPdf, 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, DsPdf 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#

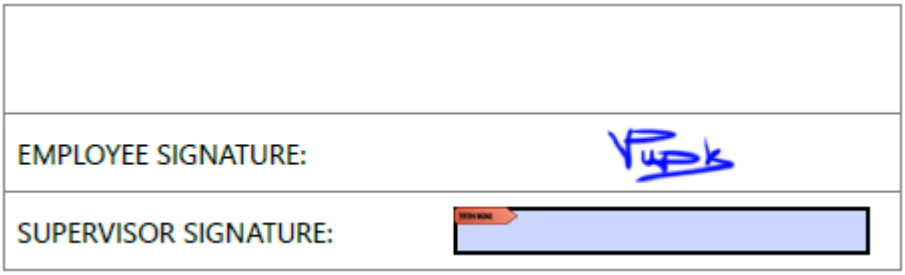
```
public static void CreatePDF(Stream stream)
{
    GcPdfDocument doc = new GcPdfDocument();
    Page page = doc.NewPage();
    TextFormat tf = new TextFormat() { Font = StandardFonts.Times, FontSize = 14
};
    page.Graphics.DrawString(
        "Hello, World!\r\nSigned below by DsPdfWeb SignDoc sample." +
        "\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", "DsPdfTest.pfx");
    X509Certificate2 cert = new X509Certificate2(File.ReadAllBytes(pfxPath),
"qq",
    X509KeyStorageFlags.MachineKeySet | X509KeyStorageFlags.PersistKeySet
    | X509KeyStorageFlags.Exportable);
    SignatureProperties sp = new SignatureProperties();
    sp.Certificate = cert;
    sp.Location = "DsPdfWeb Sample Browser";
    sp.SignerName = "DsPdfWeb";
    // Add timestamp
    sp.TimeStamp = new TimeStamp("https://freetsa.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 DsPdf, 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

DsPdf 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

DsPdf 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.

Sign Document using Custom Time-Stamp Tokens

You can create custom time-stamp tokens and sign a document using them by implementing the ITimeStampGenerator interface and assigning it to the **TimeStamp** property of SignatureProperties and TimeStampProperties classes. ITimeStampGenerator interface defines the methods for generating time-stamp tokens.

Refer to the following example code to add a custom time-stamp token and a signature with a custom time-stamp token to the PDF document:

C#

```
// Create a custom time-stamp generator.
public class TimeStampGenerator : ITimeStampGenerator
{
    public string ServerUrl;
    public string UserName;
    public string Password;
    public OID HashAlgorithm;

    public TimeStampGenerator()
    {
        HashAlgorithm = new OID("2.16.840.1.101.3.4.2.1", "SHA256");
    }
    public TimeStampGenerator(string serverUrl, string userName, string password, OID
hashAlgorithm)
    {
        ServerUrl = serverUrl;
        UserName = userName;
        Password = password;
        HashAlgorithm = hashAlgorithm;
    }

    private static long CopyStream(Stream src, Stream dst, bool
useSingleWriteOperation = false)
    {
        byte[] buffer = new byte[16 * 1024];
        int bytesRead;
        long result = 0;
        while ((bytesRead = src.Read(buffer, 0, buffer.Length)) != 0)
        {
            dst.Write(buffer, 0, bytesRead);
            result += bytesRead;
        }
        return result;
    }

    private static void Update(IDigest dgst, byte[] input)
    {
```

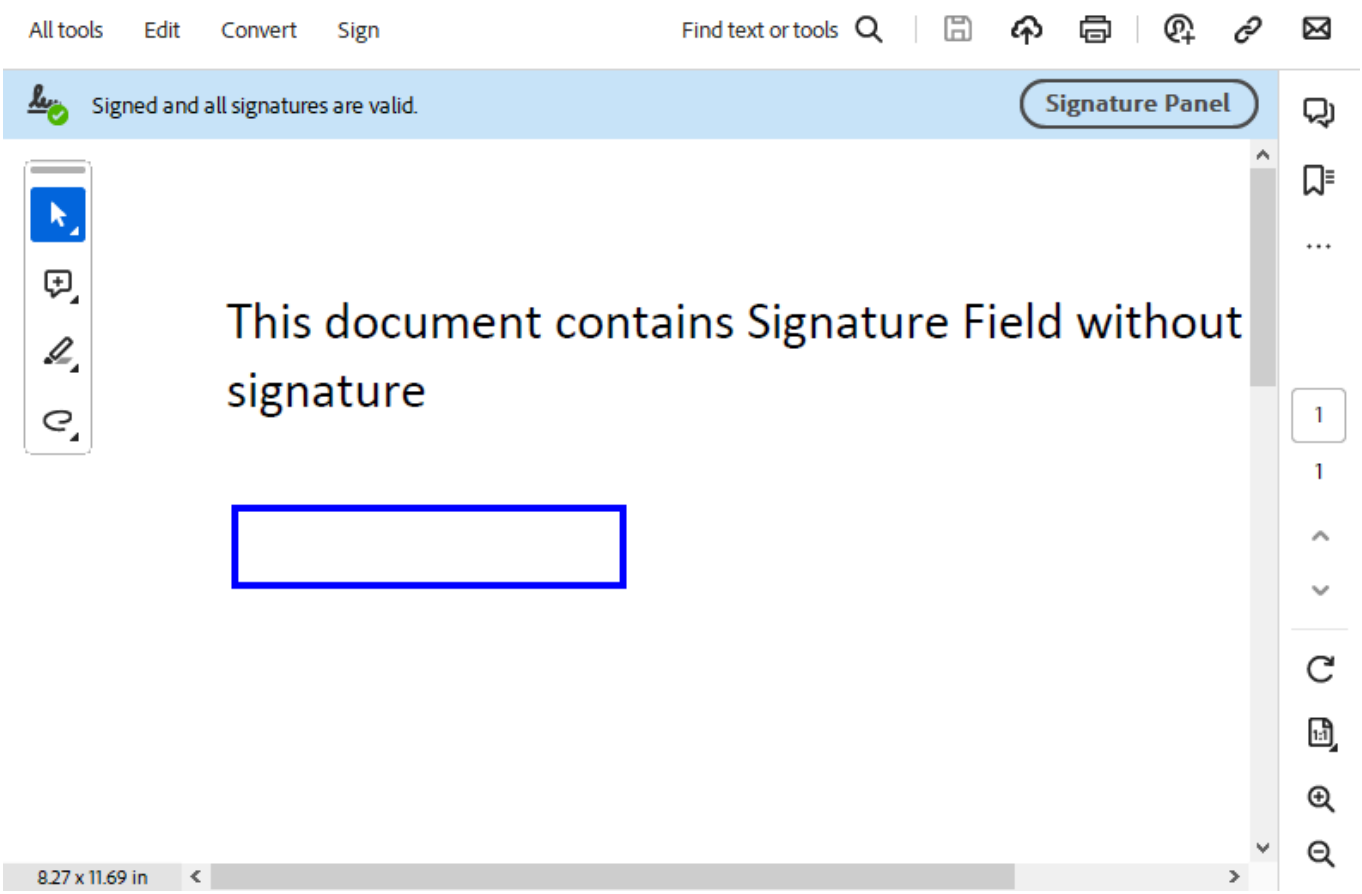
```
        Update(dgst, input, 0, input.Length);
    }

    private static void Update(IDigest dgst, byte[] input, int offset, int len)
    {
        dgst.BlockUpdate(input, offset, len);
    }

    private static byte[] Digest(IDigest dgst)
    {
        byte[] output = new byte[dgst.GetDigestSize()];
        dgst.DoFinal(output, 0);
        return output;
    }

    private static byte[] Digest(IDigest dgst, byte[] input)
    {
        Update(dgst, input);
        return Digest(dgst);
    }

    private static byte[] Digest(IDigest dgst, Stream data)
    {
        byte[] buf = new byte[8192];
        int n;
        while ((n = data.Read(buf, 0, buf.Length)) > 0)
        {
            Update(dgst, buf, 0, n);
        }
        return Digest(dgst);
    }

    private static IDigest GetMessageDigest(OID hashAlgorithm)
    {
        if (hashAlgorithm == OID.HashAlgorithms.MD2)
            return new MD2Digest();
        else if (hashAlgorithm == OID.HashAlgorithms.MD5)
            return new MD5Digest();
        else if (hashAlgorithm == OID.HashAlgorithms.SHA1)
            return new Sha1Digest();
        else if (hashAlgorithm == OID.HashAlgorithms.SHA224)
            return new Sha224Digest();
        else if (hashAlgorithm == OID.HashAlgorithms.SHA256)
            return new Sha256Digest();
        else if (hashAlgorithm == OID.HashAlgorithms.SHA384)
            return new Sha384Digest();
        else if (hashAlgorithm == OID.HashAlgorithms.SHA512)
            return new Sha512Digest();
        else if (hashAlgorithm == OID.HashAlgorithms.RIPEMD128)
            return new RipeMD128Digest();
        else if (hashAlgorithm == OID.HashAlgorithms.RIPEMD160)
```

```
            return new RipeMD160Digest();
        else if (hashAlgorithm == OID.HashAlgorithms.RIPEMD256)
            return new RipeMD256Digest();
        else if (hashAlgorithm == OID.HashAlgorithms.GOST3411)
            return new Gost3411Digest();
        throw new ArgumentException();
    }

    private byte[] GetTsaResponseForUserRequest(byte[] requestBytes)
    {
        HttpWebRequest con;
        try
        {
            con = (HttpWebRequest)WebRequest.Create(ServerUrl);
        }
        catch (Exception e)
        {
            throw new Exception(string.Format("Failed to get response from TSA server
{0}.", ServerUrl), e);
        }

        con.ContentLength = requestBytes.Length;
        con.ContentType = "application/timestamp-query";
        con.Method = "POST";
        if (!string.IsNullOrEmpty(UserName))
        {
            string authInfo = UserName + ":" + Password;
            authInfo = Convert.ToBase64String(Encoding.UTF8.GetBytes(authInfo));
            con.Headers["Authorization"] = "Basic " + authInfo;
        }

        using (Stream outp = con.GetRequestStream())
            outp.Write(requestBytes, 0, requestBytes.Length);

        HttpWebResponse httpWebResponse = (HttpWebResponse)con.GetResponse();
        using (Stream stream = httpWebResponse.GetResponseStream())
        {
            if (stream == null)
                return null;

            using (MemoryStream ms = new MemoryStream())
            {
                CopyStream(stream, ms);
                string encoding =
httpWebResponse.Headers[HttpResponseHeader.ContentEncoding];
                byte[] data = ms.ToArray();
                if (string.Compare(encoding, "base64",
StringComparison.InvariantCultureIgnoreCase) == 0)
                    data = Convert.FromBase64String(Encoding.ASCII.GetString(data));
                return data;
            }
```

```
        }
    }

    // Get time stamp token for hash.
    private byte[] GetTimeStampTokenForHash(byte[] hash)
    {
        // Setup the time stamp request.
        TimeStampRequestGenerator tsqGenerator = new TimeStampRequestGenerator();
        tsqGenerator.SetCertReq(true);

        // Generate random number.
        BigInteger nonce = BigInteger.ValueOf(unchecked((int)DateTime.Now.Ticks) +
Environment.TickCount);
        TimeStampRequest request = tsqGenerator.Generate(new
DerObjectIdentifier(HashAlgorithm.ID), hash, nonce);

        // Call the communications layer.
        byte[] requestBytes = request.GetEncoded();
        byte[] respBytes = GetTsaResponseForUserRequest(requestBytes);

        // Handle the TSA response.
        TimeStampResponse response = new TimeStampResponse(respBytes);

        // Validate communication level attributes (RFC 3161 PKIStatus).
        response.Validate(request);
        PkiFailureInfo failure = response.GetFailInfo();
        int value = failure == null ? 0 : failure.IntValue;
        if (value != 0)
            throw new Exception(string.Format("Invalid TSA response from {0}: {1}.",
ServerUrl, response.GetStatusString()));

        // Extract just the time stamp token (removes communication status info).
        TimeStampToken tsToken = response.TimeStampToken;
        if (tsToken == null)
            throw new Exception(string.Format("No timetoken in TSA response from
{0}.", ServerUrl));

        return tsToken.GetEncoded();
    }

    // Get time stamp token.
    public byte[] GetTimeStampToken(byte[] data)
    {
        // Build hash of the data.
        byte[] hash = Digest(GetMessageDigest(HashAlgorithm), data);
        return GetTimeStampTokenForHash(hash);
    }

    public byte[] GetTimeStampToken(Stream stream)
    {
        // Build hash of the data.
```

```
        byte[] hash = Digest(GetMessageDigest(HashAlgorithm), stream);
        return GetTimeStampTokenForHash(hash);
    }
}

internal class Program
{
    static void Main(string[] args)
    {
        // Generate the signature with time-stamp token.
        X509Certificate2 crt = new X509Certificate2(@"..\..\..\User.pfx", "User12");
        using (FileStream fs = new
FileStream(@"..\..\..\AdobePDFWithEmptySignatureField.pdf", FileMode.Open))
        {
            // Initialize GcPdfDocument.
            GcPdfDocument doc = new GcPdfDocument();

            // Load the PDF document from stream.
            doc.Load(fs);

            // Initialize SignatureProperties.
            SignatureProperties sp = new SignatureProperties();

            // Build a chain of certificates.
            sp.SignatureBuilder = new Pkcs7SignatureBuilder()
            {
                CertificateChain = new X509Certificate2[] { crt },
            };

            // Add custom time stamp.
            sp.TimeStamp = new TimeStampGenerator()
            {
                ServerUrl = @"http://ts.ssl.com",
            };

            // Add signature to signature field.
            sp.SignatureField = doc.AcroForm.Fields[0];

            // Sign PDF document.
            doc.Sign(sp, "signed.pdf");
        }

        // Generate a document with time-stamp.
        using (FileStream fs = new
FileStream(@"..\..\..\AdobePDFWithEmptySignatureField.pdf", FileMode.Open))
        {
            // Initialize GcPdfDocument.
            GcPdfDocument doc = new GcPdfDocument();

            // Load the PDF document from stream.
            doc.Load(fs);
```

```
            // Initialize TimeStampProperties.
            TimeStampProperties tsp = new TimeStampProperties();

            // Add custom time stamp.
            tsp.TimeStamp = new TimeStampGenerator()
            {
                ServerUrl = @"http://ts.ssl.com",
            };

            // Add time stamp to signature field.
            tsp.SignatureField = doc.AcroForm.Fields[0];

            // Add time stamp and save the document.
            doc.TimeStamp(tsp, "timestamp.pdf");
        }
    }
}
```



PDF Advanced Electronic Signatures

DsPdf 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 DsPdf:

- 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 DsPdf, 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 DsPdf, 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://freetsa.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", "DsPdfTest.pfx");
    var cert = new X509Certificate2(pfxPath, "qq");
    var sp = SignatureProperties.CreatePAdES_B_T(new
TimeStamp("https://freetsa.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)doc.AcroForm.Fields[0];
    var sig = signField.Value;
    var vp = new DocumentSecurityStore.VerificationParams();
    var pfxPath = Path.Combine("CACertCertificate.pfx");
    var cert = new X509Certificate2(pfxPath, "1234");
    vp.Certificates = new X509Certificate2[] { cert };
    if (!doc.SecurityStore.AddVerification(sig, vp))
        throw new Exception($"Could not add verification for {sig.Name}.");
    doc.Save("SignedLTV.pdf", 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", "DsPdfTest.pfx");
    var cert = new X509Certificate2(pfxPath, "qq");
    var sp = SignatureProperties.CreatePAdES_B_T(new
TimeStamp("https://freetsa.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)doc.AcroForm.Fields[0];
    var sig = signField.Value;
    var vp = new DocumentSecurityStore.VerificationParams();
    var pfxPath = Path.Combine("CACertCertificate.pfx");
    var cert = new X509Certificate2(pfxPath, "1234");
    vp.Certificates = new X509Certificate2[] { cert };
    if (!doc.SecurityStore.AddVerification(sig, vp))
        throw new Exception($"Could not add verification for {sig.Name}.");
    doc.Save("SignedLTV.pdf", 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;
}
```

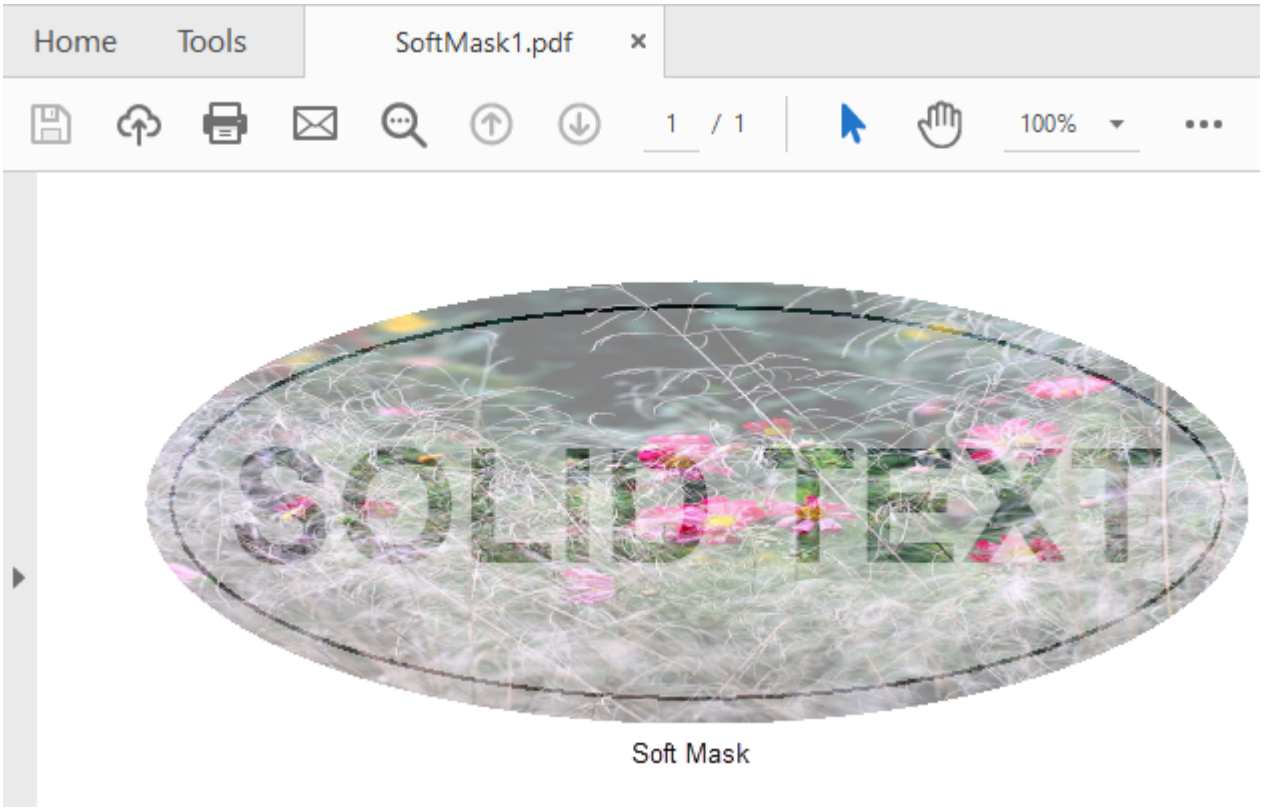
Soft Mask

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

> **Note**:

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



Soft Mask

To create soft mask using DsPdf:

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;
}
```

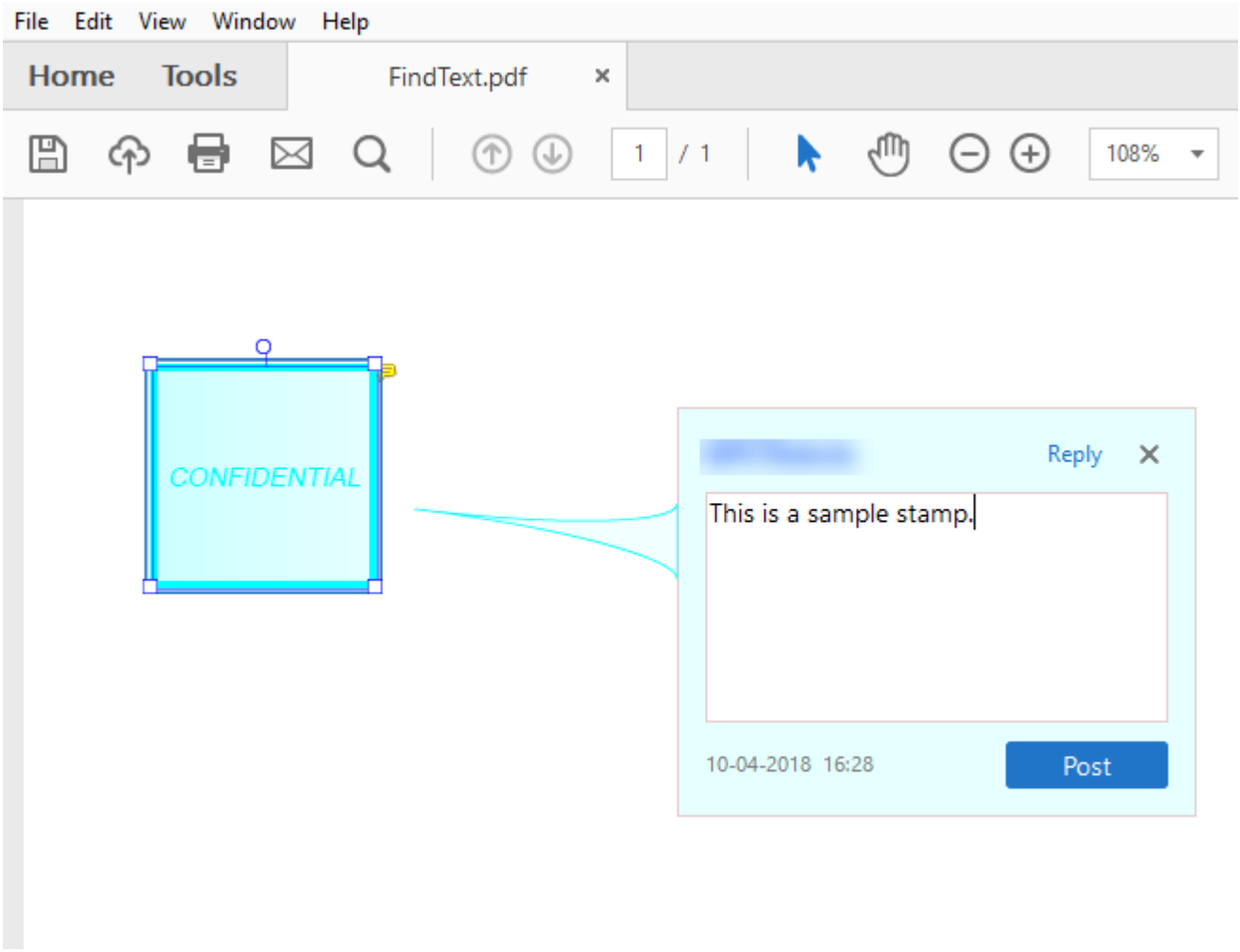
Back to Top

For more information about implementation of soft mask feature using DsPdf, see DsPdf 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).

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



Add Stamp

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

To add a stamp:

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

C#

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

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

```
        Rect = new RectangleF(100.5F, 110.5F, 72, 72),
    };

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

Back to Top

Modify Stamp

To modify stamp annotation, you can set the properties of stamp annotation you used on a page. For instance, setting Contents 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 DsPdf, see DsPdf 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. DsPdf 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 DsPdf, see DsPdf sample browser.

Parse PDF Documents

DsPdf 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 DsPdf 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("DsPdf.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

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

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

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

C#

```
// Open an arbitrary PDF, load it into a temp document and use the map to find some
texts:
using (var fs = new FileStream("Test.pdf", FileMode.Open, FileAccess.Read))
{
    var doc1 = new GcPdfDocument();
    doc1.Load(fs);
    var tmap = doc1.Pages[0].GetTextMap();

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

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

Extract Text Paragraphs

DsPdf 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 DsPdf 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 DsPdf may not correspond correctly to the logical paragraphs as would be recognized by a human.

Extract Data from Tables

DsPdf 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

DsPdf 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, DsPdf 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.Write(s);
            Console.Write("\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 DsPdf.

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.

DsPdf 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

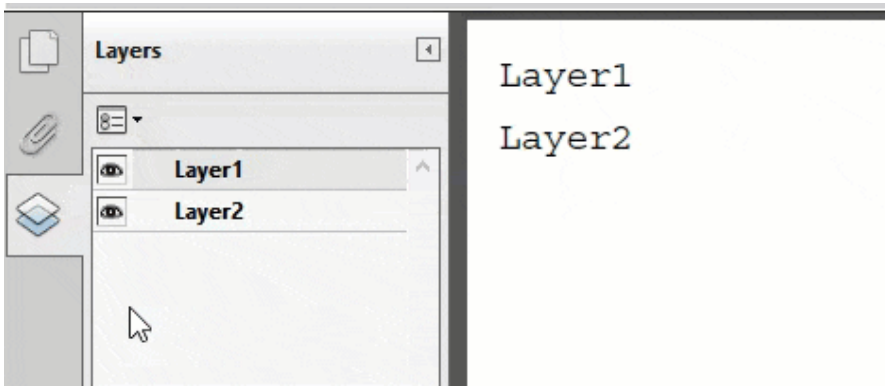
To create layers in a PDF and draw on them:

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

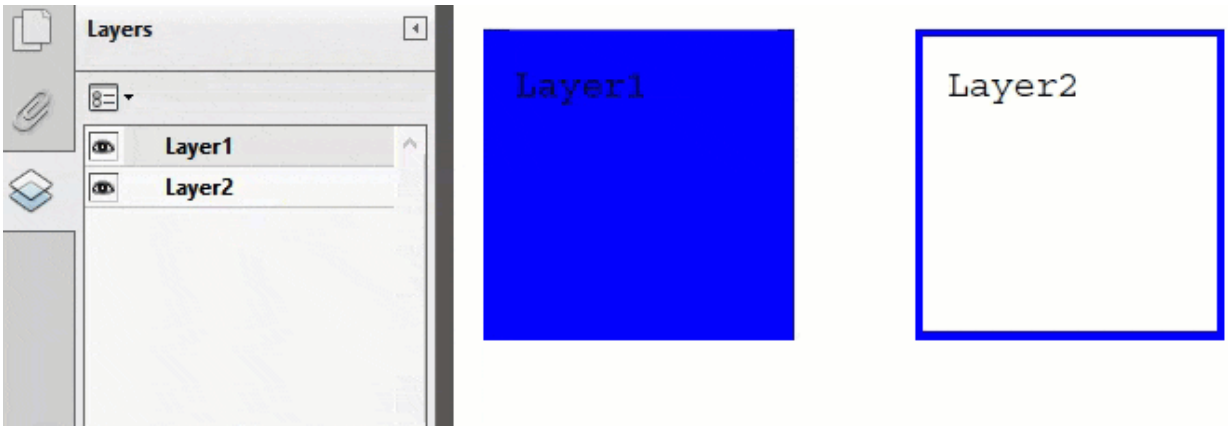
A FormXObject can also be associated with a specific layer. The following code example demonstrates this:

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:



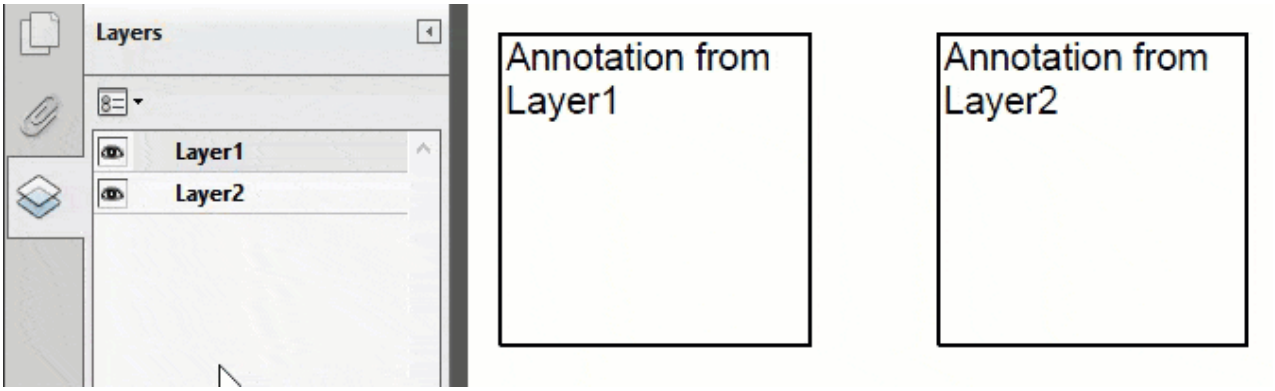
Create Layers and Associate Annotation

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:



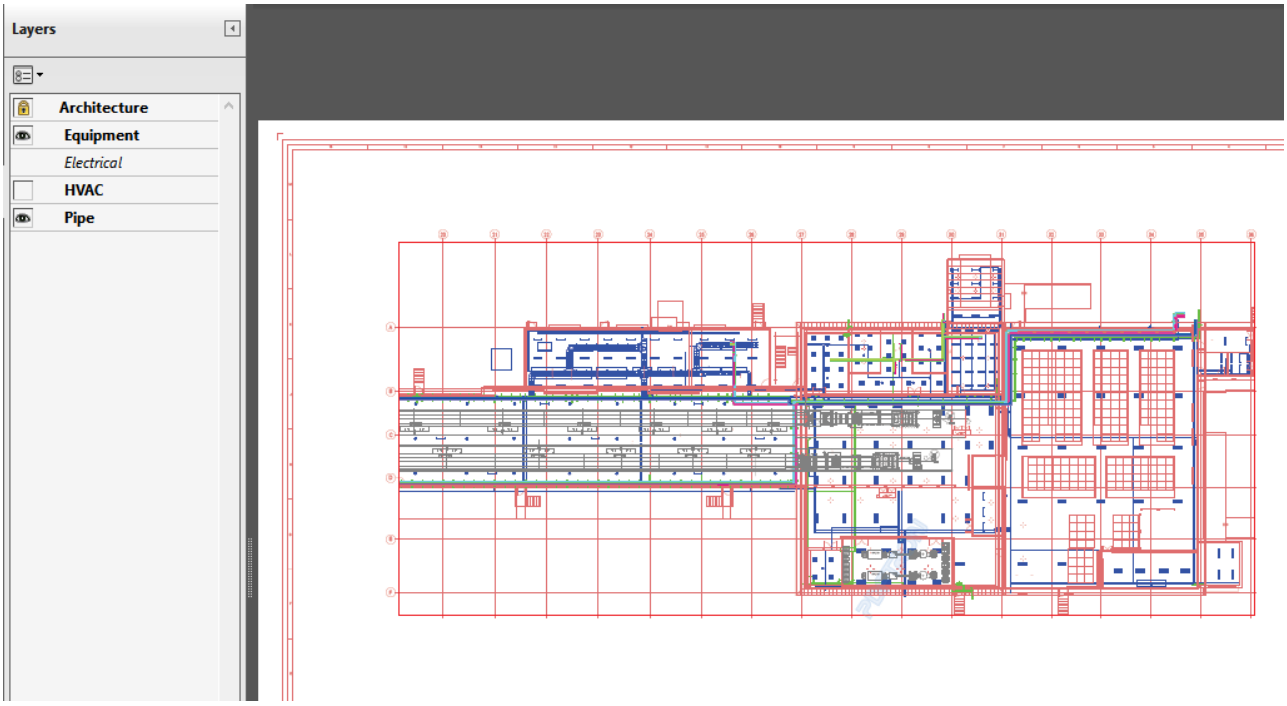
Set Layer Properties

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.SetLayerInitialViewState("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: DsPdfViewer 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 DsPdf, 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[find.PageIndex].Graphics;
            g.BeginLayer(layer);
            doc.Pages[find.PageIndex].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. DsPdf 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

DsPdf 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

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

Text alignment

DsPdf 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

DsPdf 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

DsPdf 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, DsPdf allows you to mix different text formats in the same paragraph using **TextLayout** and DsPdfGraphics.DrawTextLayout method to draw the text layout at a specified location.

Text rotation

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

Vertical text

DsPdf 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

DsPdf 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

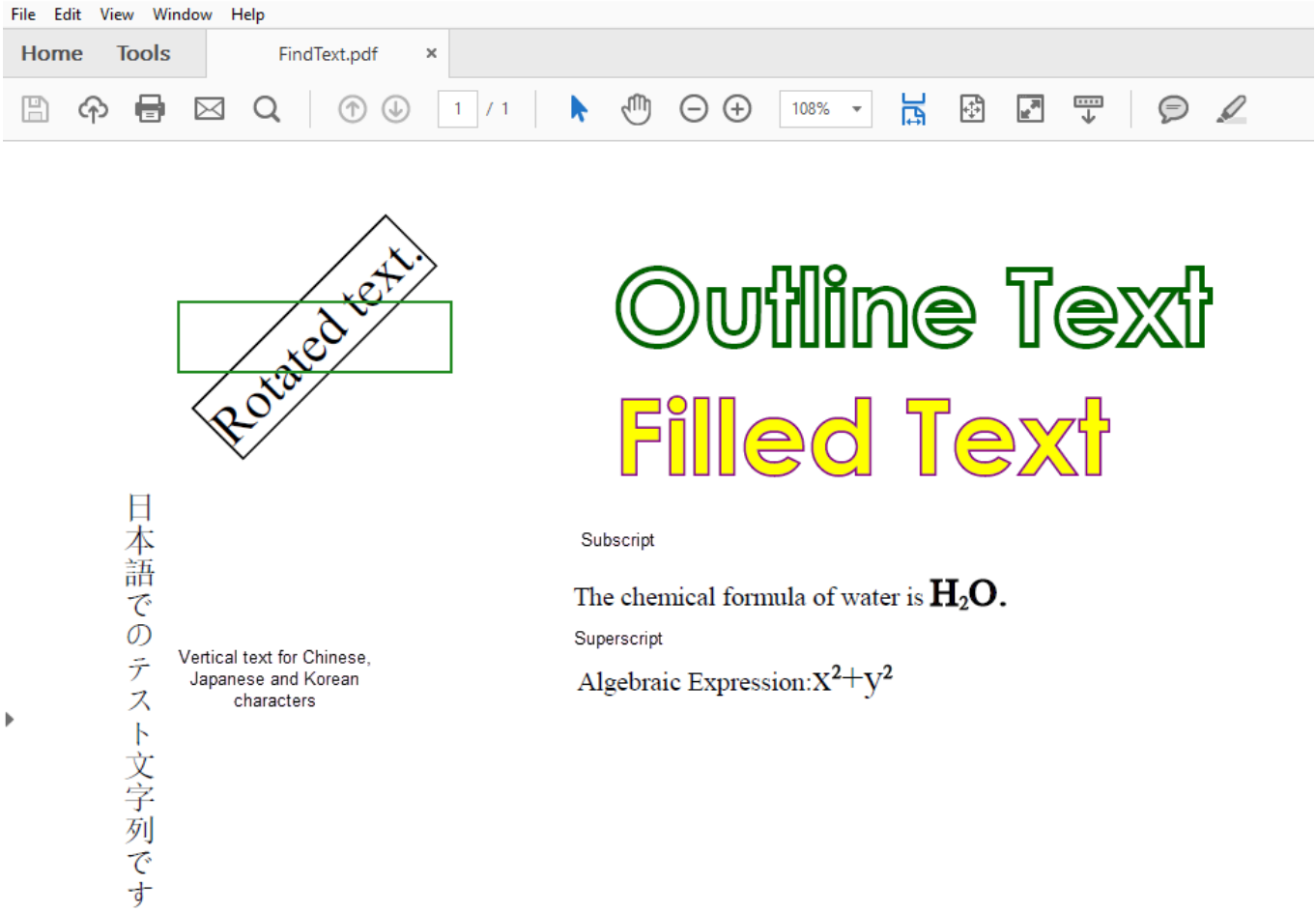
DsPdf offers TrimmingGranularity and WrapMode properties of the TextLayout class that allows trimming the text at word or character, and wrap text in a PDF document. This allows you to display ellipsis or any other character at the end of a text line that does not fit in the allocated space.

Subscript and superscript

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

Paragraph alignment and formatting

DsPdf 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 DsPdf, 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 DsPdf 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 DsPdf, see DsPdf 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

DsPdf 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 DsPdf provides you an option to rotate the sideways text in counter clockwise direction using the RotateSidewaysCounterclockwise property.
Further, SidewaysInVerticalText and UprightInVerticalText property of the TextFormat class also provides options to

display the text sideways or upright respectively. These properties are especially useful for rendering Latin text within the East-Asian language text.

C#

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

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

Back to Top

Outline Text and Fill Text

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

C#

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

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

Back to Top

Text Trimming and Wrapping

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

C#

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

tl.EllipsisCharCode = 0x007E;

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

Back to Top

Subscript and Superscript

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

C#

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


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

Back to Top

Handle Paragraph

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

C#

```
public void CreatePDF(Stream stream)
{
    var doc = new GcPdfDocument();
    var page = doc.NewPage();
    var g = page.Graphics;
    // By default, DsPdf 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 DsPdf, see DsPdf sample browser.

Rotated Text

DsPdf allows you to draw rotated text in unrotated rectangular bounds using DrawRotatedText and MeasureRotatedText methods of GcGraphics class. **DrawRotatedText** draws text at an angle in a specified rectangle, whereas **MeasureRotatedText** calculates the bounds where to draw the text.

Both methods accept the same parameters despite their different functioning. The following table lists the parameters these methods accept:

Parameters	Description
textLayout	TextLayout to draw. This includes one or multiple text lines with various text formats. TextAlignment property specifies the alignment of text in each text line. It is important to draw a rotated text. **Note:** A few other properties of TextLayout have no effect when drawing a rotated text: MaxWidth, MaxHeight, FlowDirection, CanSkipFirstLineWithIndentation, ObjectRects, ParagraphAlignment, MarginLeft, MargingRight, MarginTop, MarginBottom, ColumnWidth, and RowHeight.
angle	Text rotation angle in degrees. The expected range is -90 and +90, specifying an angle in degrees. Positive angles refer to clockwise rotation. Angles less than -90 are treated as -90 degrees, and angles greater than +90 are treated as +90 degrees.
verticalStacking	Stacks text lines either horizontally (along the top and bottom sides of the rectangle) or vertically (along the left and right sides of the rectangle).
rect	Target rectangle for the text.
alignment	Alignment of the whole text rectangle within the target rectangle using RotatedTextAlignment enumeration to: Top Left, Top Right, Top Center, Bottom Left, Bottom Right, Bottom Center, Middle Left, Middle Right, and Middle Center.

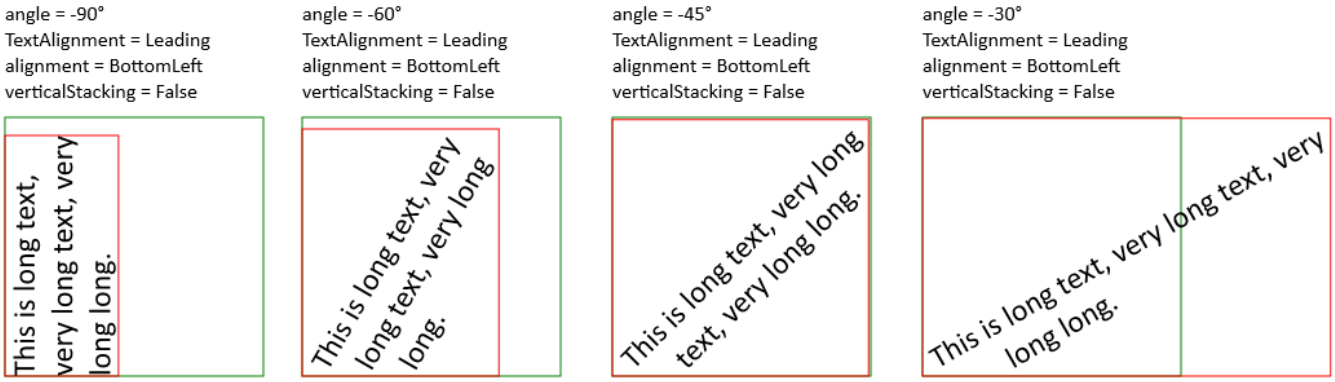
Note: The methods may change or split the original TextLayout into multiple parts. Hence, if necessary, create a clone of the TextLayout in advance.

Refer to the example code to draw multiple rotated texts in different settings:

C#

```
// Initialize GcPdfDocument.
var doc = new GcPdfDocument();

// Add page with graphic.
var g = doc.Pages.Add(new SizeF(1050, 400)).Graphics;

// Draw rotated text with specified angle and alignment.
Draw(g, 10, angle: -90, false, RotatedTextAlignment.BottomLeft,
TextAlignment.Leading);
Draw(g, 240, angle: -60, false, RotatedTextAlignment.BottomLeft,
TextAlignment.Leading);
Draw(g, 480, angle: -45, false, RotatedTextAlignment.BottomLeft,
TextAlignment.Leading);
Draw(g, 720, angle: -30, false, RotatedTextAlignment.BottomLeft,
TextAlignment.Leading);

// Save PDF.
doc.Save("RotatedText.pdf");

// Define Draw method.
static void Draw(GcGraphics g, int x, int angle, bool verticalStacking,
    RotatedTextAlignment rotatedAlign, TextAlignment textAlign)
{
```

```
    // Initialize RectangleF.
    var rect = new RectangleF(x, 100, 200, 200);

    // Draw rectangle.
    g.DrawRectangle(rect, new Pen(Color.Green, 1));

    // Initialize TextLayout.
    var tl = g.CreateTextLayout();

    // Set text format.
    var fmt = new TextFormat
    {
        FontName = "Calibri",
        FontSize = 18,
    };

    // Add the text.
    tl.Append("This is long text, very long text, very long long.", fmt);

    // Set text alignment.
    tl.TextAlignment = textAlign;

    // Clone TextLayout.
    var tlCopy = tl.Clone(true);

    // Calculate bounds of rotated text inside a rectangle.
    var tlRect = g.MeasureRotatedText(tlCopy, angle, verticalStacking, rect,
rotatedAlign);

    // Draw rectangle with text.
    g.DrawRectangle(tlRect, new Pen(Color.Red, 1));

    // Draw rotated text.
    g.DrawRotatedText(tl, angle, verticalStacking, rect, rotatedAlign);

    // Draw strings with all text fromat details.
    fmt.FontSize = 12;
    g.DrawString($"angle = {angle}°", fmt, new PointF(x, 10));
    g.DrawString($"TextAlignment = {tl.TextAlignment}", fmt, new PointF(x, 30));
    g.DrawString($"alignment = {rotatedAlign}", fmt, new PointF(x, 50));
    g.DrawString($"verticalStacking = {verticalStacking}", fmt, new PointF(x, 70));
}
```



Draw Excel Like Rotated Text

With DrawRotatedText method, DsPdf allows you to define DrawExcelText method, which simulates the Excel renderer used for drawing rotated text. This method will differ from the DrawRotatedText method, as it considers the positive angles to render the text in a counterclockwise direction.

Refer to the example code to draw rotated text in specified unrotated rectangular bounds similar to Excel:

C#

```
namespace ExcelTextSimulator
{
    // Define enums.
    enum HorizontalAlignment
    {
        Left,
        Right,
        Center
    }

    enum VerticalAlignment
    {
        Top,
        Bottom,
        Center
    }

    internal class Program
    {
        static void Main(string[] _)
        {
            // Initialize GcPdfDocument.
            var doc = new GcPdfDocument();

            // Add page with graphic.
            var gg = doc.NewPage().Graphics;

            // Draw rotated text.
            Draw(gg, new RectangleF(50, 150, 500, 250));
```

```
            // Save PDF.
            doc.Save("ExcelLikeRotatedText.pdf");
        }

        // Define Draw method.
        static void Draw(GcGraphics g, RectangleF rect)
        {
            // Draw rectangle.
            g.DrawRectangle(rect, new Pen(Color.Green, 1));

            // Initialize TextLayout.
            var tl = g.CreateTextLayout();

            // Set text format.
            var fmt = new TextFormat
            {
                FontName = "Calibri",
                FontSize = 30,
                FontSizeInGraphicUnits = true
            };

            // Add the text.
            tl.Append("Quick brown", fmt);
            fmt.FontSize = 60;
            tl.Append(" fox", fmt);
            fmt.FontSize = 30;
            tl.Append(" jumps over the lazy dog.", fmt);
            fmt.FontSize = 50;
            tl.Append(" Quick brown fox jumps", fmt);
            fmt.FontSize = 20;
            tl.Append(" over the lazy dog.", fmt);

            int angle = 45;

            // Draw text.
            DrawExcelText(g, tl, angle, rect, HorizontalAlignment.Right,
VerticalAlignment.Top);
        }

        // Define DrawExcelText method.
        static void DrawExcelText(GcGraphics g, TextLayout tl, int degrees,
RectangleF rect, HorizontalAlignment hAlign, VerticalAlignment vAlign)
        {
            if (degrees == 90)
            {
                if (vAlign == VerticalAlignment.Bottom)
                    tl.TextAlignment = TextAlignment.Leading;
                else if (vAlign == VerticalAlignment.Top)
                    tl.TextAlignment = TextAlignment.Trailing;
                else
                    tl.TextAlignment = TextAlignment.Center;
```

```
}
else if (degrees == -90)
{
    if (vAlign == VerticalAlignment.Top)
        tl.TextAlignment = TextAlignment.Leading;
    else if (vAlign == VerticalAlignment.Bottom)
        tl.TextAlignment = TextAlignment.Trailing;
    else
        tl.TextAlignment = TextAlignment.Center;
}
else
{
    if (hAlign == HorizontalAlignment.Left)
        tl.TextAlignment = TextAlignment.Leading;
    else if (hAlign == HorizontalAlignment.Right)
        tl.TextAlignment = TextAlignment.Trailing;
    else
        tl.TextAlignment = TextAlignment.Center;
}

RotatedTextAlignment align;
if (vAlign == VerticalAlignment.Top)
{
    if (hAlign == HorizontalAlignment.Left)
        align = RotatedTextAlignment.TopLeft;
    else if (hAlign == HorizontalAlignment.Right)
        align = RotatedTextAlignment.TopRight;
    else
        align = RotatedTextAlignment.TopCenter;
}
else if (vAlign == VerticalAlignment.Bottom)
{
    if (hAlign == HorizontalAlignment.Left)
        align = RotatedTextAlignment.BottomLeft;
    else if (hAlign == HorizontalAlignment.Right)
        align = RotatedTextAlignment.BottomRight;
    else
        align = RotatedTextAlignment.BottomCenter;
}
else
{
    if (hAlign == HorizontalAlignment.Left)
        align = RotatedTextAlignment.MiddleLeft;
    else if (hAlign == HorizontalAlignment.Right)
        align = RotatedTextAlignment.MiddleRight;
    else
        align = RotatedTextAlignment.MiddleCenter;
}

// Draw rotated text inside a specific rectangle.
g.DrawRotatedText(tl, -degrees, false, rect, align);
```

```
        }
    }
}
```

Draw Text in Slanted Rectangles

With DrawSlantedText method of GcGraphics class, DsPdf also allows you to draw rotated text in specified slanted rectangular bounds, similar to Excel. This method is similar to DrawRotatedText method except for the parameter of SlantedTextAlignment type.

SlantedTextAlignment enumeration provides the following six different modes for the slanted rectangles:

Modes	Description
BelowRotatedInside	The text appears below the rectangle side, rotated to the same angle as text. The side above the text is rotated inside the rectangle.
BelowRotatedOutside	The text appears below the rectangle side, rotated to the same angle as text. The side above the text is rotated outside the rectangle.
AboveRotatedInside	The text appears above the rectangle side, rotated to the same angle as text. The side below the text is rotated inside the rectangle.
AboveRotatedOutside	The text appears above the rectangle side, rotated to the same angle as text. The side below the text is rotated outside the rectangle.
CenterInsideOutside	The text appears at the center between the rectangle sides, rotated to the same angle as text. The side above the text is rotated inside the rectangle.
CenterOutsideInside	The text appears at the center between the rectangle sides, rotated to the same angle as text. The side above the text is rotated outside the rectangle.

Refer to the example code to draw rotated text in specified slanted rectangular bounds in different modes similar to Excel:

C#

```
// Initialize GcPdfDocument.
var doc = new GcPdfDocument();
```

```
// Add page with graphic.
var g = doc.Pages.Add(new SizeF(910, 640)).Graphics;

// Draw rotated text in slanted rectangles.
int angle = -70;
var slantedAlign1 = SlantedTextAlignment.BelowRotatedInside;
var slantedAlign2 = SlantedTextAlignment.BelowRotatedOutside;
var slantedAlign3 = SlantedTextAlignment.AboveRotatedInside;
var slantedAlign4 = SlantedTextAlignment.AboveRotatedOutside;
var slantedAlign5 = SlantedTextAlignment.CenterInsideOutside;
var slantedAlign6 = SlantedTextAlignment.CenterOutsideInside;
bool verticalStacking = false;

int x1 = 100;
int y1Head = 10;
int y1 = 100;
int y2Head = 320;
int y2 = 410;

// Draw text and rectangle with specified angle and alignment.
Draw(g, x1, y1Head, y1, angle, verticalStacking, slantedAlign1,
TextAlignment.Leading);
Draw(g, x1 + 270, y1Head, y1, angle, verticalStacking, slantedAlign2,
TextAlignment.Trailing);
Draw(g, x1 + 540, y1Head, y1, angle, verticalStacking, slantedAlign3,
TextAlignment.Center);

Draw(g, x1, y2Head, y2, angle, verticalStacking, slantedAlign4,
TextAlignment.Leading);
Draw(g, x1 + 270, y2Head, y2, angle, verticalStacking, slantedAlign5,
TextAlignment.Trailing);
Draw(g, x1 + 540, y2Head, y2, angle, verticalStacking, slantedAlign6,
TextAlignment.Center);

// Save the PDF.
doc.Save("TextinSlantedRectangle.pdf");

// Define Draw method.
static void Draw(GcGraphics g, int x, int yHead, int y, int angle, bool
verticalStacking, SlantedTextAlignment slantedAlign, TextAlignment textAlign)
{
    RectangleF rect;
    if (!verticalStacking)
    {
        // Initialize RectangleF.
        rect = new RectangleF(x, y, 160, 200);
        float dx = (float)(200.0 / Math.Tan(Math.PI * angle / -180.0));

        // Set switch cases for different slanted text alignments.
        switch (slantedAlign)
```

```
    {
        case SlantedTextAlignment.BelowRotatedInside:
        case SlantedTextAlignment.AboveRotatedOutside:
        case SlantedTextAlignment.CenterInsideOutside:

            // Draw the polygon.
            g.DrawPolygon([
                new PointF(x + dx, y),
                new PointF(x + dx + 160, y),
                new PointF(x + 160, y + 200),
                new PointF(x, y + 200)],
                new Pen(Color.Red, 1));
            break;
        case SlantedTextAlignment.BelowRotatedOutside:
        case SlantedTextAlignment.AboveRotatedInside:
        case SlantedTextAlignment.CenterOutsideInside:

            // Draw the polygon.
            g.DrawPolygon([
                new PointF(x, y),
                new PointF(x + 160, y),
                new PointF(x - dx + 160, y + 200),
                new PointF(x - dx, y + 200)],
                new Pen(Color.Red, 1));
            break;
    }
}
else
{
    // Initialize RectangleF.
    rect = new RectangleF(x, y, 200, 160);
    float dy = (float)(200.0 * Math.Tan(Math.PI * angle / 180.0));

    // Set switch cases for different slanted text alignments.
    switch (slantedAlign)
    {
        case SlantedTextAlignment.BelowRotatedInside:
        case SlantedTextAlignment.AboveRotatedOutside:
        case SlantedTextAlignment.CenterInsideOutside:
            if (angle >= 0)

                // Draw the polygon.
                g.DrawPolygon([
                    new PointF(x, y),
                    new PointF(x + 200, y + dy),
                    new PointF(x + 200, y + dy + 160),
                    new PointF(x, y + 160)],
                    new Pen(Color.Red, 1));
            else

                // Draw the polygon.
```

```
                g.DrawPolygon([
                    new PointF(x, y - dy),
                    new PointF(x + 200, y),
                    new PointF(x + 200, y + 160),
                    new PointF(x, y - dy + 160)],
                    new Pen(Color.Red, 1));
            break;
        case SlantedTextAlignment.BelowRotatedOutside:
        case SlantedTextAlignment.AboveRotatedInside:
        case SlantedTextAlignment.CenterOutsideInside:
            if (angle >= 0)

                // Draw the polygon.
                g.DrawPolygon([
                    new PointF(x, y - dy),
                    new PointF(x + 200, y),
                    new PointF(x + 200, y + 160),
                    new PointF(x, y - dy + 160)],
                    new Pen(Color.Red, 1));
            else

                // Draw the polygon.
                g.DrawPolygon([
                    new PointF(x, y),
                    new PointF(x + 200, y + dy),
                    new PointF(x + 200, y + dy + 160),
                    new PointF(x, y + 160)],
                    new Pen(Color.Red, 1));
            break;
    }
}

// Draw the rectangle.
g.DrawRectangle(rect, new Pen(Color.Blue, 0.5f) { DashStyle = DashStyle.Dash });

// Initialize TextLayout.
var tl = g.CreateTextLayout();

// Set text format.
var fmt = new TextFormat
{
    FontName = "Calibri",
    FontSize = 18,
};

// Add the text.
tl.Append("Working of different slanted text alignments.", fmt);

// Set text alignment.
tl.TextAlignment = textAlign;
```

```
    // Draw text in slanted rectangle.
    g.DrawSlantedText(tl, angle, verticalStacking, rect, slantedAlign);

    // Draw strings with all details.
    fmt.FontSize = 12;
    g.DrawString($"angle = {angle}°", fmt, new PointF(x, yHead));
    g.DrawString($"TextAlignment = {tl.TextAlignment}", fmt, new PointF(x, yHead +
20));
    g.DrawString($"alignment = {slantedAlign}", fmt, new PointF(x, yHead + 40));
    g.DrawString($"verticalStacking = {verticalStacking}", fmt, new PointF(x, yHead +
60));
}
```

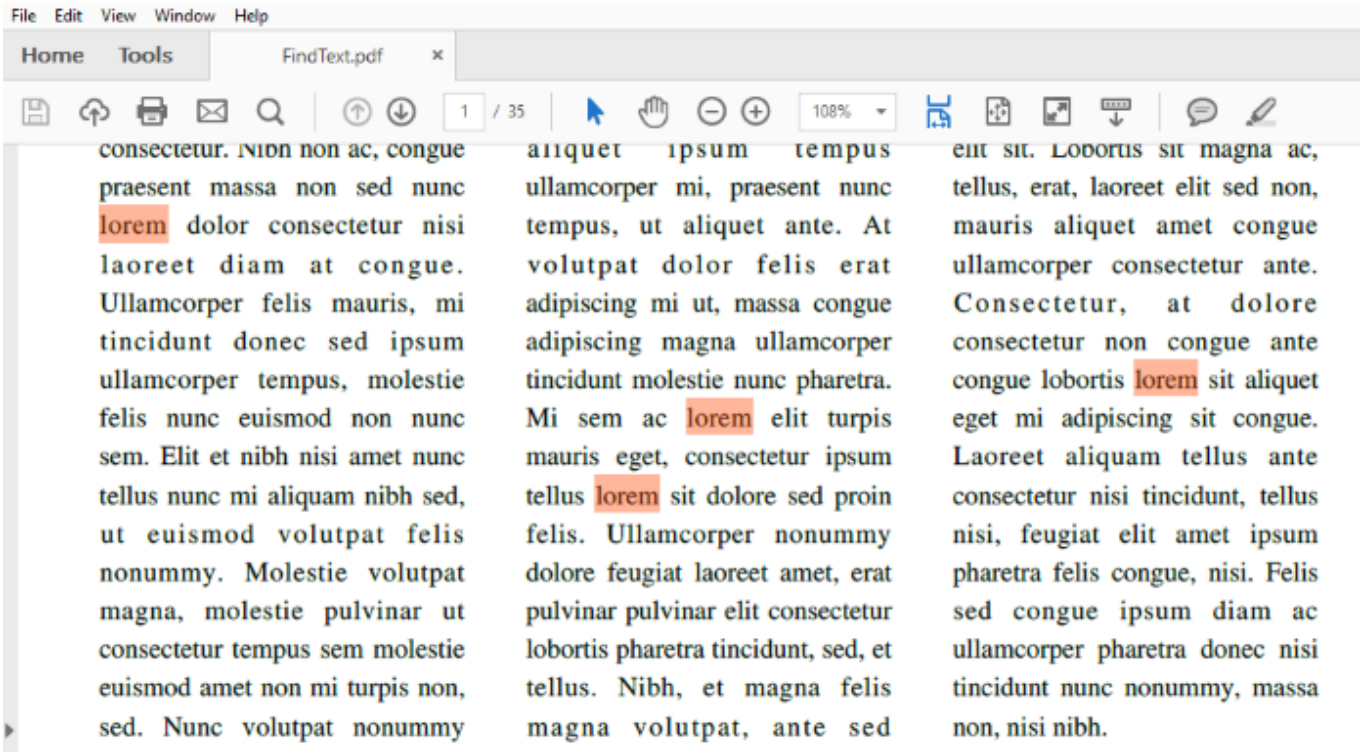
Text Search, Replace and Delete

Search Text

DsPdf allows text search in a PDF document to find all occurences of the specified text. The library supports all common find text options including regular expressions, case-sensitive search, etc. It also works across line breaks, so logically connected text that is rendered on different text lines can also be found. You can use FindText method of the GcPdfDocument to search text in a PDF document. This method accepts object of FindTextParams and **OutputRange** class as parameters to find all the occurrences of the searched string in the loaded document. The **FindTextParams** class represents the target text to be searched. The class also lets you incorporate other useful search options discussed below:

Search and Highlight Text

FindText method returns a list of all occurrences of the searched text. You can iterate through the list and highlight the search results using **FillPolygon** and **DrawPolygon** methods of the **GcGraphics** class.



The example below shows how to search and highlight a text string in a PDF document:

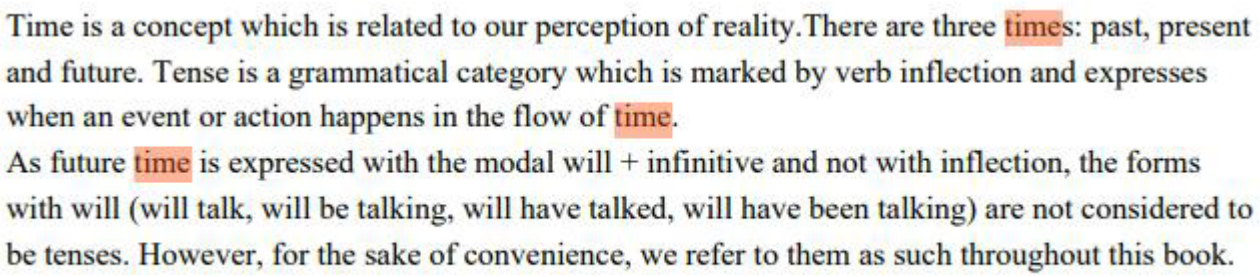
C#

```
public void CreatePDF(Stream stream)
{
    //load file
    var doc = new GcPdfDocument();
    using var fs = File.OpenRead("TimeSheet.pdf");
    doc.Load(fs);

    //define text bounds
    var findText = new FindTextParams("HOURS", true, false);

    //find text
    IList<FoundPosition> findTextList = doc.FindText(findText);

    //highlight text
    foreach (FoundPosition text in findTextList)
    {
        //get bounds of each occurrence of found text
        var g = doc.Pages[text.PageIndex].Graphics;
        Quadrilateral[] pos = text.Bounds;

        //highlight the text
```

```
        g.DrawPolygon(pos[0], Color.Yellow, 1);
        g.FillPolygon(pos[0], Color.FromArgb(100, Color.OrangeRed));
    }

    //save pdf
    var newDoc = new GcPdfDocument();
    newDoc.Load(fs);
    doc.Save("FindText.pdf");

}
```

Case-sensitive Search

Case sensitivity is also one of the criteria while searching for a text string. Using DsPdf library, you can specify whether the text search should be case sensitive or not. To search for a text with matching case, you can set **matchCase** parameter of the **FindTextParams** method to true.


Time is a concept which is related to our perception of reality.There are three times: past, present and future. Tense is a grammatical category which is marked by verb inflection and expresses when an event or action happens in the flow of time.
As future time is expressed with the modal will + infinitive and not with inflection, the forms with will (will talk, will be talking, will have talked, will have been talking) are not considered to be tenses. However, for the sake of convenience, we refer to them as such throughout this book.

The example below shows how to search strings having specific case in a PDF document:

C#

```
//find word "time", the word "Time" or "TIME" will be ignored
var findWord = new FindTextParams("time", false, true);
var findText = doc.FindText(findWord);
```

Whole Word Search

DsPdf lets you search for a whole word or you can also search for instances that are subset of a certain word present in the PDF document. To search for a whole word, you can set **wholeWord** parameter of the **FindTextParams** method to **true**.

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

The example below shows how to search whole word strings in a PDF document:

C#

```
//find word "time", the word "overtime" will be ignored
var findWord = new FindTextParams("Time", true, false);
var findText = doc.FindText(findWord);
```

Regular Expression Search

Regular expressions are useful when you want to search variable text strings that use common pattern such as date, time, email address, etc. instead of searching a particular text or phrase. To search using regular expressions, you need to pass regular expression as a string parameter to the **FindTextParams** method and set its **regex** parameter to **true**.

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

C#

```
//finds all the dates present in PDF document, using regular expressions
var findWord = new FindTextParams(@"\d+[/-]\w+[/-]\d\d", false, false, 72, 72, false,
true);
var findText = doc.FindText(findWord);
```

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

Replace Text

With DsPdf, you can replace a text in the whole document or its specific page by using ReplaceText method which is available in the **GcPdfDocument** and Page classes, and on the ITextMap interface. This method accepts the object of FindTextParams class and the new text string along with other parameters to find and replace all occurrences of the target text. It searches the target text and replaces it with the new text along with adjusting the space required to accommodate the replaced text.

The code below shows how to replace a text in the whole document:

C#

```
// replace word ".NET Standard 2.0" with ".NET 6" in document
using (FileStream fs = new FileStream(@"..\..\..\DotnetFramework.pdf", FileMode.Open,
FileAccess.Read, FileShare.Read))
{
    GcPdfDocument doc = new GcPdfDocument();
    doc.Load(fs);
    FindTextParams ftp = new FindTextParams(".NET Standard 2.0", true, false);
```

```
    doc.ReplaceText(ftp, ".NET 6", null, null, null);
    doc.Save("DotnetFramework_Document.pdf");
}
```

Delete Text

DsPdf allows you to delete a text in the whole document or a specific page by using DeleteText method which is available in the **GcPdfDocument** and **Page** classes and, on the **ITextMap** interface. This method accepts the object of **FindTextParams** class and DeleteTextMode enumeration. The DeleteTextMode enumeration provides two options - 'Standard' and 'PreserveSpace' which represent two modes of deleting text in a Pdf document.

On deleting a text in the **Standard** mode, the text following the deleted text shifts to fill the void created by deleted text. However, in the **PreserveSpace** mode, the document retains an empty space at the place of the deleted text and text after the deleted text does not move.

The code below shows how to delete a text string from first page of a PDF document using standard mode:

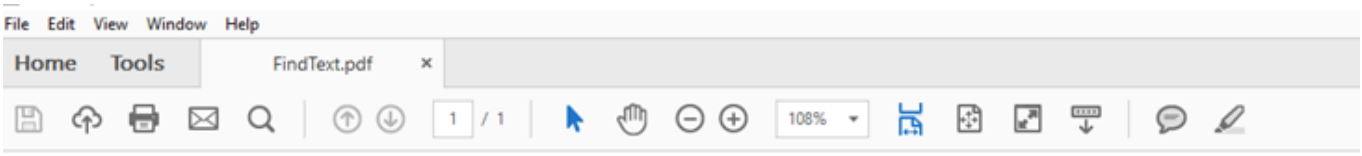
C#

```
// delete word "wetlands" from the first page using DeleteTextMode.Standard
using (FileStream fs = new FileStream(@"..\..\..\Wetlands.pdf", FileMode.Open,
FileAccess.Read, FileShare.Read))
{
    GcPdfDocument doc = new GcPdfDocument();
    doc.Load(fs);
    FindTextParams ftp = new FindTextParams("wetlands", true, false);
    doc.Pages[0].DeleteText(ftp, DeleteTextMode.Standard);
    doc.Save("wetlands_deleted.pdf");
}
```

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).

DsPdf 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 DsPdf 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 **DS.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
```

Access Primitive and High-Level PDF Objects

A PDF document consists of some primitive and high-level PDF objects. Generally, a PDF document contains nine primitive types of objects and can be interpreted as a graph of linked primitive PDF objects, where an object is one of the following types defined in the PDF specification:

- PDF array
- PDF bool
- PDF dictionary
- PDF name
- PDF null
- PDF number
- PDF reference
- PDF stream
- PDF string

All high-level PDF objects in object model (such as Page, AnnotationBase, Action, etc.) are implemented as wrappers around primitive PDF objects. A wrapper contains a reference to the underlying primitive PDF type (PdfDict, PdfArray, PdfDictObject, etc.) and provides methods and properties for accessing and manipulating the underlying object. The root class for all high-level objects is PdfWrapperBase; it contains a reference to the underlying PDF primitive object defined by **IPdfObject**.

DsPdf allows you to work directly with the primitive objects used to build all the high-level entities in a PDF document, such as DocumentInfo, a PDF dictionary, using the following listed interfaces and classes, and their methods and properties in GrapeCity.Documents.Pdf.Spec namespace:

> **Note:** All the types and their members mentioned below are for advanced users only. The reader of this document must have a basic idea of PDF specification, direct and indirect PDF objects, and how a PDF file is organized.

Interface/Class	Description
IPdfObject	It is the common interface supported by all PDF objects in a GcPdfDocument that are persisted in a PDF file. Indirect and ObjID properties allow you to identify indirect PDF objects and IDs of the PDF objects.
IPdfArray	It is the common interface implemented by PdfArray, PdfArrayObject, and PdfArrayWrapper types.
IPdfArrayExt	It contains extension methods for the IPdfArray interface.
IPdfDict	It is the common interface implemented by PdfDict, PdfDictObject, and PdfDictWrapper types.
IPdfDictExt	It contains extension methods for the IPdfDict interface.
IPdfName	It is the common interface for PdfName and PdfNameObject.
IPdfNameExt	It contains extension methods for the IPdfName interface.
IPdfNumber	It is the common interface for PdfNumber and PdfNumberObject.
IPdfNumberExt	It contains extension methods for the IPdfNumber interface.
IPdfRef	It is the common interface for PdfRef and PdfRefObject.
IPdfRefExt	It contains extension methods for the IPdfRef interface.
IPdfString	It is the common interface for PdfString and PdfStringObject.

IPdfStringExt	It contains extension methods for the IPdfString interface.
IPdfBool	It is the common interface for PdfBool and PdfBoolObject.
IPdfBoolExt	It contains extension methods for the IPdfBool interface.
IPdfNull	It is the common interface for PdfNull and PdfNullObject.
IPdfNullExt	It contains extension methods for the IPdfNull interface.
PdfArray	It represents a direct PDF array object.
PdfArrayObject	It represents an indirect PDF array object.
PdfArrayWrapper	It represents an array wrapper object.
PdfDict	It represents a direct PDF dictionary object.
PdfDictObject	It represents an indirect PDF dictionary object.
PdfDictWrapper	It represents a dictionary wrapper object.
PdfName	It represents a direct PDF name object. This class overrides GetHashCode() and Equals(object) methods and defines the equality and inequality operators. This class is immutable.
PdfNameObject	It represents an indirect PDF name object.
PdfNumber	It represents a direct PDF number object. The class overrides GetHashCode() and Equals(object) methods and defines the equality and inequality operators. This class is immutable.
PdfNumberObject	It represents an indirect PDF number object.
PdfStreamObjectBase	It represents a PDF stream. It is always an indirect object, as a stream cannot be a direct object in PDF.
PdfRef	It represents a direct PDF reference object. This class overrides GetHashCode() and Equals(object) methods. The class is immutable.
PdfRefObject	It represents an indirect PDF reference object.
PdfString	It represents a direct PDF string object. This class overrides GetHashCode() and Equals(object) methods and defines the equality and inequality operators. The class is immutable.
PdfStringObject	It represents an indirect PDF string object.
PdfBool	It represents a direct PDF bool object. You cannot create instances of this class from user code; the two predefined instances are PdfBool.True and PdfBool.False. Overrides GetHashCode() and Equals(object), which define equality and inequality operators.
PdfBoolObject	It represents an indirect PDF bool object.
PdfNull	It represents a direct PDF null object. You cannot create instances of this class from user code; instead, use the PdfNull.Instance predefined instance. It overrides GetHashCode() and Equals(object), which define equality and inequality operators. This class is immutable.
PdfNullObject	It represents an indirect PDF null object.

The PDF specification defines the properties that can be present in this dictionary (Creator, Author, etc.), but PDF producers can add arbitrary custom properties, such as the SourceModified property, which is often found in various real-world PDF files. Types from GrapeCity.Documents.Pdf.Spec namespace allow you to access (read, write, or edit) such custom elements.

Since most high-level objects in a PDF file are PDF dictionaries, in the DsPdf API, the corresponding objects are derived from the **PdfDictWrapper** class, which in turn is derived from PdfWrapperBase and uses **IPdfDict** as the underlying object. The GetPdfStream, GetPdfStreamInfo, and GetPdfStreamData methods of PdfWrapperBase can retrieve data from the PDF stream associated with the PDF dictionary.
Each high-level PDF object (depending on its type) implements one of the primitive interfaces so that the user can use the extension methods of GrapeCity.Documents.Pdf.Spec namespace with these high-level objects.

Refer to the following example code to get image properties from a PDF document:

C#

```
// Initialize GcPdfDocument.
GcPdfDocument doc = new GcPdfDocument();

// Load PDF document.
doc.Load(fs);

// Get image from the PDF document.
var imgs = doc.GetImages();
var pi = imgs[0].Image;

// Write image ID.
Console.WriteLine($"PdfImage object ID: {pi.ObjID}");

/* The PdfImage is a descendant of PdfDictWrapper object and has a lot of methods
   that allow you to get properties and data from the underlying PDF stream object.
*/
using (PdfStreamInfo psi = pi.GetPdfStreamInfo())
{
    // Get image information such as length filter name, filter decode parameters,
etc.
    Console.WriteLine($"    Image stream length: {psi.Stream.Length}");
    Console.WriteLine($"        ImageFilterName: {psi.ImageFilterName}");
    Console.WriteLine($"ImageFilterDecodeParams: {psi.ImageFilterDecodeParams}");

    // Dump content of ImageFilterDecodeParams.
    foreach (var kvp in psi.ImageFilterDecodeParams.Dict)
    {
        Console.WriteLine($"{kvp.Key}: {kvp.Value}");
    }

    // Get value of BlackIs1.
    var blackIs1 = psi.ImageFilterDecodeParams.GetBool(PdfName.Std.BlackIs1, null);
    Console.WriteLine($"BlackIs1: {blackIs1}");
}

// Dump properties of PdfImage dictionary.
Console.WriteLine();
Console.WriteLine("Properties of PdfImage dictionary:");
foreach (KeyValuePair<PdfName, IPdfObject> kvp in pi.PdfDict.Dict)
{
    Console.WriteLine($"{kvp.Key}: {kvp.Value}");
}
```

```
// Get color space and bits per component.
var cs = pi.Get<IPdfObject>(PdfName.Std.ColorSpace);
Console.WriteLine($"ColorSpace: {cs.GetType().Name} {cs}");
var bpc = pi.Get<IPdfObject>(PdfName.Std.BitsPerComponent);
Console.WriteLine($"BitsPerComponent: {bpc?.GetType().Name} {bpc}");
```

Render HTML to PDF

DsPdf library along with **DsHtml** library, lets you easily render HTML content to PDF document. With a utility library like DsHtml, 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.

DsHtml 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. 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 DS.Documents.Html package contains the following namespaces:

- GrapeCity.Documents.Html namespace
  provides GcHtmlRenderer(Obsolete), GcHtmlBrowser, PdfOptions, ImageOptions, PageOptions, HtmlPage classes etc.
- GrapeCity.Documents.Pdf namespace provides the GcPdfGraphicsExt and HtmltoPdfFormat classes.
- GrapeCity.Documents.Drawing namespace provides GcBitmapGraphicsExt and HtmlToImageFormat class.

Install DsHtml Package

Refer the steps below to install DsHtml 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 DS.Documents.Pdf under the **Browse** tab and click **Install**.
4. Similarly, install DS.Documents.Html.

   > **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.

5. Once, the DsHtml 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;
   ```

6. Apply DsPdf license to **GcHtmlBrowser** class of DsHtml 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 html = "<html><body><h1>My First Heading</h1> My first paragraph. </body></html>";
     var re = new GcHtmlBrowser(html);
     re.ApplyGcPdfLicenseKey("key");

   - To license all the instances
     GcHtmlBrowser.SetGcPdfLicenseKey("key");

7. Write the sample code.

Render HTML Webpage to PDF Document

DsHtml can render HTML webpage to PDF document. DsPdf provides the PdfOptions class and RenderToPdf method of GcHtmlBrowser class to render HTML files to PDF documents.

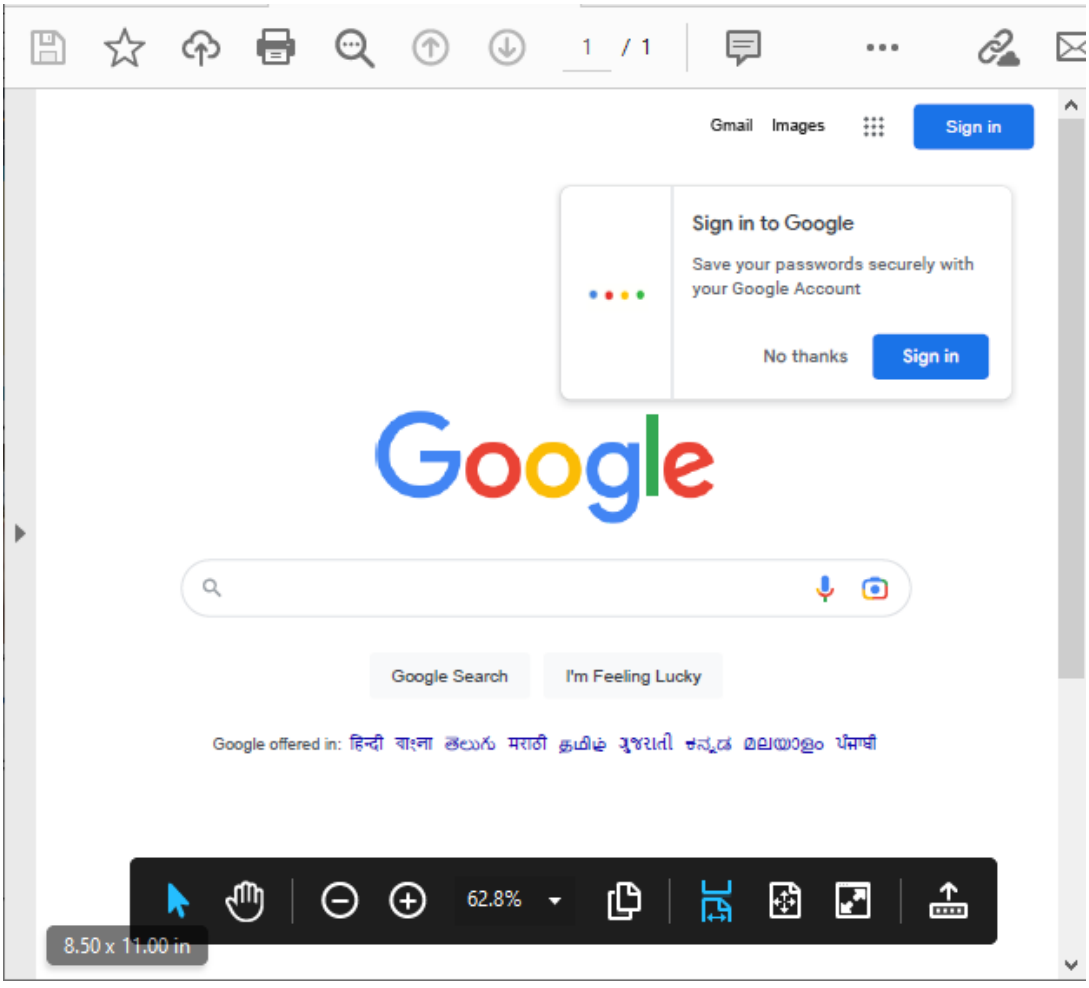
To render the HTML webpage to a PDF document, follow the steps below:

1. Specify the PDF file path for rendering HTML webpage.
2. Specify the HTML source (URI).
3. Define the PDF document settings using the **PdfOptions** class.
4. Convert the HTML webpage to PDF file using SaveAsPdf method of **HtmlPage** class.

C#

```
// Get a temporary file where the web page will be rendered:
var tmp = Path.GetTempFileName();

// The Uri of the web page to render:
var uri = new Uri("http://www.google.com");

// Create an instance of GcHtmlBrowser that is used to render HTML:
var browserPath = BrowserFetcher.GetSystemChromePath();
using var browser = new GcHtmlBrowser(browserPath);

// Create an HtmlPage instance rendering the source Uri:
using var htmlPage = browser.NewPage(uri);

// Render the source Web page to the temporary file:
htmlPage.SaveAsPdf(tmp);

// 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);
```

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 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:

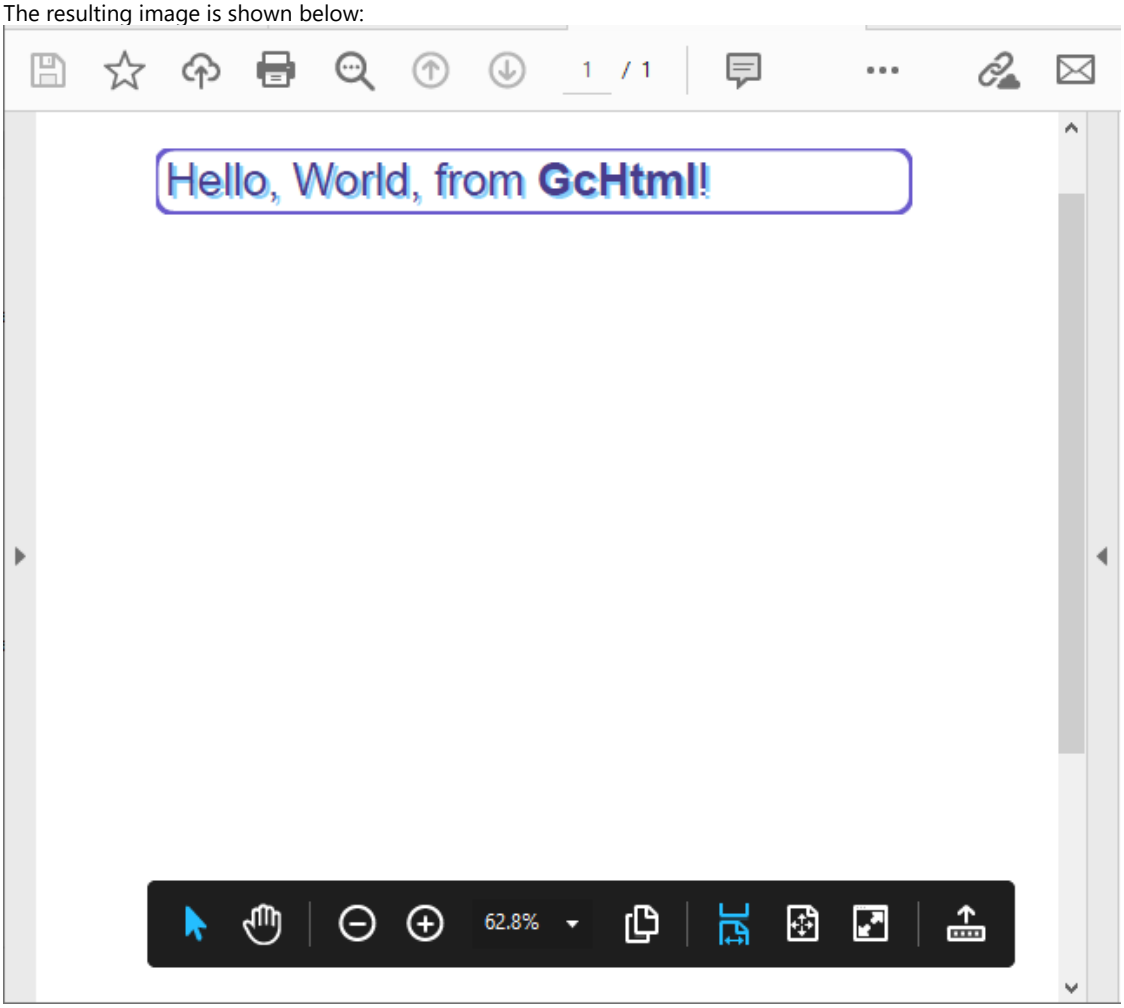
1. Create an instance of the **GcPdfDocument** class.
2. Configure PDF settings using the HtmlToPdfFormat class.
3. Add an HTML string to a PDF document using the DrawHtml method of **GcPdfGraphicsExt** class.
4. Save the PDF file using the Save method of the **GcPdfDocument** class.

C#

```
// HTML code that represents the content to render:
var html = "<!DOCTYPE html>" +
    "<html>" +
    "<head>" +
    "<style>" +
    "span.bold {" +
        "font-weight: bold;" +
    "}" +
    "p.round {" +
        "font: 36px arial, sans-serif;" +
        "color: DarkSlateBlue;" +
        "border: 4px solid SlateBlue;" +
        "border-radius: 16px;" +
        "padding: 3px 5px 3px 5px;" +
        "text-shadow: 3px 2px LightSkyBlue;" +
    "}" +
    "</style>" +
    "</head>" +
    "<body>" +
    "<p class='round'>Hello, World, from <span class='bold'>DsHtml</span>!</p>" +
    "</body>" +
    "</html>";

// Create a new PDF document, add a page, get graphics to draw on:
var doc = new GcPdfDocument();
var page = doc.NewPage();
var g = page.Graphics;

try
{
    // Create an instance of GcHtmlBrowser that is used to render HTML:
    var browserPath = BrowserFetcher.GetSystemChromePath();
    using var browser = new GcHtmlBrowser(browserPath);

    // Render HTML.
    var ok = g.DrawHtml(browser, html, 72, 72,
        new HtmlToPdfFormat(false) { MaxPageWidth = 6.5f, MaxPageHeight = 9f },
        out SizeF size);
}
catch (Exception ex)
{
    throw new Exception($"Error:\n{ex.Message}");
}
// Done:
doc.Save(stream);
```

Back to Top

The resulting image is shown below:



For more information on rendering HTML to PDF using DsPdf, see DsPdf sample browser.

Tips to Migrate from Obsolete GcHtmlRenderer Class

If your application uses obsolete **GcHtmlRenderer** class to convert the HTML pages or content to a pdf format, you can use following steps to quickly update to the new **GcHtmlBrowser** class which does not depend on a custom build of Chromium and does not require GPL or LGPL licenses.

1. In Solution Explorer, go to Project > **Dependencies** > **Packages** and remove any references to
   - GrapeCity.Documents.Html.Windows.X64
   - GrapeCity.Documents.Html.Linux.X64
   - GrapeCity.Documents.Html.Mac.X64
2. Check for licensing calls and if they exist, change them as follows:

   C#

   ```
   GcHtmlRenderer.SetGcPdfLicenseKey(key); -> GcHtmlBrowser.SetGcPdfLicenseKey(key);
   ```

3. In order to create and use an instance of GcHtmlBrowser, you require the path to a Chromium based browser on the current system. For instance, in case of Chrome browser:
   - You can get the path to an existing instance of Chrome installed on the current system.

     C#

     ```
     var path = BrowserFetcher.GetSystemChromePath();
     ```

   - Or, you can download and install Chrome in a location of your choice.

     C#

     ```
     var tp = Path.GetTempPath();
     var bf = new BrowserFetcher() { DestinationFolder = Path.Combine(tp, ".gc-chromium") };
     var path = bf.GetDownloadedPath();
     ```

> **Note:** We recommend using **Chrome** browser with GcHTMLBrowser class as Edge has some differences in the implementation of some DevTools features.

4. Create an instance of GcHtmlBrowser. Note that **RunWithNoSandbox** option may be needed on some Linux systems.

C#
```
if (RuntimeInformation.IsOSPlatform(OSPlatform.Linux))
    return new GcHtmlBrowser(path, new LaunchOptions { RunWithNoSandbox = true });
else
    return new GcHtmlBrowser(path)
```

5. In all calls to GcGraphics.DrawHtml() method, insert browser instance as the first parameter.

C#
```
g.DrawHtml(html, ...); -> g.DrawHtml(browser, html, ...);
```

6. Look for instances of GcHtmlRenderer class which are rendering Uri such as

C#
```
using var re = new GcHtmlRenderer(uri);
...
re.RenderToPdf(file, new PdfSettings() {…});
```

and replace it with

C#
```
// Create an HtmlPage from the URI
// (DefaultBackgroundColor and WindowSize options from Pdf/Jpeg/PngSettings
// have moved to PageOptions, while some other options are now in LaunchOptions):
using var htmlPage = browser.NewPage(uri, new PageOptions() { WindowSize = pixelSize;… });
...
htmlPage.SaveAsPdf(file, new PdfOptions() {...});
```

> **Note:** Few methods and properties of JpegSettings and PngSettings classes have been moved to LaunchOptions and PageOptions classes.

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 DsPdf 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

DsPdf 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

DsPdf 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

DsPdf library also provides InterpolationMode property in the SaveAsImageOptions class, which is utilized to set interpolation mode while drawing bitmap images in PDF files or saving the PDF files as images; by default, it is NearestNeighbor.

Refer to the following example code to set InterpolationMode while saving the PDF file as images:

C#

```
class Program
{
    GrapeCity.Documents.Pdf.GcPdfDocument Document;
    GrapeCity.Documents.Pdf.GcPdfGraphics Graphics;
    GrapeCity.Documents.Pdf.Page Page;
    public const float Resolution = 25.4f;
    public const float MM = 1;
    public const float CM = MM * 10;
    System.Drawing.SizeF PageSize;
    System.Drawing.RectangleF TextRect;
    static void Main(string[] args)
    {
        // Initialize GcPdfDocument.
        var pdf = new GrapeCity.Documents.Pdf.GcPdfDocument();

        // Draw page in PDF document.
```

```
        new Program(pdf).DrawPage();

        // Instantiate SaveAsImageOptions.
        SaveAsImageOptions options = new SaveAsImageOptions();

        // Set InterpolationMode to NearestNeighbor.
        options.InterpolationMode =
GrapeCity.Documents.Drawing.InterpolationMode.NearestNeighbor;

        // Save PDF document.
        pdf.Save("test.pdf");

        // Save PDF document as BMP image with SaveAsImageOptions.
        pdf.SaveAsBmp("test.bmp", null, options);

        // Save PDF document as PNG image with SaveAsImageOptions.
        pdf.SaveAsPng("test.png", null, options);

        // Save PDF document as JPEG image with SaveAsImageOptions.
        pdf.SaveAsJpeg("test.jpg", null, options);

    }
    Program(GrapeCity.Documents.Pdf.GcPdfDocument doc)
    {
        Document = doc;
    }

    // Configure page settings for PDF document.
    void AddPage()
    {
        Page = Document.NewPage();
        Page.PaperKind = GrapeCity.Documents.Common.PaperKind.A4;
        Page.Landscape = false;
        PageSize = Page.GetRenderSize(Resolution, Resolution);
        Graphics = Page.Graphics;
        Graphics.Resolution = Resolution;
        TextRect = new System.Drawing.RectangleF(0, 0, PageSize.Width,
PageSize.Height);
        TextRect.Inflate(-2.5f * CM, -2.7f * CM);
        Graphics.DrawRectangle(TextRect, new
GrapeCity.Documents.Drawing.Pen(System.Drawing.Color.Green, 2f * MM));
    }

    // Draw image on Bitmap.
    void DrawBitmap(System.Drawing.RectangleF rect)
    {
        var pngImage =
GrapeCity.Documents.Drawing.Image.FromFile(@"..\..\..\qrcode.png");
        Graphics.DrawImage(pngImage, rect, rect,
GrapeCity.Documents.Drawing.ImageAlign.StretchImage);
    }
```

```
    // Draw page in PDF document.
    void DrawPage()
    {
        AddPage();
        DrawBitmap(new System.Drawing.RectangleF(5 * CM, 5 * CM, 10 * CM, 10 * CM));
    }
}
```

Note: The interpolation mode only affects the way bitmap images are drawn on a graphic, i.e., the result of DrawImage method and bitmap image resizing. Interpolation mode does not affect any other graphics operations. In particular, if a PDF is saved to an image format, the only items affected by interpolation mode would be bitmap images embedded in the original PDF, if they exist.

Back to Top

Save as SVG

In addition to above mentioned common image formats, DsPdf 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("DsPDFRenderToSVG.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("DsPDFRenderToSVGZ.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

DsPdf 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 DsImaging 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

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

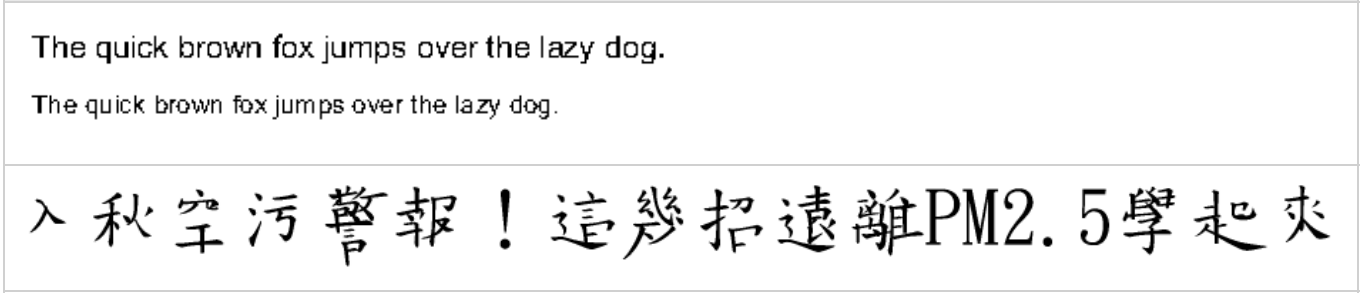
Hinting instructions are included in some TrueType fonts which improve their look by reusing some glyph parts in different glyphs regardless of their font size. TrueType hinting instructions, in DsPdf, supports drawing CJK characters as combinations of other smaller glyph pieces which enhances their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

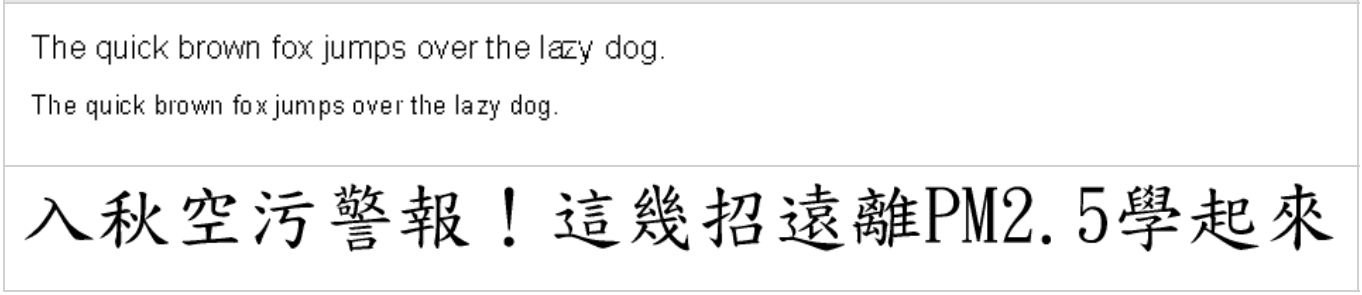
However, if the **EnableHinting** property is explicitly set to false, then the hinting instructions cannot be enabled.

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 Intructions



Enabled Hinting Intructions



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
```

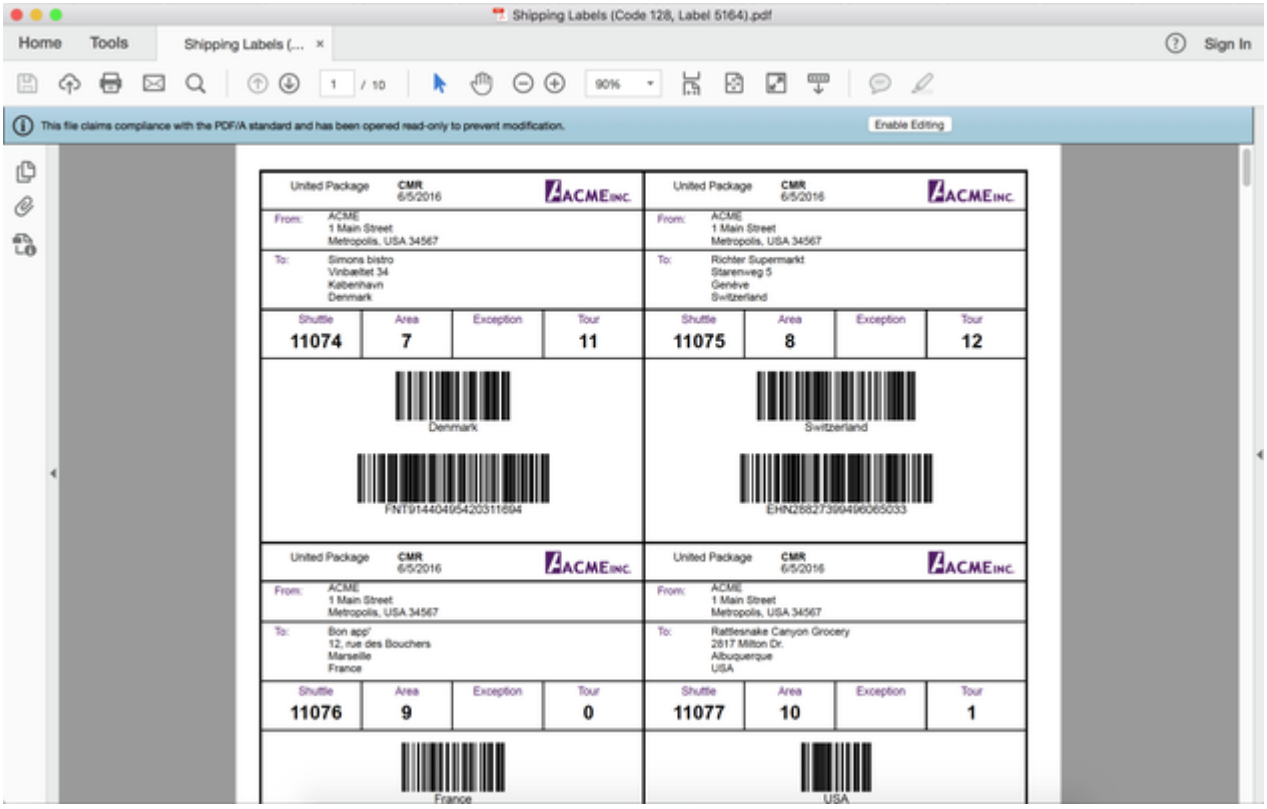
```
        string s1 = @"入秋空污警報！這幾招遠離PM2.5學起來";
        //Define text formatting attributes
        var tf1 = new TextFormat()
        {
            Font = font,
            FontSize = 20,

        };
        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.

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



Supported barcode symbologies

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

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

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

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

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

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

Back to Top

Barcode properties

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

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

Back to Top

Add Barcodes

To add barcode using DsPdf:

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

Barcode.cs

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

Back to Top

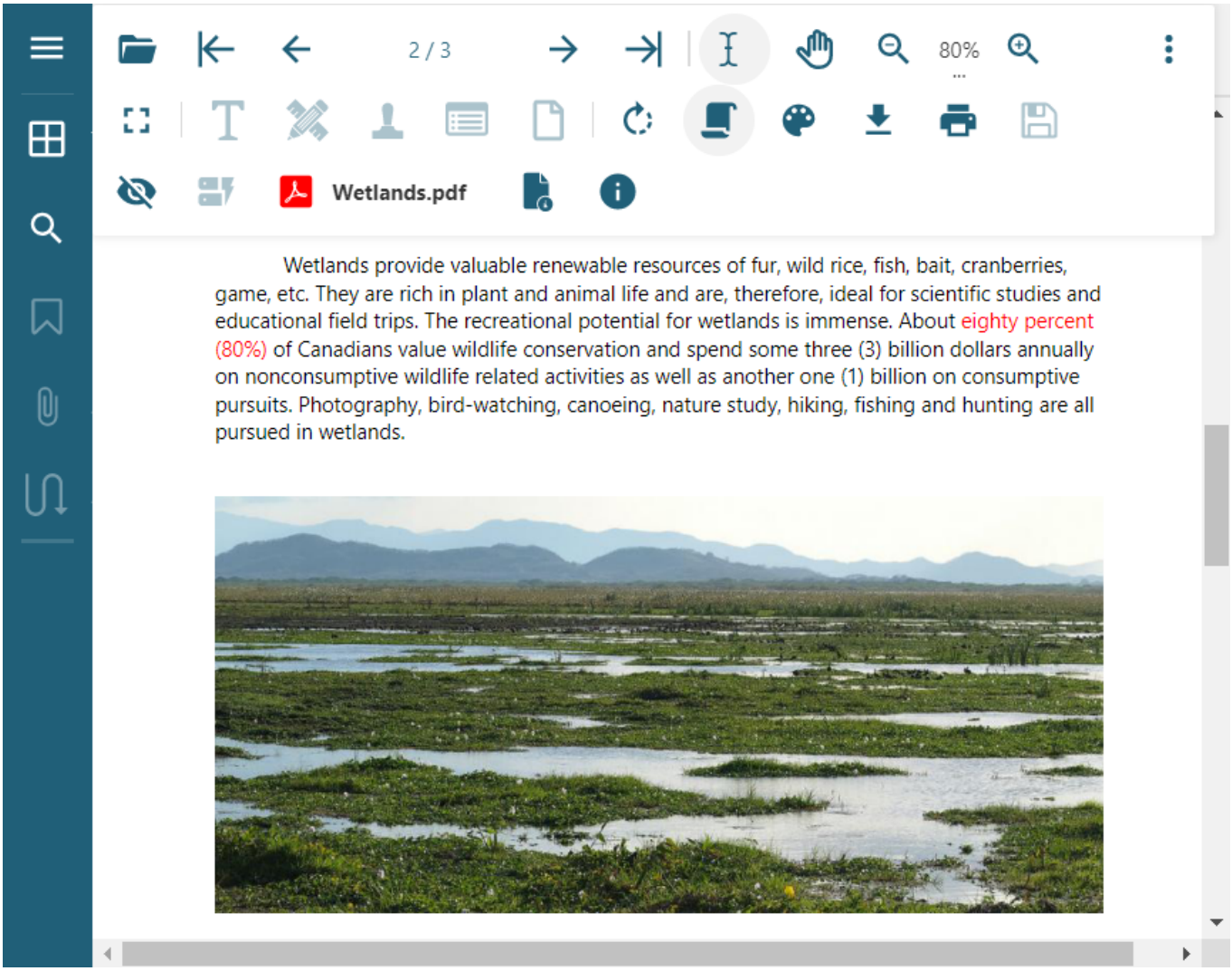
For more information about implementation of barcodes in DsPdf, see DsPdf sample browser.

Document Solutions PDF Viewer Overview

Document Solutions PDF Viewer (DsPdfViewer, previously GcPdfViewer) 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 DsPdfViewer 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 Document Solutions PDF Viewer with the DsPdf 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.



DsPdfViewer supports many standard PDF features:

- **Fill, submit and reset forms**
  The DsPdfViewer supports filling, submitting and resetting filled forms. To save them as a PDF on server, you

can also use Document Solutions for PDF API on the server.

- **Print filled forms**
  The DsPdfViewer allows you to directly print the filled-in forms from the Print option.
- **Print rotated document**
  The DsPdfViewer enables a user to rotate the document pages and directly print the rotated document.
- **Display page label titles**
  The DsPdfViewer supports the display of page label titles, so that you can distinguish the content topic in the document.
- **Annotations**
  The DsPdfViewer supports many annotations in the PDF document, without loss of any properties.
- **JavaScript actions**
  The DsPdfViewer supports JavaScript actions related to form fields, buttons and document.
- **Outline panel**
  The DsPdfViewer provides outline panel to list outlines and navigate to different positions in the document.
- **Text selection using caret**
  The DsPdfViewer supports selecting horizontal text, vertical and RTL text with the help of default text selection caret.
- **Password-protected documents**
  The DsPdfViewer supports documents that are password protected and lets you open PDF file through password input dialog.
- **Page-level and document-level attachments.**
  The DsPdfViewer supports both page-level and document-level attachments. The user can double click the attachment files to open the attachments.
- **Article threads**
  The DsPdfViewer supports navigating through article threads in a PDF file via a separate panel in the sidebar.
- **Edit PDF documents**
  The DsPdfViewer 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 DsPdfViewer 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

Document Solutions PDF Viewer supports the following types of license:

- **Evaluation License**
- **Production License**

Evaluation License

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

'*Powered by Document Solutions 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.

Production License

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

offers two types of licenses: standard and professional.

Standard License

Included with every DsPdf license purchase. This deployment includes all features of the viewer except those that require the viewer to use SupportApi property (i.e., connect the viewer to DsPdf on the server). For more information, see View PDF.

Professional License

Includes all the features of the Standard license in addition to features that require the viewer to use the SupportApi property (i.e., connect the viewer to DsPdf on the server). For more information, see Edit PDF.

Please refer to DsPdfViewer License Options to know more about Standard and Professional Viewer License.

How to apply your license key

To apply evaluation/production license in DsPdfViewer, set DsPdfViewer licensed deployment key to DsPdfViewer.License property before creating and initializing DsPdfViewer:

```
<script>
     // Add your license
     DsPdfViewer.LicenseKey = 'your_license_key';
     // Add your code
     window.onload = function(){
      const viewer = new DsPdfViewer("#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>PDF Viewer Demo | PDF Plugin</title>
    <script>
        function loadPdfViewer(selector) {
            DsPdfViewer.LicenseKey = 'your_license_key';
            var viewer = new DsPdfViewer(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="dspdfviewer.js"></script></body>
</html>
```

Redistribution

Note that the following files should be included while redistributing DsPdfViewer.

Script Files

- dspdfviewer.js
- dspdfviewer.worker.js

CSS Files

- dark-yellow.css
- light-blue.css
- viewer.css

> **Note:** If you are using the default theme, the above-mentioned css files can be excluded. If you want to use a different theme, make a "themes" subfolder in the folder where the viewer files are already placed. Place the css files specific to the themes in the subfolder. This will allow the themes to work automatically.

View PDF

Document Solutions 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

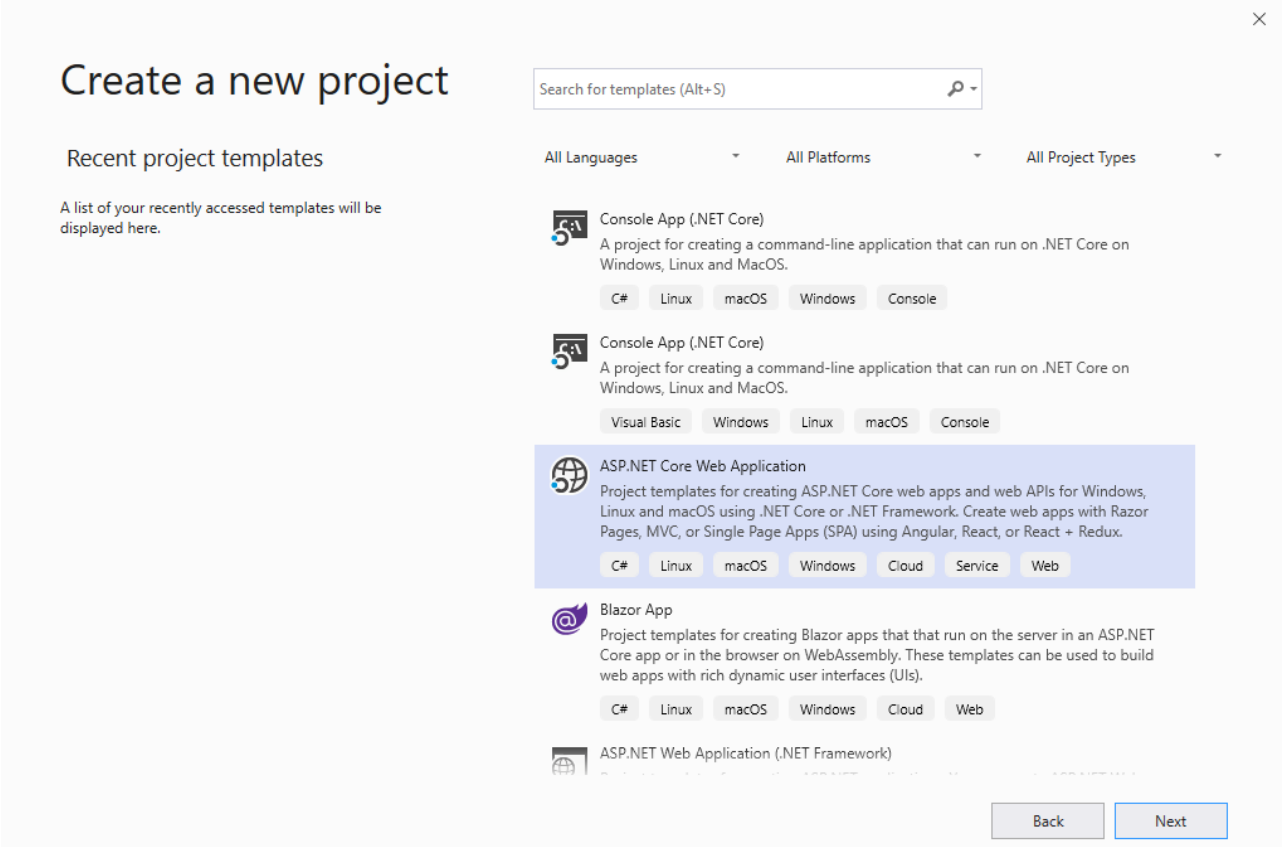
Configure PDF Viewer

Configuring PDF Viewer

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

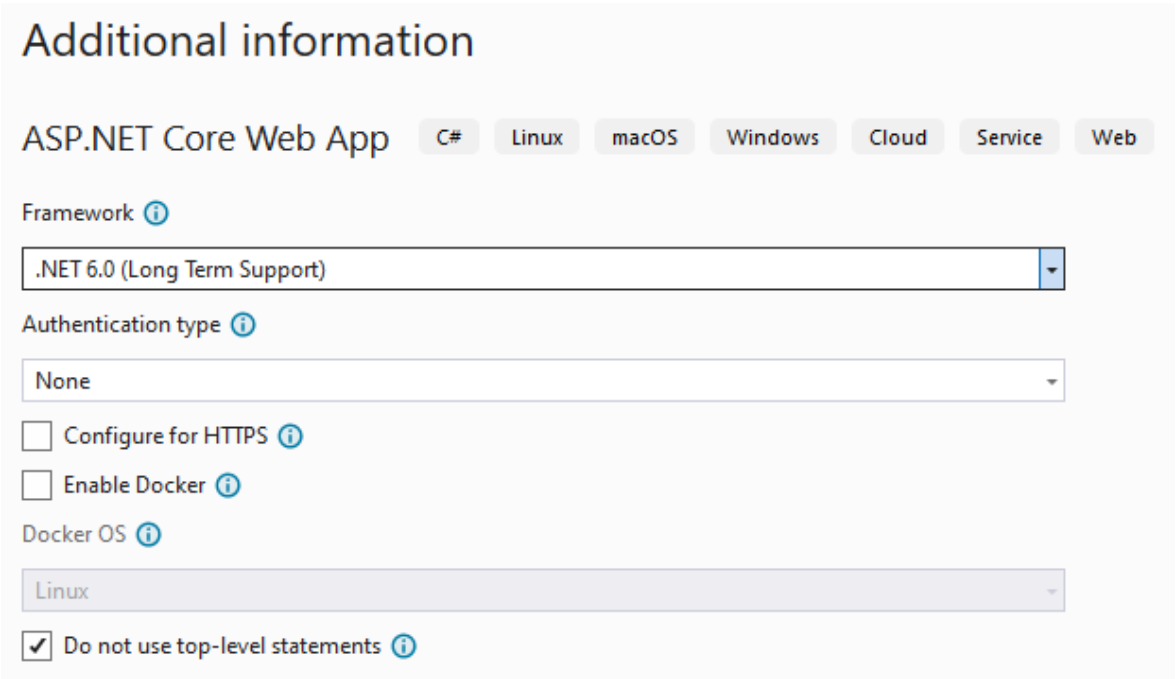
.NET 6/ .NET 7

1. Open Microsoft Visual Studio 2022 and select **Create a new project | ASP.NET Core Web Application**.



2. In the Create a new ASP.NET Core web application dialog, select NET Core 6.0 as the target framework.

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

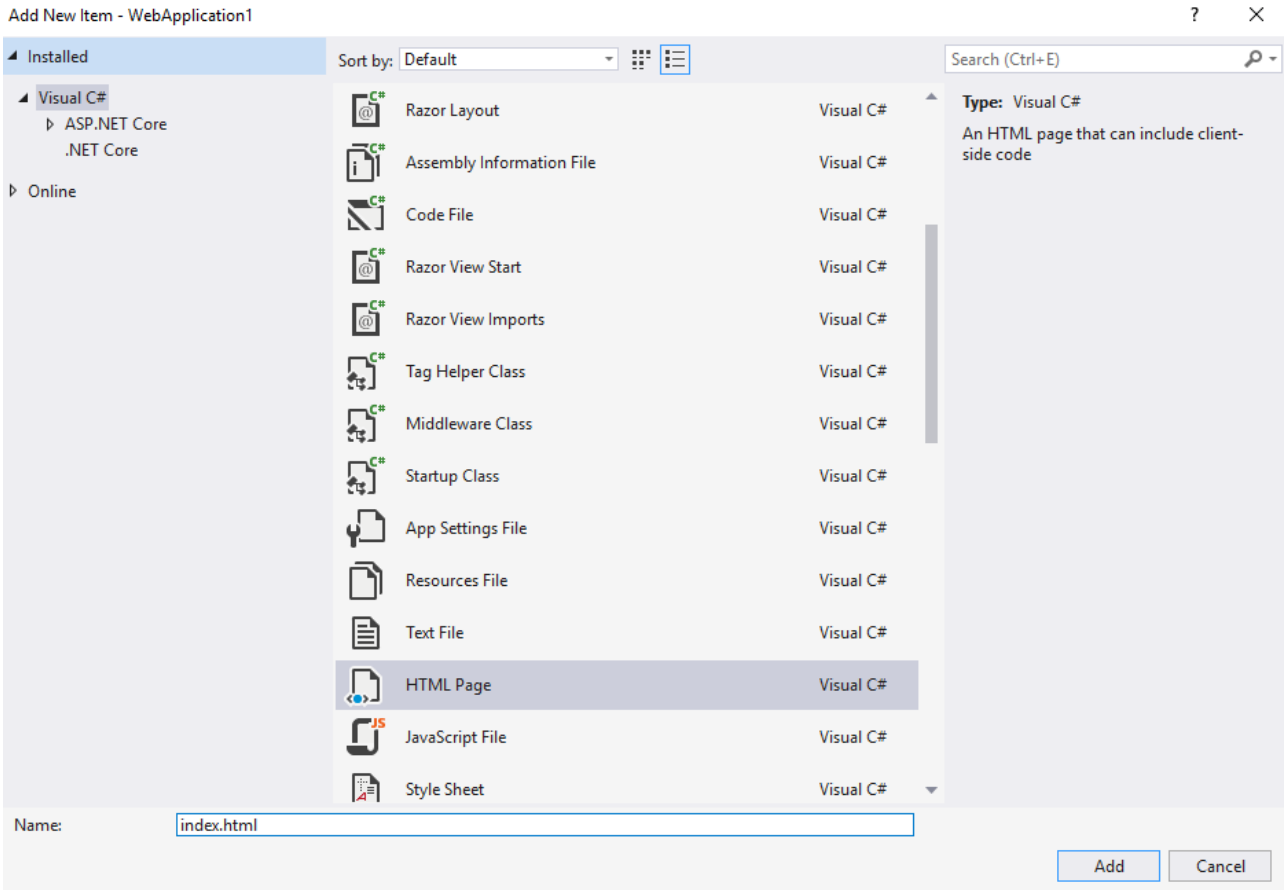


3. Make sure that sample project builds and runs fine (shows the 'Welcome' screen in browser). Next steps assume that the project is named as 'WebApplication1'.
4. Run the following command to install DsPdfViewer. Make sure that the directory location in command prompt is set to lib folder. The DsPdfViewer will be installed in WebApplication1\wwwroot\lib:

```
npm install @mescius/dspdfviewer
```

> **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\@mescius\dspdfviewer*.

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



6. 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>PDF 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 DsPdfViewer(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/@mescius/dspdfviewer/dspdfviewer.js "></script>
</body>
</html>
```

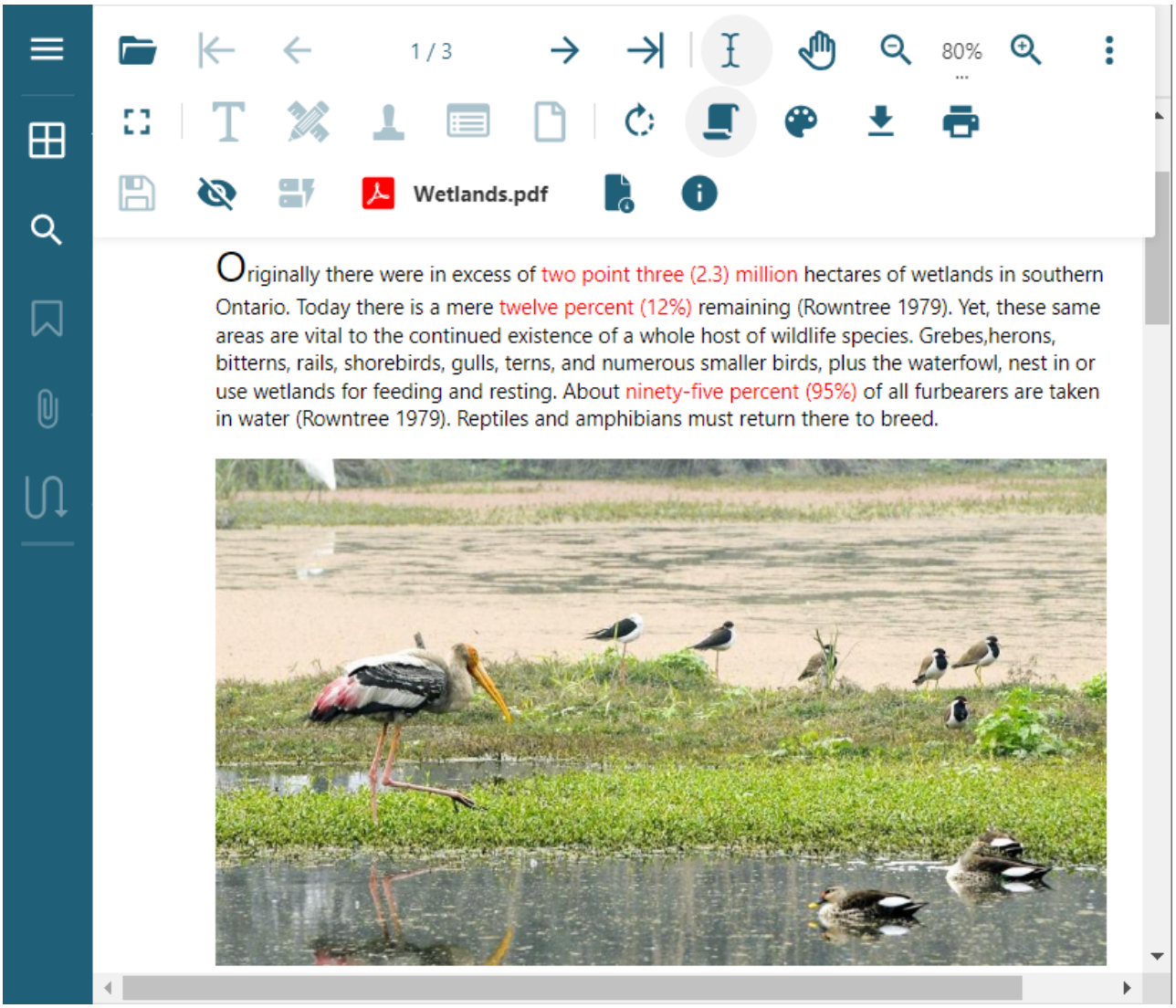
> **Note:** Besides adding DsPdfViewer to the page, the above code also loads a static PDF (Wetlands.pdf) into it on startup. To make sure it works, place the Wetlands.pdf in the wwwroot directory.

7. Modify the Program.cs file by replacing the existing code with the following code:

Program.cs

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddRazorPages();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Error");
    // The default HSTS value is 30 days. You may want to change this for production
scenarios, see https://aka.ms/aspnetcore-hsts.
    app.UseHsts();
}
app.UseHttpsRedirection();
app.UseDefaultFiles();
app.UseStaticFiles();
app.UseRouting();
app.UseAuthorization();
app.MapRazorPages();
app.Run();
```

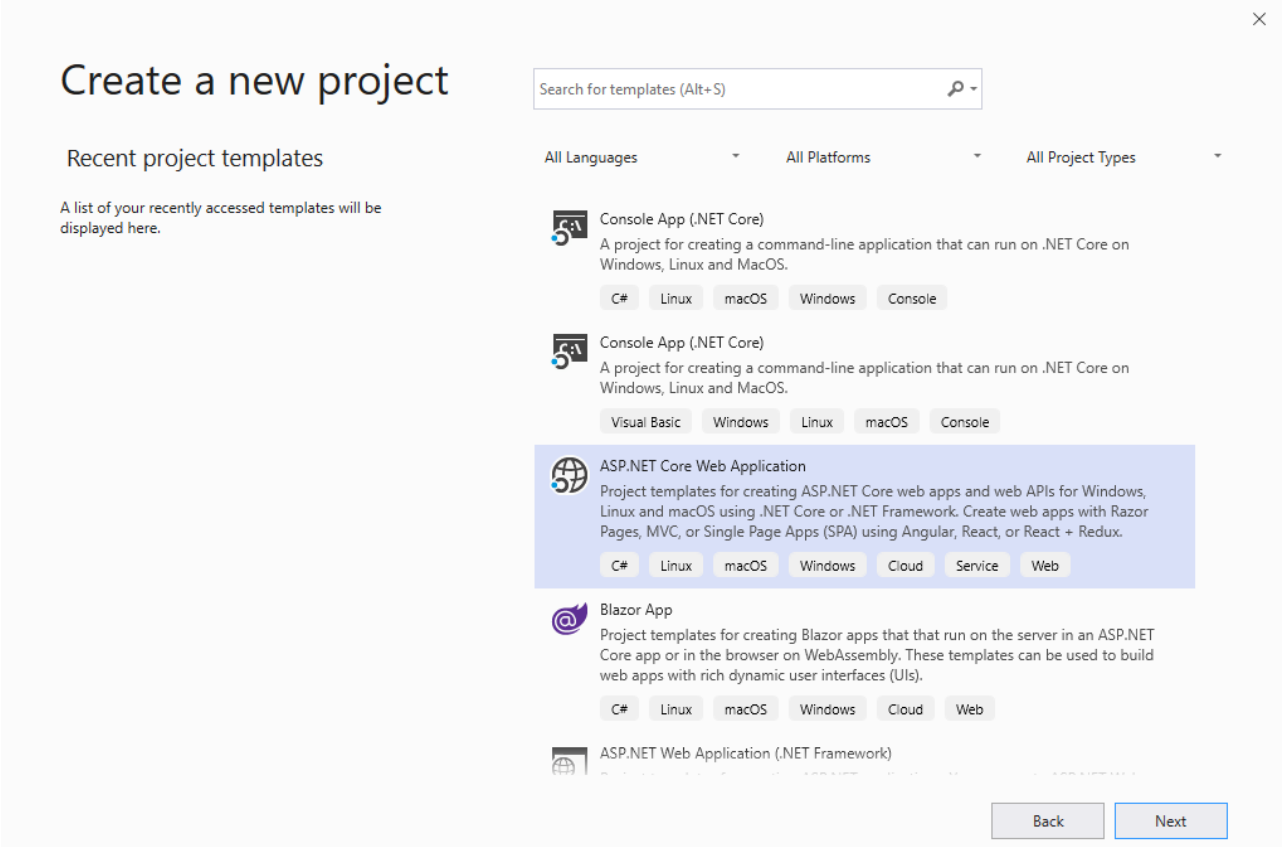
8. Build and run the application. A page with the DsPdfViewer (loaded with Wetlands.pdf) will show in your default browser.



For more information, refer View PDF in DsPdfViewer demos.

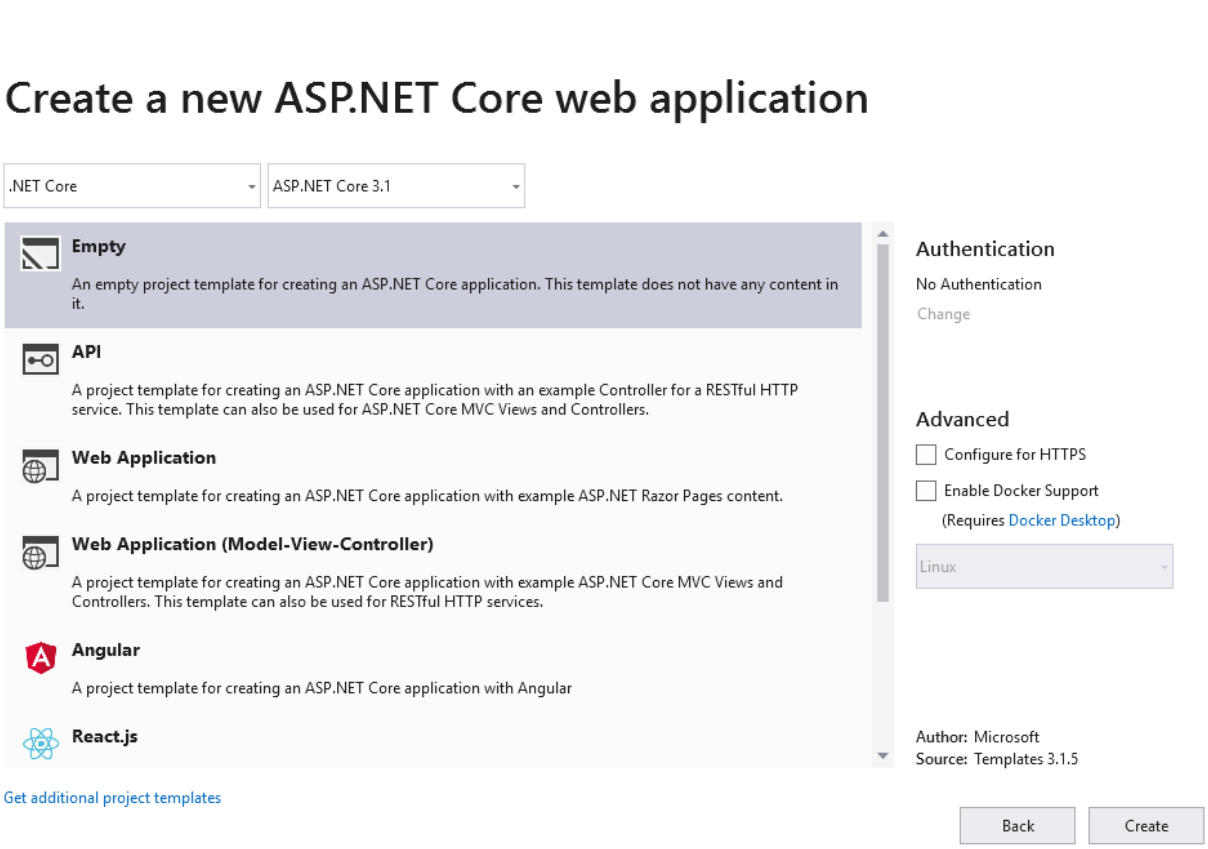
.NET Core 3.1/ .NET 5

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

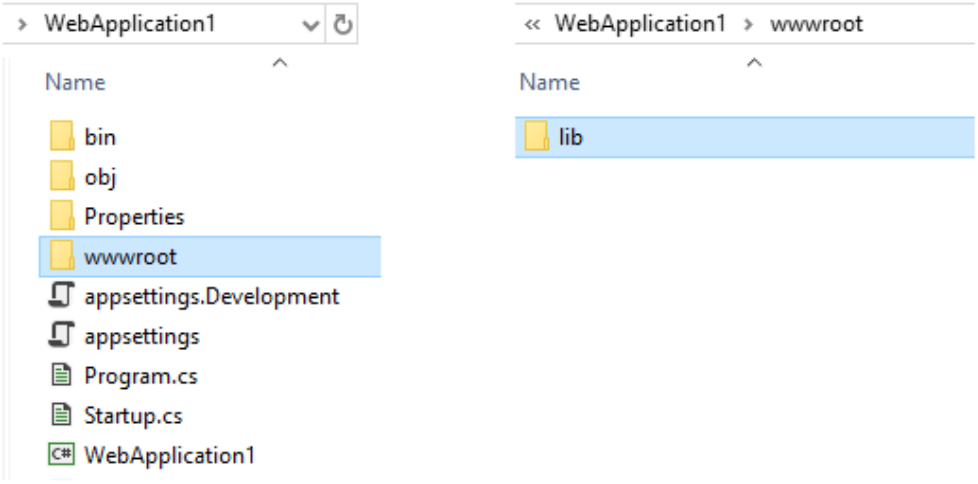


2. In the 'Create a new ASP.NET Core web application' dialog, select the following:
   - .NET Core / ASP.NET Core 3.1
   - 'Empty' - Project template

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



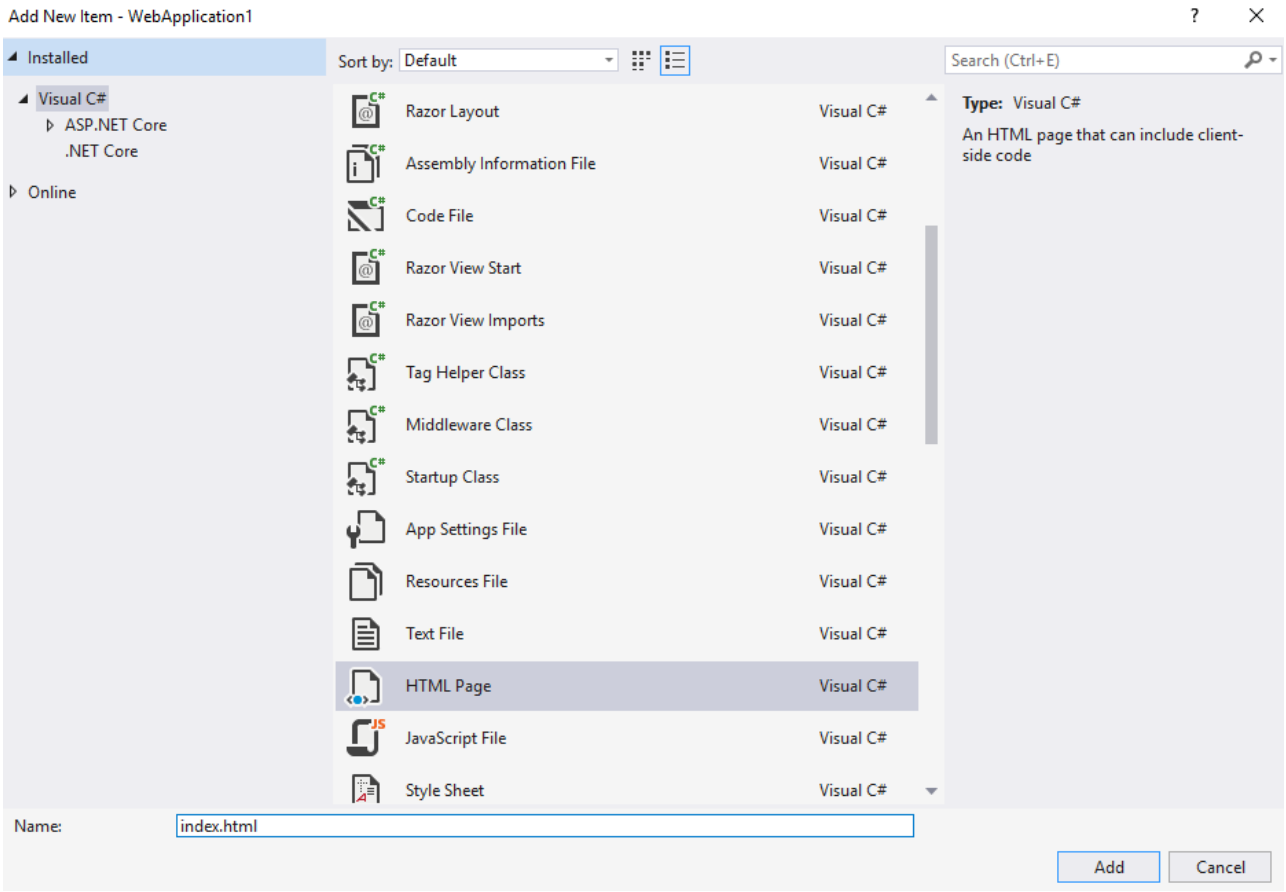
3. Make sure that sample project builds and runs fine (shows the 'Hello World!' screen in browser). Next steps assume that the project is named as 'WebApplication1'.
4. Open the project in File Explorer and create the 'wwwroot' and 'lib' directories as shown below: *WebApplication1\wwwroot WebApplication1\wwwroot\lib*



5. Run the following command to install DsPdfViewer. Make sure that the directory location in command prompt is set to *lib* folder. The DsPdfViewer will be installed in *WebApplication1\wwwroot\lib:* `npm install @mescius/dspdfviewer`

> **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\@mescius\dspdfviewer*.

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>PDF 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 DsPdfViewer(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/@mescius/dspdfviewer/dspdfviewer.js "></script>
</body>
</html>
```

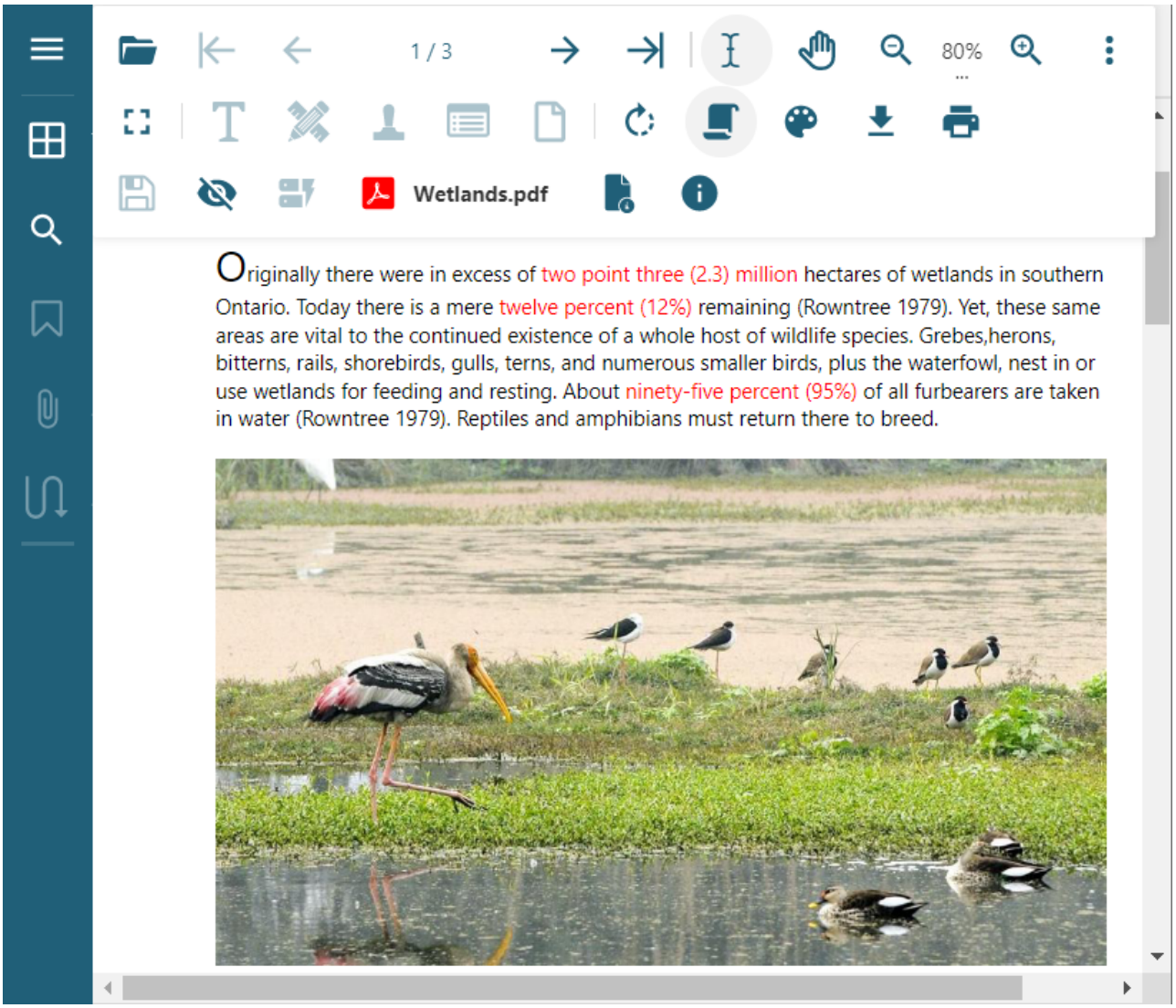
> **Note:** Besides adding DsPdfViewer to the page, the above code also loads a static PDF (Wetlands.pdf) into it on startup. To make sure it works, place the Wetlands.pdf in the wwwroot directory.

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

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddRazorPages();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Error");
    // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.
    app.UseHsts();
}
app.UseHttpsRedirection();
app.UseDefaultFiles();
app.UseStaticFiles();
app.UseRouting();
app.UseAuthorization();
app.MapRazorPages();
app.Run();
```

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



For more information, refer View PDF in DsPdfViewer demos.

Loading PDF Files

DsPdfViewer's open method supports loading PDF files from local path, same domain and another domain URL sources.

Load from Local File

To open a local PDF file in DsPdfViewer, click the **Open** button ( ) present on the top-left corner of the viewer. It opens the Open dialog to choose a PDF file.

You can also open a PDF file using code. Refer to the following example code to open a PDF file from local.

```
// Local path.
viewer.open("assets/pdf/newsletter.pdf");
```

Load from URL

To open a PDF file from the same domain URL in DsPdfViewer, pass the URL string in the open method.

Refer to the following example code to open a PDF file from same domain URL.

```
// URL path.
viewer.open("https://developer.mescius.com/documents-api-pdf/docs/newsletter.pdf");
```

Note: The above example code has been taken from a DsPdfViewer demo hosted on MESCIUS website. Also, it opens a file hosted on the MESCIUS website to make sure the URL belongs to the same domain.

Load from another Domain URL

To open a PDF file from another domain URL in DsPdfViewer, pass the URL string with CorsProxy in the open method.

Refer to the following example code to open a PDF file from another domain URL.

```
// Another domain URL path.
viewer.open("https://localhost:7288/api/pdf-viewer/CorsProxy?
url=https://developer.mescius.com/documents-api-pdf/docs/newsletter.pdf");
```

Note: SupportApi should be configured in order to use CorsProxy method. For more information, see Configure PDF Editor.

Features

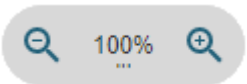
DsPdfViewer supports standard as well as advanced PDF viewer features:

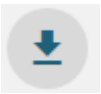
- Toolbar and Panel Icons
- Custom Context Menu
- Advanced Search Options
- View PDF Elements
- Initial View Settings

Toolbar and Panel Icons

DsPdfViewer 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 Document Solutions 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.
Search Bar		Allows you to search for text with some advanced options.
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.

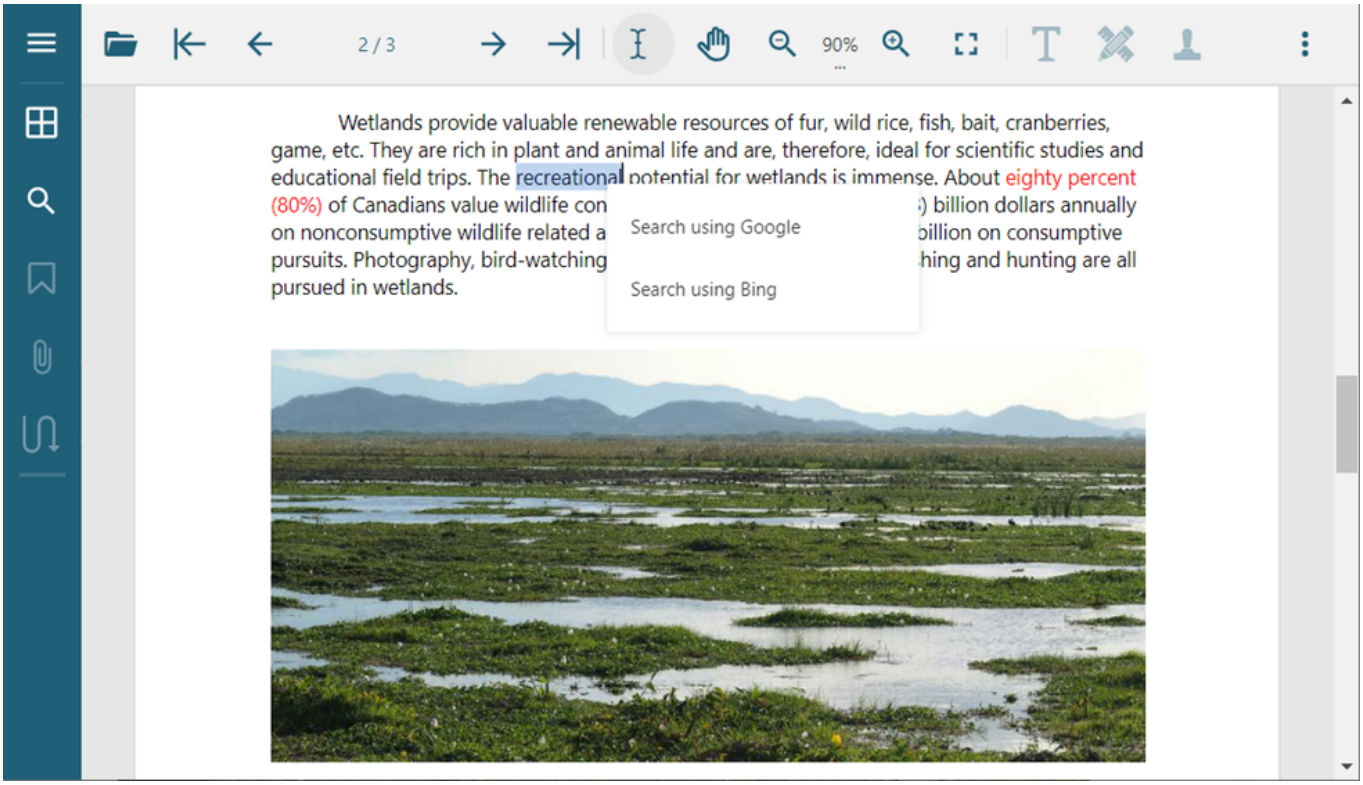
Features	Panel Icons	Description
Thumbnail navigation		Allows you to see the preview of all available pages in the PDF document.
Search		Allows you to search for text with match-case and whole-word search options. **Note:** You can use the search option as a floating search bar or it can be added to the sidebar. For more information, see Search.
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 DsPdfViewer, visit the DsPdf Sample browser. Here, you can see all the PDF features that are supported and running in the viewer.

Custom Context Menu

DsPdfViewer 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 DsPdfViewer:

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 DsPdfViewer's context menu to use browser's context menu by using below code:

Index.cshtml

```
var viewer = new DsPdfViewer("#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).

Search

DsPdfViewer lets you perform the Search operation using several advanced search options, such as match case, wild card, proximity, and more. By default, the Search icon is available in the toolbar, which, when clicked, opens the floating search bar. However, you can even place the Search panel in the sidebar by disabling the Floating search bar, as described ahead.

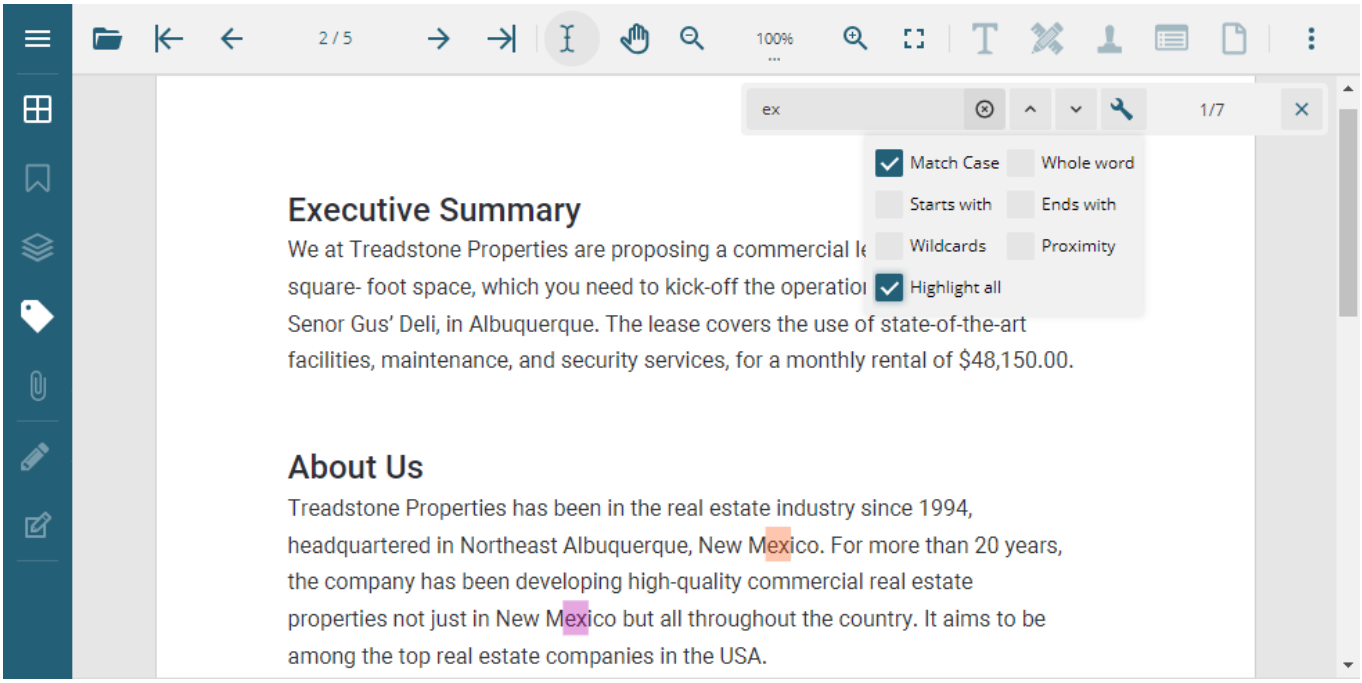
Floating Search Bar

DsPdfViewer provides a floating text search bar at the top-right corner, just below the toolbar. You can open the floating text search bar using the toolbar icon () or by pressing **Ctrl+F**. Then, you can initiate the search by typing in the desired search term in the designated search input field. As you start typing, the search mechanism automatically begins searching for matches within the document, highlighting or displaying results in real-time as you type. This dynamic search process allows immediate feedback and helps a user quickly locate the relevant content.

Note: When you press Ctrl+F again, DsPdfViewer will focus on the search input again if it is not already focused.

The floating text search bar provides various advanced search options in the **Settings** () button to customize and fine-tune your search preferences. It eliminates irrelevant options and narrows down the scope of a search query.

You can close the floating text search bar using the **Close** () button or by pressing the **ESC** key when the search input is focused.



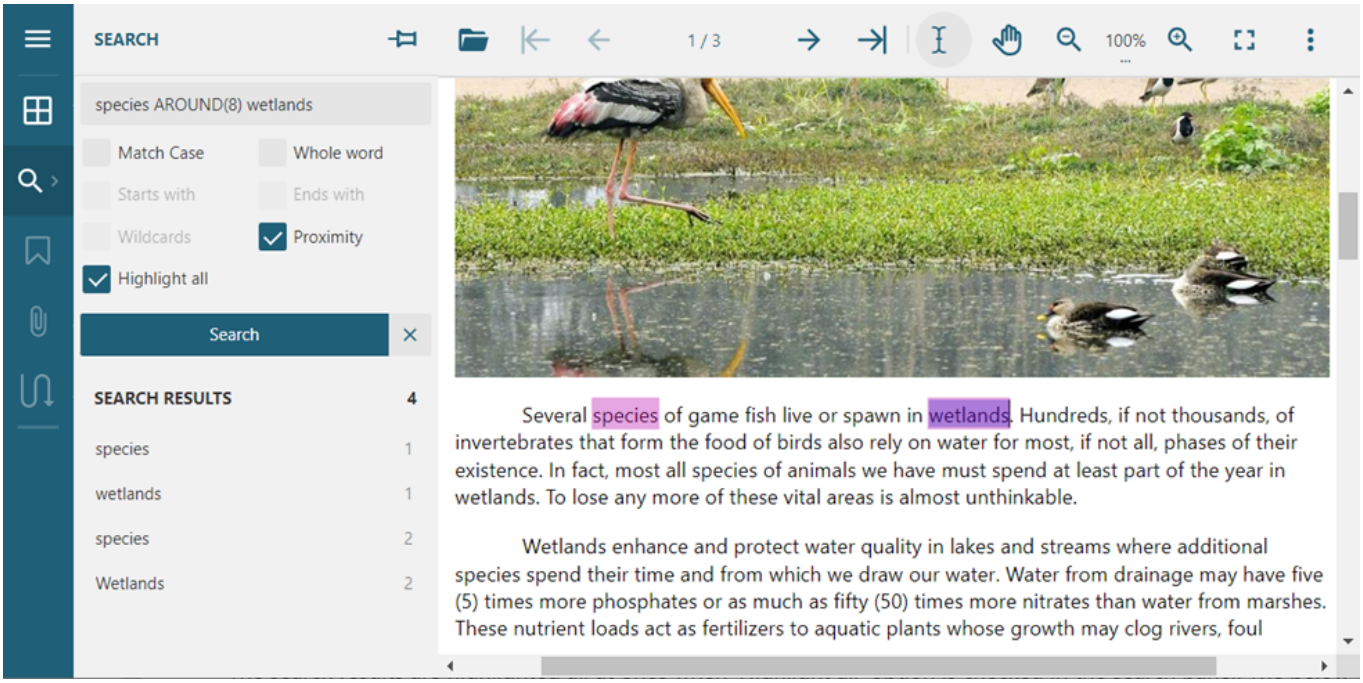
Sidebar Search Panel

You can also use the sidebar search panel by setting 'useFloatingSearchBar' option to false. The sidebar search panel also provides advanced search options, along with displaying the count of total search results and the page number on which the specific result is found.

Refer to the following example code to use the sidebar search panel:

C#

```
// Use sidebar search panel instead of floating search bar.
var viewer = new DsPdfViewer("#root", { useFloatingSearchBar: false });
```



Advance Search Options

The following sections explain the various advanced search options provided in DsPdfViewer:

- **Match Case**
- **Whole Word**
- **Starts With**
- **Ends With**
- **Wildcards**
- **Proximity**
- **Highlight all**

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 displays "**test**", "**test**s", "**test**ing" 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 displays "**demo**", "**Demo**" but not "demonstration", "demos".

Starts With

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

Example: If you search "us", the search results display "**us**", "**us**ed", "**Us**ers" but not "various", "status".

Ends With

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

Example: If you search "at", the search results display "**at**", "form**at**", "tre**at**" 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. DsPdfViewer 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 "**te**ntative", "**te**xt", "ex**te**nsive", "poli**te**"
- **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 display "**tre**at", "star**te**r", "elabora**te**", "cen**tre**"

> **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.

Highlight all

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 DsPdfViewer('#root',
    {
        useCanvasForSelection:
        {
            selectionColor: 'rgba(0, 0, 195, 0.25)',
            highlightColor: 'rgba(255, 0, 0, 0.35)',
            inactiveHighlightColor: "rgba(180, 0, 170, 0.35)"
        }
    });
```

View PDF Elements

DsPdfViewer 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. DsPdf allows you to work with layers and set their properties, refer Layers for more information.

In DsPdfViewer, 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.

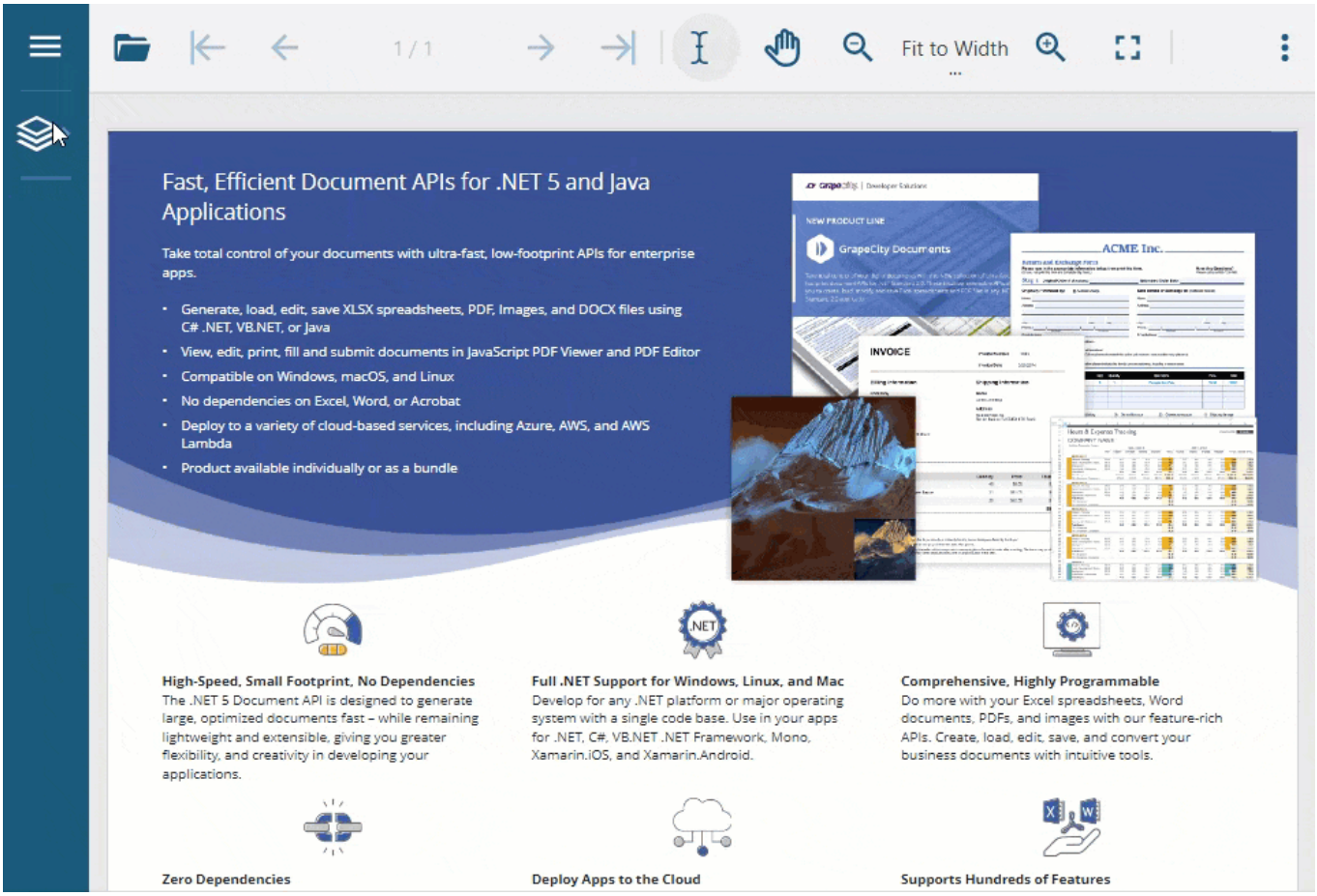
Enable Layers Panel in DsPdfViewer

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 DsPdfViewer Layers panel.



View Zoom Dependent Layers

According to the PDF specifications 2.0, groups of optional content layers in a PDF can be associated with specific ranges of magnification (zoom values) at which they should be made visible in a compliant viewer. DsPdfViewer supports loading of such zoom-dependent layers in a PDF, and the layers will be automatically shown or hidden

depending on the current zoom factor.

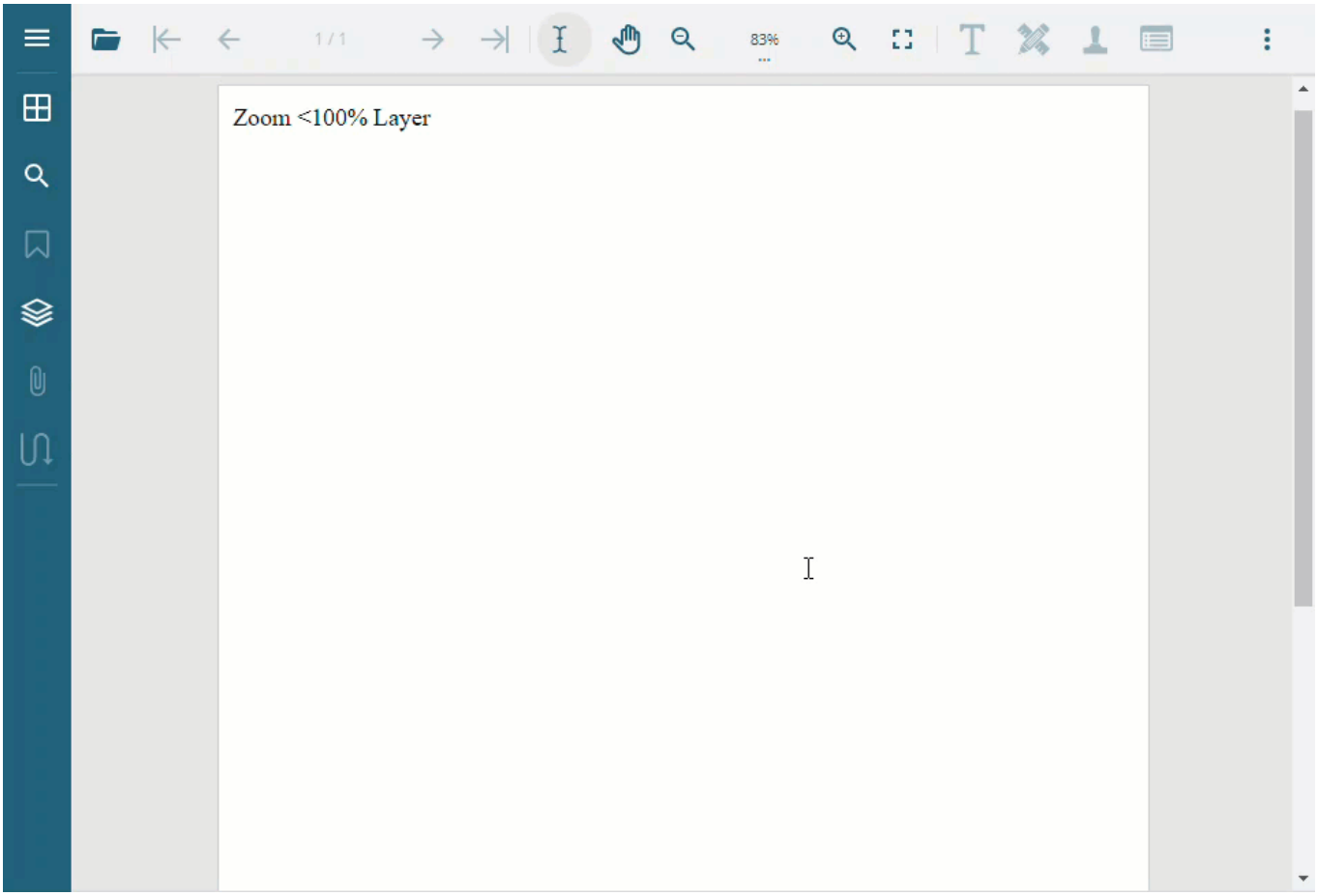
The visibility flags in the viewer reflect the zoom factor changes applied to the zoom-dependent layers by checking or unchecking the checkbox for visible or not visible layers, respectively, depending on the current zoom value of the viewer. DsPdfViewer implements the said behavior based on PDF specifications 2.0, which state that zoom is a dictionary specifying a range of magnifications at which the content in this optional content group is best viewed. Zoom contains one or both of the following:

- min: The minimum recommended magnification factor at which the group shall be ON. The default value is 0.
- max: The maximum magnification factor below which the group shall be ON. The default value is infinity.

The max zoom factor value is excluded, which means that if the max zoom factor for a layer is 2 (200%), and the view zoom is 200%, then the layer will not be visible. The following table demonstrates relationships between view zoom value and layer visibility:

Layer min and max Values	View Zoom Value	Result Layer Visibility
min: 1 (100%) max: 2 (200%)	99%	Not Visible
min: 1 (100%) max: 2 (200%)	100%	Visible
min: 1 (100%) max: 2 (200%)	150%	Visible
min: 1 (100%) max: 2 (200%)	199%	Visible
min: 1 (100%) max: 2 (200%)	200%	Not Visible
min: 1 (100%) max: 2 (200%)	201%	Not Visible

The below GIF image depicts the visibility of optional zoom-dependent layers in a PDF document depending on the zoom value of the viewer:



Note: This feature is available in both the standard and professional versions of DsPdfViewer. However, SupportApi needs to be configured in order to save a PDF document as images at a specific zoom factor.

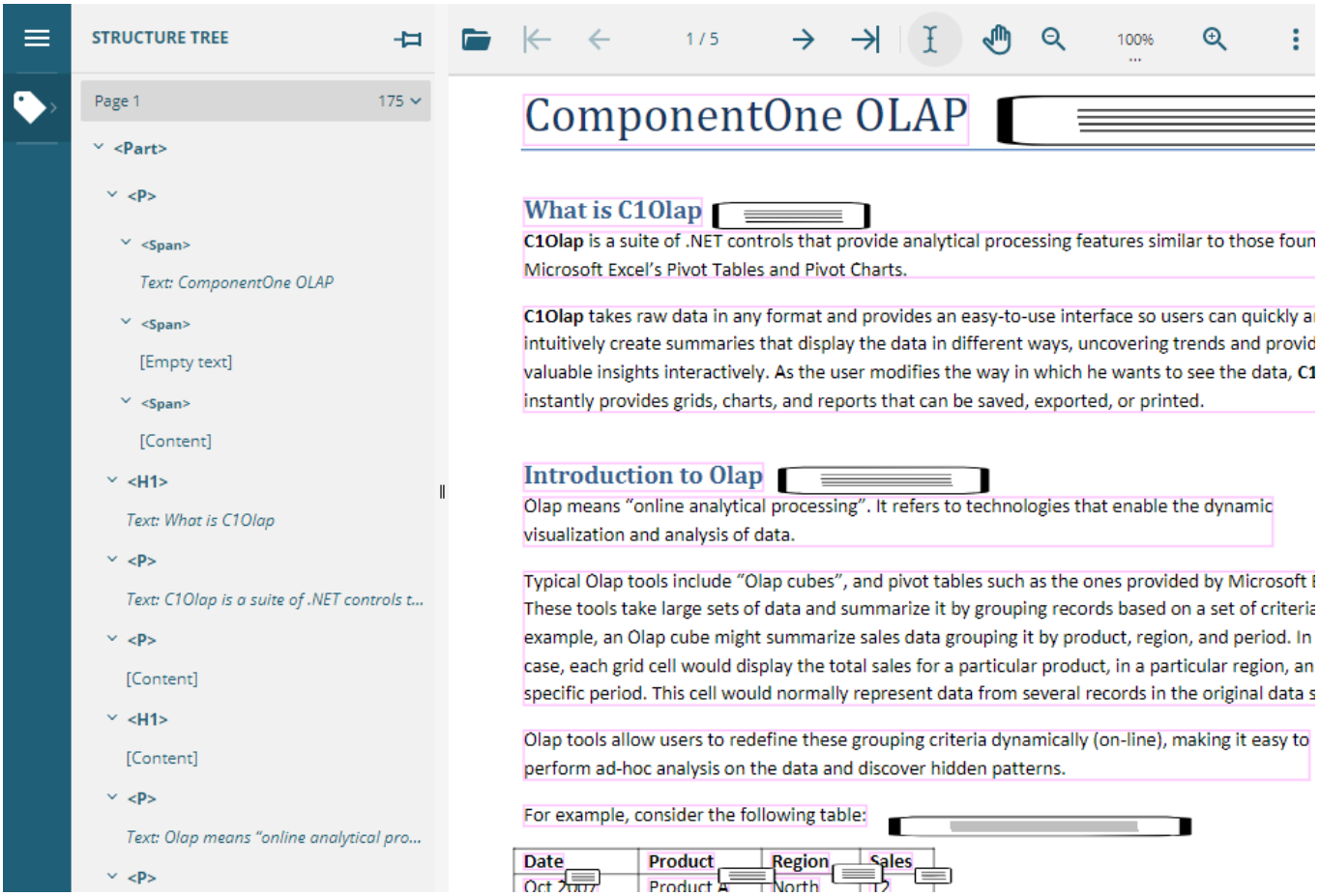
Limitation

The editing of the layers is not supported, i.e., you cannot change the visibility of a layer using the Layers panel if the visibility of the layer depends on the zoom factor.

View Structured Content in Tagged PDF

DsPdf allows you to create or modify tagged PDF documents. Refer Tagged PDF to know more. You can use the Structure Tree panel of DsPdfViewer 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 DsPdfViewer. The number in the page header indicates the total number of items in the structure tree on that page (excluding the root element).



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

Index.cshtml

```
var viewer = new DsPdfViewer(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.

DsPdfViewer supports displaying PDF documents containing XFA content, by default. It also allows you to select and copy the content from XFA forms. Moreover, DsPdfViewer 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 DsPdfViewer(selector, { enableXfa: false });
```

Limitation

- XFAF (XFA Foreground) subset is not supported.

- Editing and save operations are not supported in XFA forms.

Initial View Settings

DsPdfViewer supports opening a PDF document with initial view settings similar to Acrobat. The initial view settings determine how the document will be initially displayed when opened in the DsPdfViewer. DsPdfViewer client-side API provides a **viewerPreferences** type that gets information about the initial view settings defined for the PDF document, such as **openAction**, **pageMode**, and **pageLayout**.

The viewerPreferences property of DsPdfViewer gets the initial view settings information set by a user. Refer to the following example code, which depicts the same:

```
var viewerPreferences = await viewer.viewerPreferences;
```

The following table lists the settings that can accessed through viewerPreferences:

Client-Side API	Description
openAction	The openAction gets the initial open action information set by the user.
pageMode	The pageMode gets the initial page mode information set by the user.
pageLayout	The pageLayout gets the initial page layout information set by the user.

Refer to the following example code to get the open action initial view settings:

```
var viewerPreferences = await viewer.viewerPreferences;
var openAction = viewerPreferences.openAction;
```

Refer to the following example code to find open action destination page index:

```
// Find open action destination page index.
const openAction = await viewer.openAction;
if(openAction && openAction.dest) {
  const pageRef = openAction.dest[0];
  const targetPageIndex = await viewer.resolvePageIndex(pageRef);
}
```

Refer to the following example code to get the page mode initial view settings:

```
var viewerPreferences = await viewer.viewerPreferences;
var pageMode = viewerPreferences.pageMode;
```

Refer to the following example code to get the page layout initial view settings:

```
var viewerPreferences = await viewer.viewerPreferences;
var pageLayout = viewerPreferences.pageLayout;
```

DsPdfViewer also provides a **ignoreInitialView** option if you wish to load the PDF document without considering the initial view settings specified in the PDF document. Refer to the following example code to ignore the initial view settings:

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

Note: This feature is available in both the standard and professional versions of DsPdfViewer.

Limitation

There are a few settings that are currently not supported by DsPdfViewer. The unsupported settings are as follows:

Options	Initial View Settings
Page layout	• Single Page Continuous • Two-Up (Facing) • Two-Up Continuous (Facing) • Two-Up (Cover Page) • Two-Up Continuous (Cover Page)
User Interface Options	• Hide menu bar • Hide window controls
Window Options	• Resize window to initial page • Center window on screen • Show (File Name and Document Title)

Edit PDF

Document Solutions 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
- PDF Organizer
- Comments Tool
- Graphical Signature Tool
- Digital Signature
- Share and Collaborate

Configure PDF Editor

When connected to a server running DsPdf (using the SupportApi property, see below), DsPdfViewer allows you to edit PDF documents by using the general editing options, annotations and form editor tools. To configure **DsPdfViewer** for editing PDF documents, you need to download **SupportAPI** to connect to Document Solutions for PDF (DsPdf) on server, which enables the PDF editing operations and saves PDF documents on client. The DsPdfViewer works with server-side API, that is, DsPdf 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 DsPdfViewer for Editing

1. A PDF is loaded into the DsPdfViewer in one of the following ways:
   - Using DsPdf
   - 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.

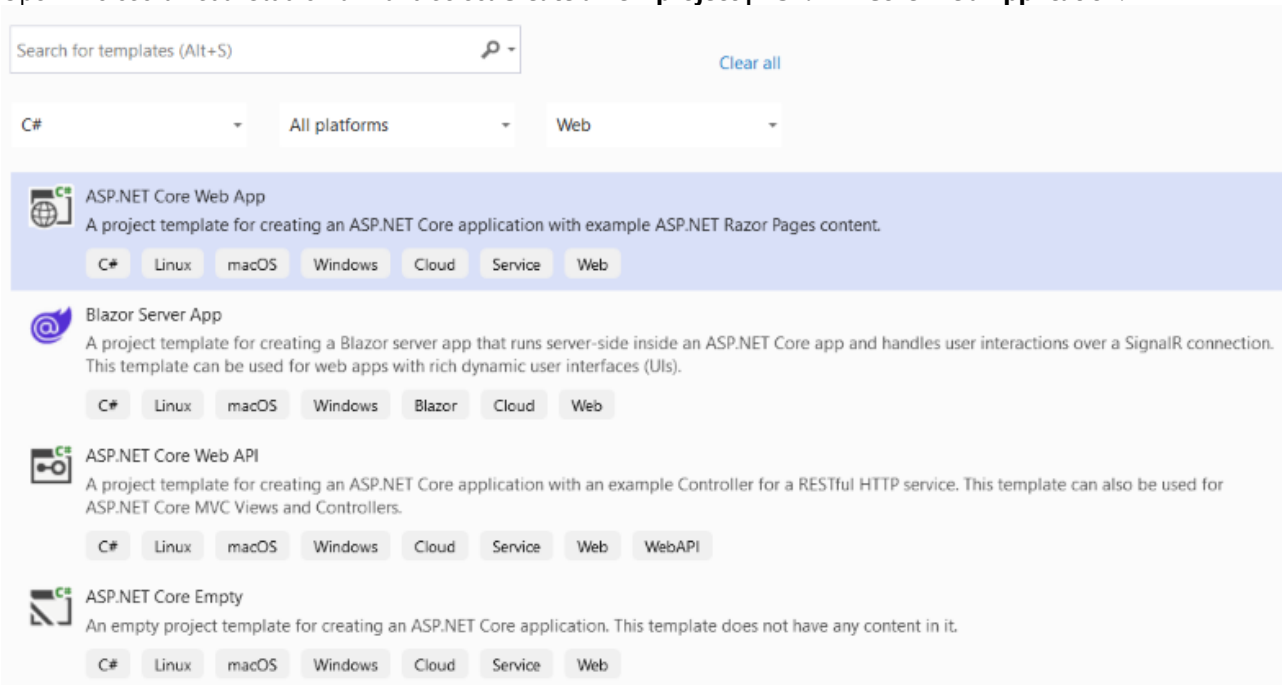
Using the DS.Documents.Pdf.ViewerSupportApi package

The **DS.Documents.Pdf.ViewerSupportApi** package can be downloaded from NuGet and is also included in **nupkg** folder of the DsPdf distribution zip along with other packages. You need add the package in your project references while creating ASP .NET Core Web Application and ASP .NET WebForms Application.

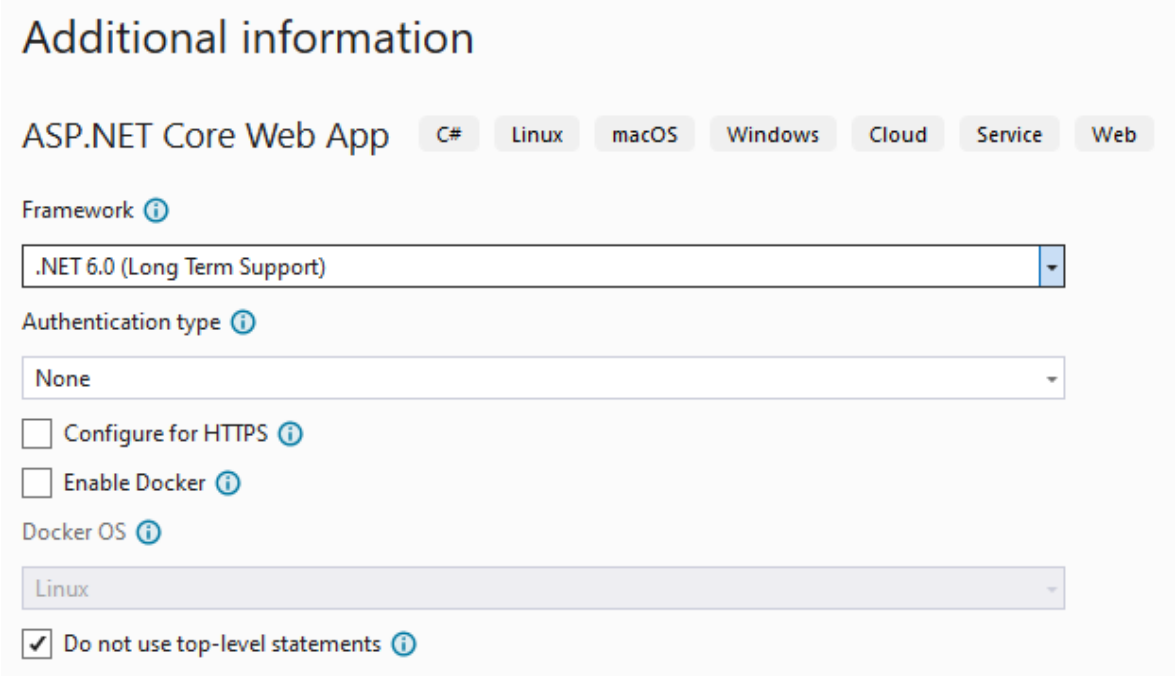
Web Application in .NET 6/ .NET 7

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

1. Open Microsoft Visual Studio 2022 and select **Create a new project | ASP.NET Core Web Application**.

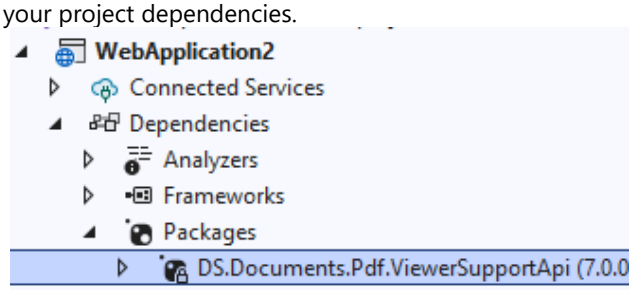


2. In the Create a new ASP.NET Core web application dialog, select NET Core 6.0 as the target framework.



> **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 'WebApplication1'.
4. Run the following command to install DsPdfViewer. Make sure that the directory location in command prompt is set to lib folder. The DsPdfViewer will be installed in WebApplication1\WebApplication1\wwwroot\lib:
   `npm install @mescius/dspdfviewer`
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 "DS.Documents.Pdf.ViewerSupportApi".
8. On the left panel, select **DS.Documents.Pdf.ViewerSupportApi** as shown in image below:
9. On the right panel, click **Install** to install the **DS.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 **DS.Documents.Pdf.ViewerSupportApi** package is added to your project dependencies.



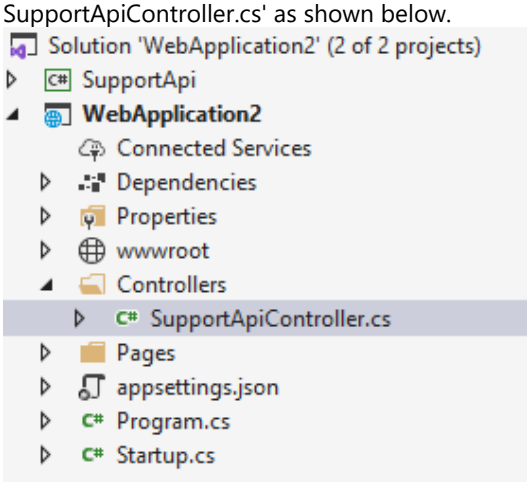
10. Modify the default content of WebApplication2\WebApplication2\Pages\Index.cshtml with the following code:

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/@@mescius/dspdfviewer/dspdfviewer.js" asp-append-
version="true"></script>
<script>
    var viewer = new DsPdfViewer("#host", { supportApi: 'api/pdf-viewer' });
    viewer.addDefaultPanels();
    viewer.addAnnotationEditorPanel();
    viewer.addFormEditorPanel();
    viewer.beforeUnloadConfirmation = true;
    viewer.newDocument();
</script>
```

11. Create a 'Controllers' folder in project and add a Controller choosing the MVC Controller – Empty template, naming it SupportApiController.cs' as shown below.



12. 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
    {

    }
}
```

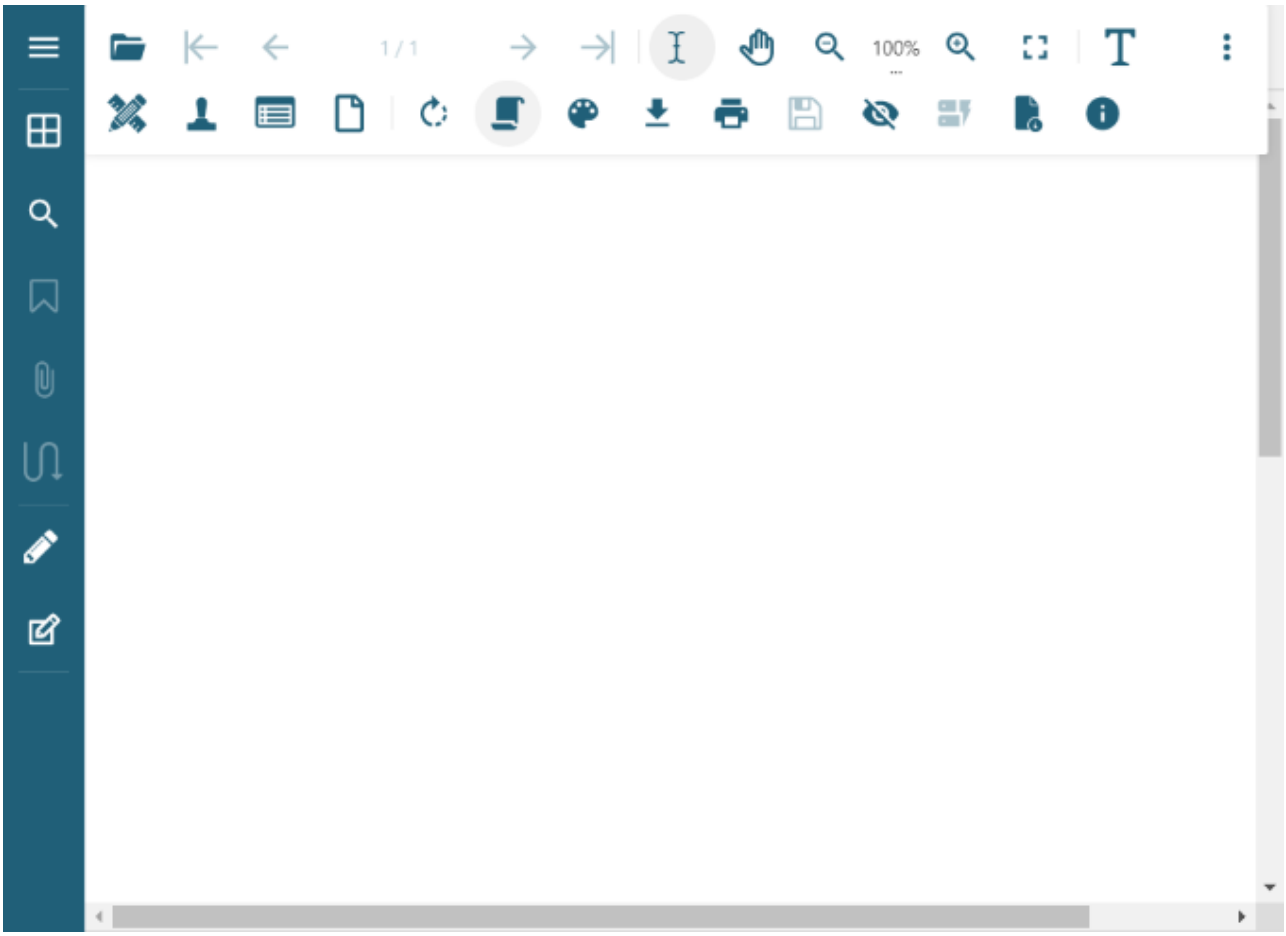
13. Modify **Program.cs** by adding the following lines of code to default code available on the page:

C#

```
using GrapeCity.Documents.Pdf.ViewerSupportApi.Connection;
using GrapeCity.Documents.Pdf.ViewerSupportApi.Controllers;
using GrapeCity.Documents.Pdf.ViewerSupportApi.Models;
using ProtoBuf.Meta;
using System.Reflection;
using System.Security.Cryptography.X509Certificates;
```

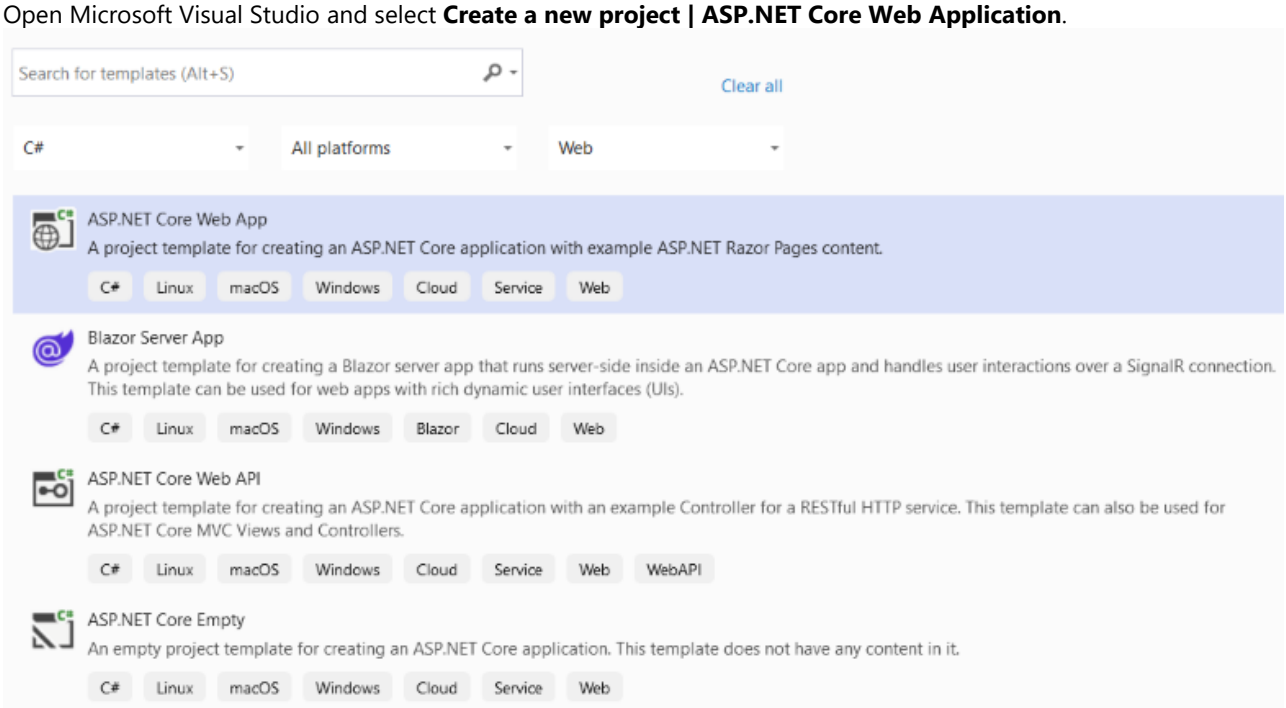
```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddRazorPages();
builder.Services.AddCors();
GcPdfViewerHub.ConfigureServices(builder.Services);

//Server side signing implementation
var app = builder.Build();
app.UseCors("WebCorsPolicy");

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
app.UseExceptionHandler("/Error");

// The default HSTS value is 30 days. You may want to change this for production
scenarios, see https://aka.ms/aspnetcore-hsts.
app.UseHsts();
}
app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();
app.UseAuthorization();

// Suggest using top level route registrations
IApplicationBuilder applicationBuilder = app.UseEndpoints(endpoints =>
{
endpoints.MapControllers();
});

// Suggest using top level route registrations
app.MapRazorPages();
app.Run();
```

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

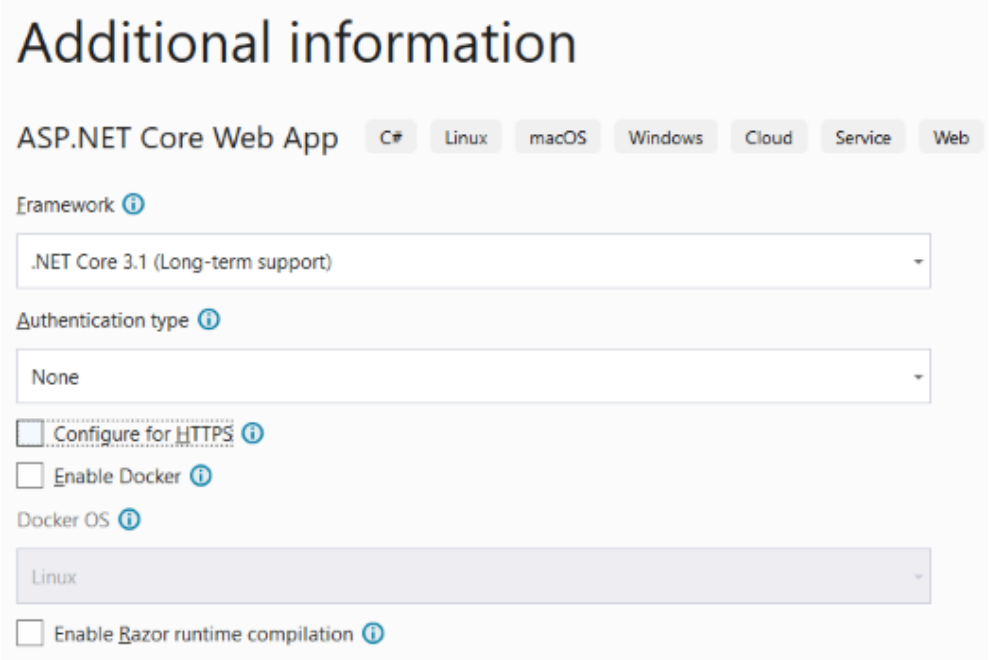


Web Application in .NET Core 3.1/ .NET 5

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

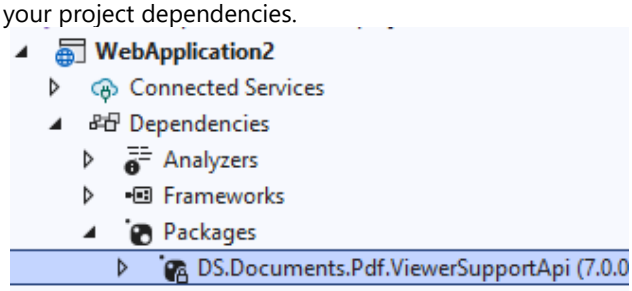


2. In the **Create a new ASP.NET Core web application** dialog, select the following:
NET Core 3.1(Long-term support)



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 DsPdfViewer. Make sure that the directory location in command prompt is set to lib folder. The DsPdfViewer will be installed in WebApplication2\WebApplication2\wwwroot\lib:
   `npm install @mescius/dspdfviewer`
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 "DS.Documents.Pdf.ViewerSupportApi".
8. On the left panel, select **DS.Documents.Pdf.ViewerSupportApi** as shown in image below:
9. On the right panel, click **Install** to install the **DS.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 **DS.Documents.Pdf.ViewerSupportApi** package is added to your project dependencies.



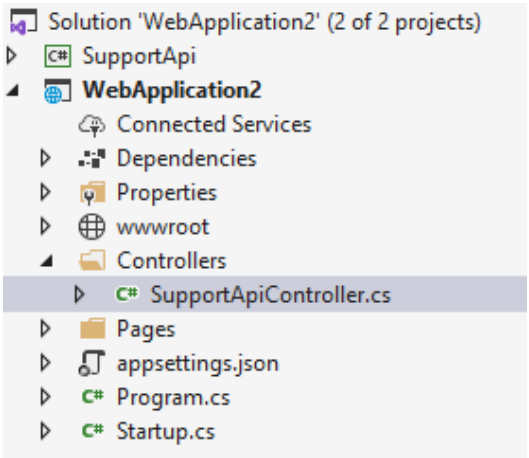
10. Modify the default content of WebApplication2\WebApplication2\Pages\Index.cshtml with the following code:

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/@@mescius/dspdfviewer/dspdfviewer.js" asp-append-
version="true"></script>
<script>
    var viewer = new DsPdfViewer("#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. 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
    {

    }
}
```

13. 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();
    ```

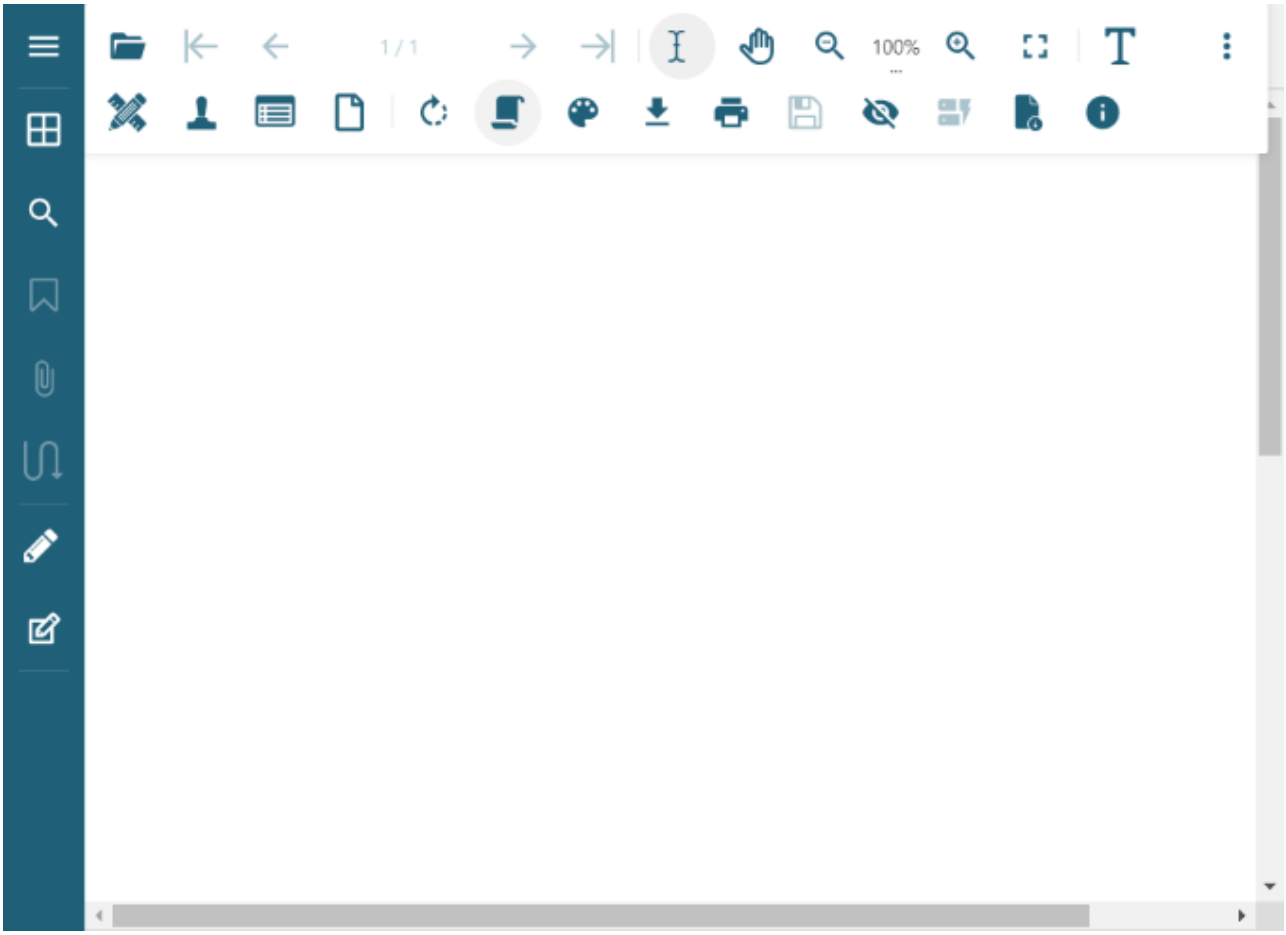
    The final startup.cs will look like below:

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();
```

```
            }
        }
}
```

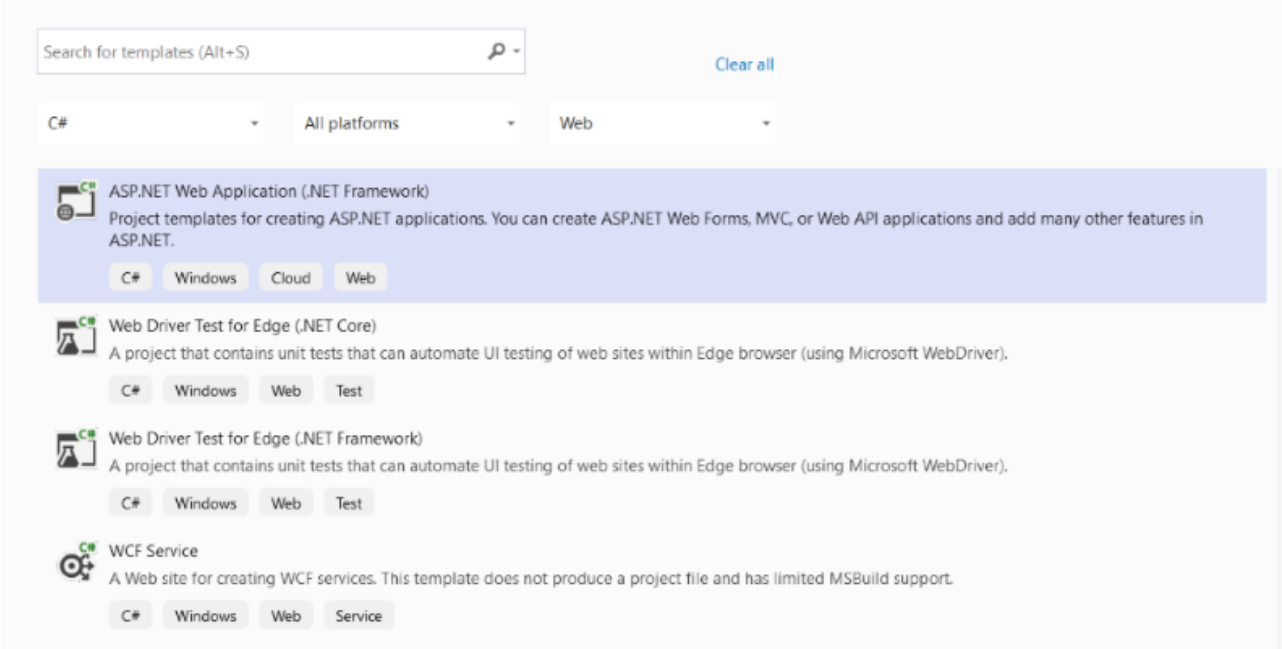
14. Build and run the application to view DsPdfViewer in your browser which contains the annotation and form editor tools to edit PDF documents.



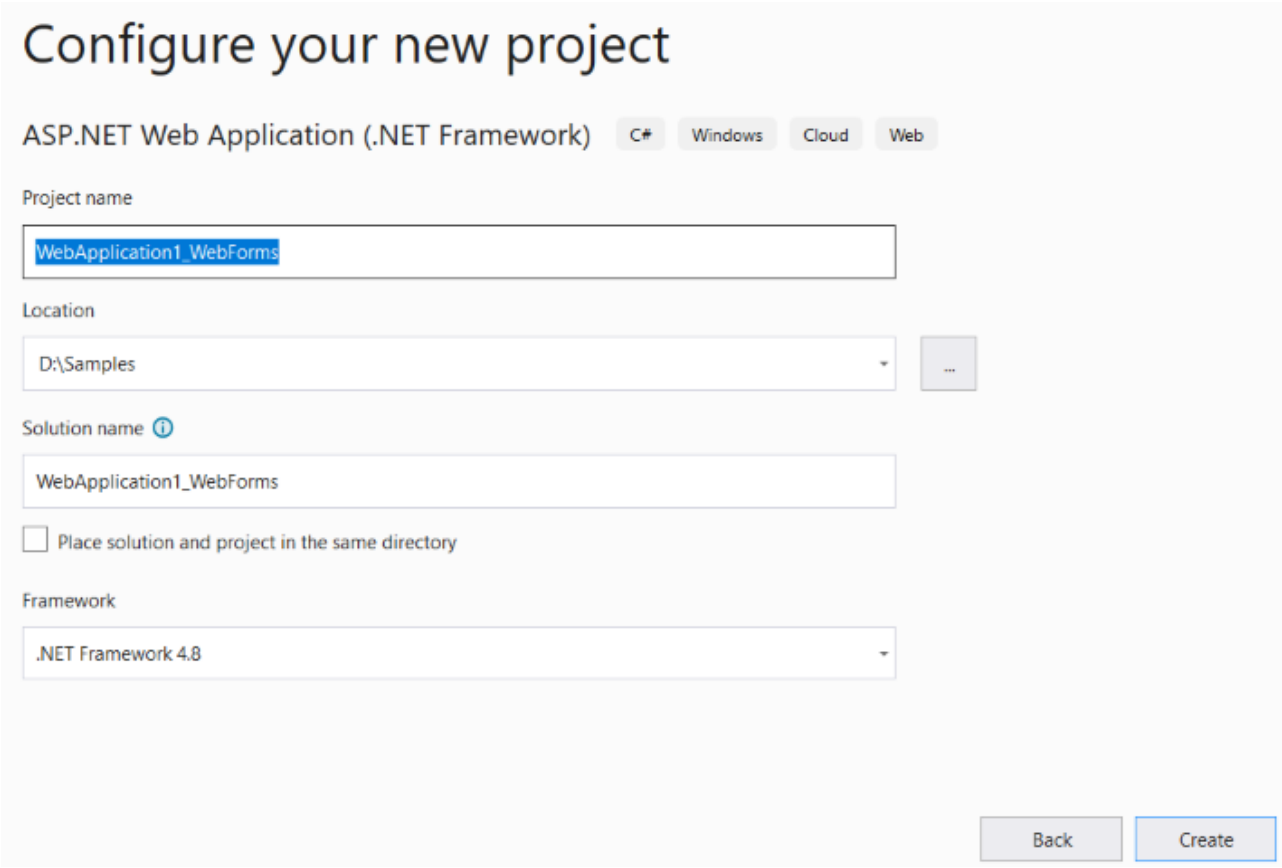
WebForms Application

The steps listed below describe how to configure DsPdfViewer 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)**.

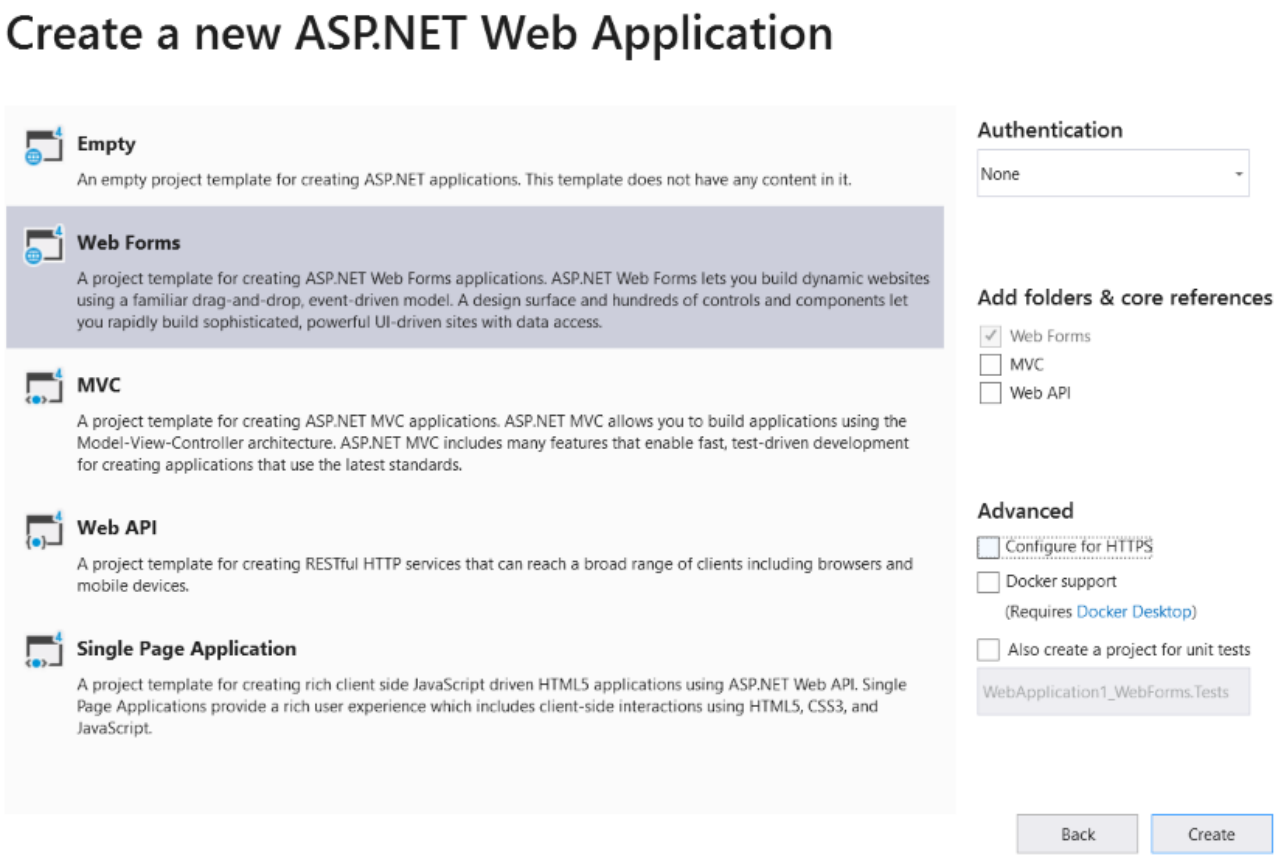


2. In **'Configure your new project'** dialog, name the project and select '**.NET Framework 4.8'** framework.



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.



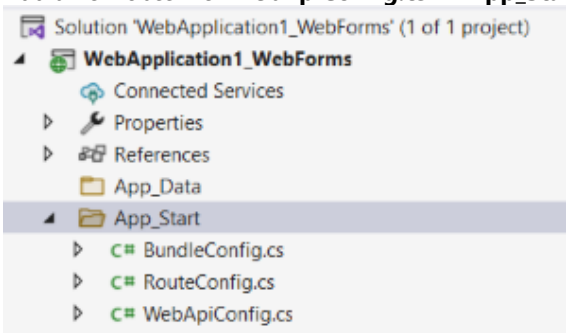
4. Run the following command to install DsPdfViewer. Make sure that the directory location in command prompt is set to Scripts folder. The DsPdfViewer will be installed in WebApplication1_WebForms\WebApplication1_WebForms\Scripts:
   `npm install @mescius/dspdfviewer`
5. Replace the code in <asp:Content>..</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/@mescius/dspdfviewer/dspdfviewer.js"></script>
    <script>
        var viewer = new DsPdfViewer("#host", { supportApi: 'SupportApi' });
        viewer.addDefaultPanels();
        viewer.addAnnotationEditorPanel();
        viewer.addFormEditorPanel();
        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:
   - *DS.Documents.Pdf.ViewerSupportApi*
   - *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:

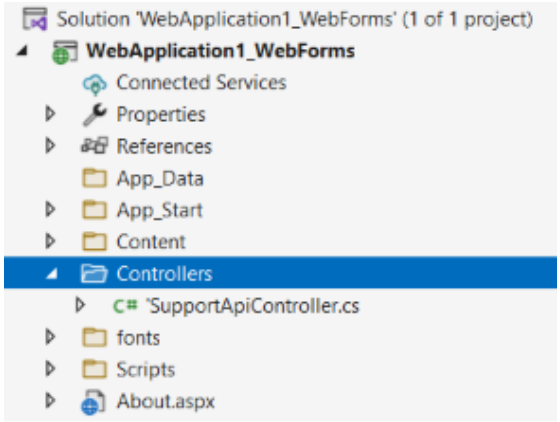
```
using System.Web.Http;

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:

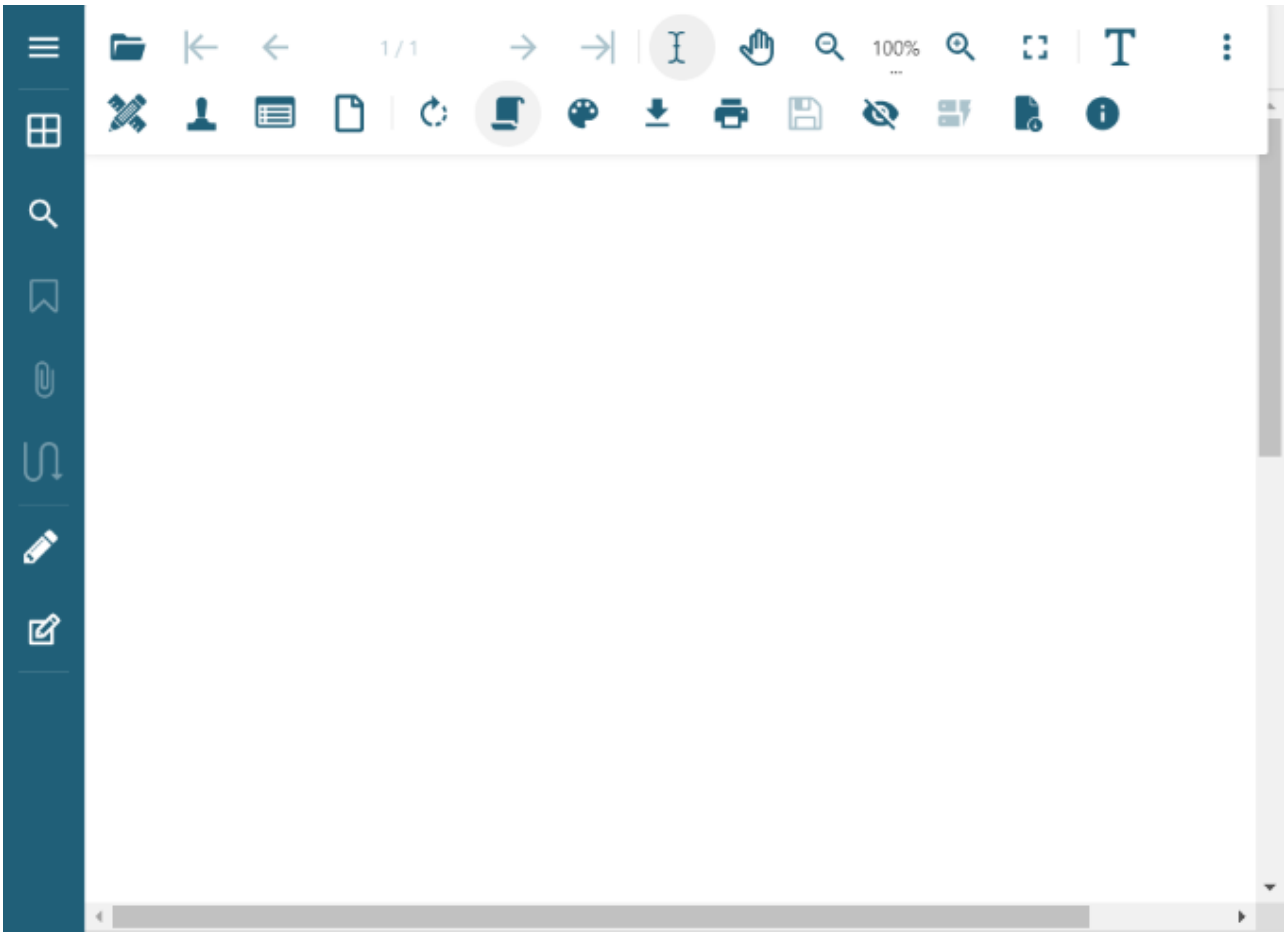


10. Replace the code in 'SupportApiController.cs' with below code:

SupportApiController.cs

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

namespace WebApplication1_WebForms.Controllers
{
    public class SupportApiController : GcPdfViewerController
    {

    }
}
```

11. Build and run the application to view DsPdfViewer 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 DsPDF distribution file. It contains a DsPdfViewerWeb folder which further contains:

- SupportApi - An ASP.NET Core web library which implements the DsPdfViewer'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 DS.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 DsPdfViewer demos.

Editors

Document Solutions PDF Viewer provides the below editors which can be used to edit PDF documents:

- Annotation Editor
- Form Editor

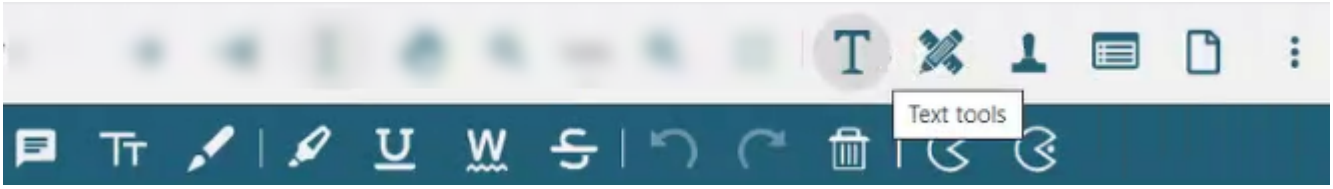
Annotation Editor

The Annotation editor in DsPdfViewer 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 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.

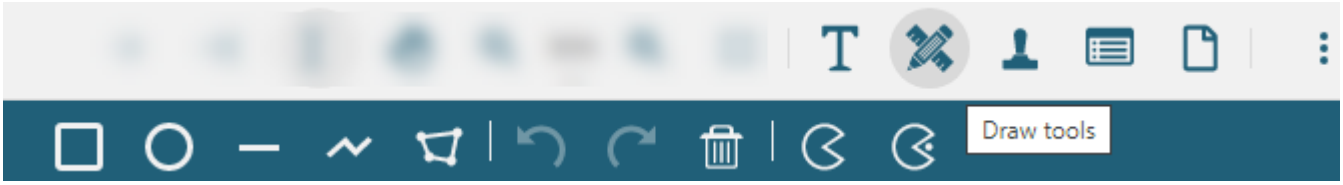
Text Tools

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.



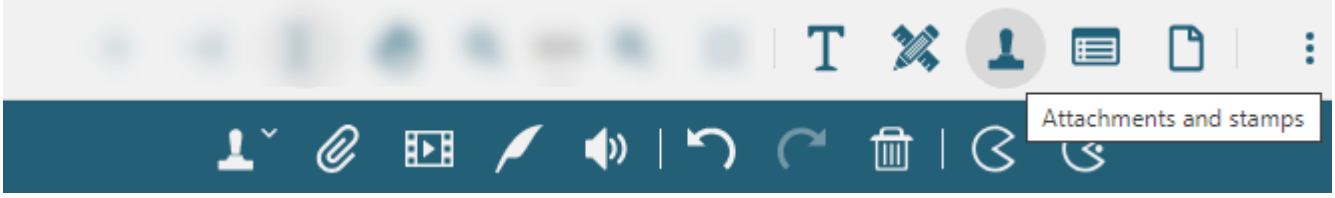
Draw Tools

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 the current page, performing undo or redo operations, and reordering or reorganizing the pages. For more information about PDF Organizer, see PDF Organizer.

The Form tools button  in the above toolbar provides various form field buttons. To know more, refer Form Editor.

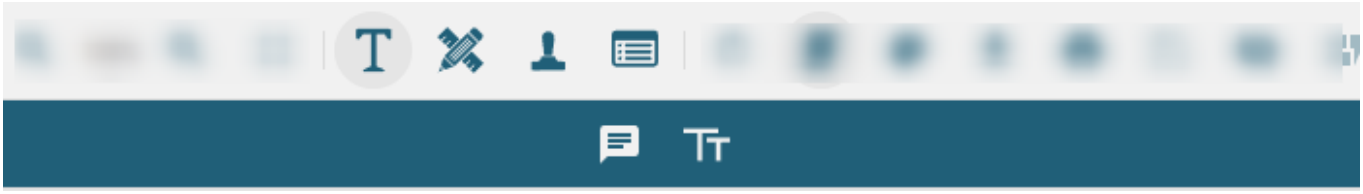
> **Note:** The editing tools (explained above) are automatically enabled in the main toolbar when SupportApi is configured in the project (which allows editing operations in a PDF document).

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:

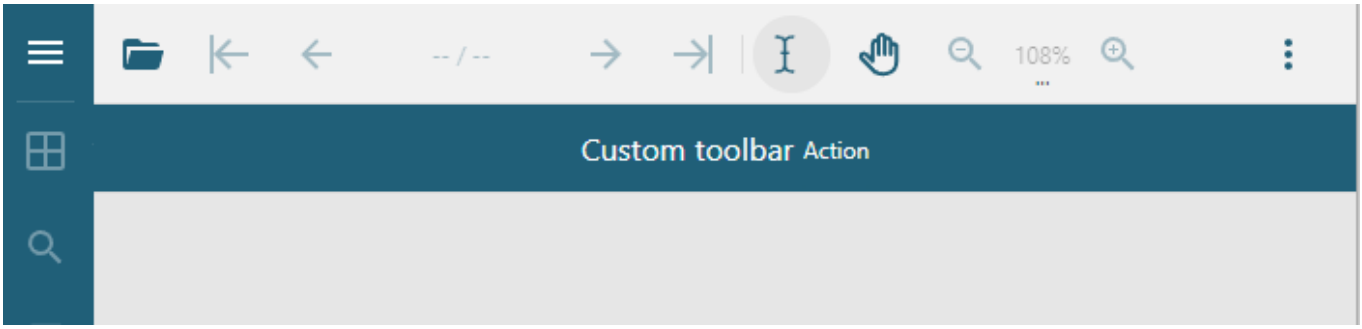


You can also customize the second toolbar by using the **render** handler for **secondToolbar** option:

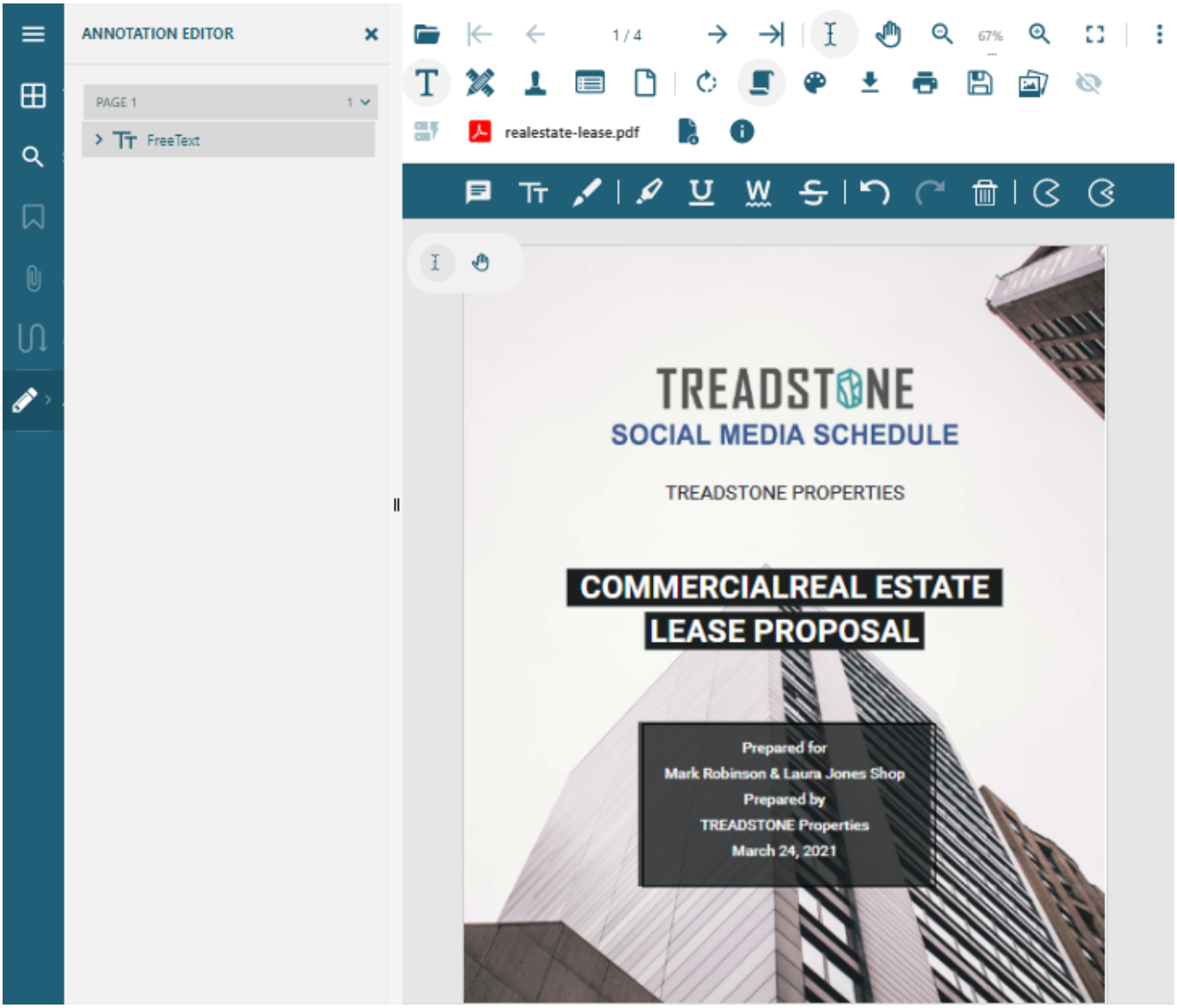
Index.cshtml

```
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:

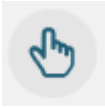


Alternatively, you can access all the available annotations through the Annotation editor's toolbar which opens on clicking the Annotation editor button in the side panel. The below image shows Annotation editor and its toolbar in DsPdfViewer with a PDF document containing a Text annotation.



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.
RichMedia Annotation		Adds media resources to 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.
Highlight Annotation		Adds highlight over selected text of PDF.
Underline Annotation		Adds underline over selected text of PDF.
Squiggly Annotation		Adds squiggly annotation over selected text of PDF.

Strike-out Annotation		Strikes-out selected text of 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 DsPdfViewer. The below annotations are explained in detail in following topics:

- Redact Annotation
- Stamp Annotation
- Link Annotation
- RichMedia Annotation

Apart from the different types of annotations described above, DsPdfViewer 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
	Saves the modified document as images on client
	Creates a new blank document
	Inserts a blank page
	Deletes current page
	Reorders and reorganizes the pages in a PDF document. For more information, see PDF Organizer.

Note: There is a difference in how the Highlight annotation looks in DsPdfViewer and in Adobe Acrobat reader.

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 });
    });
```

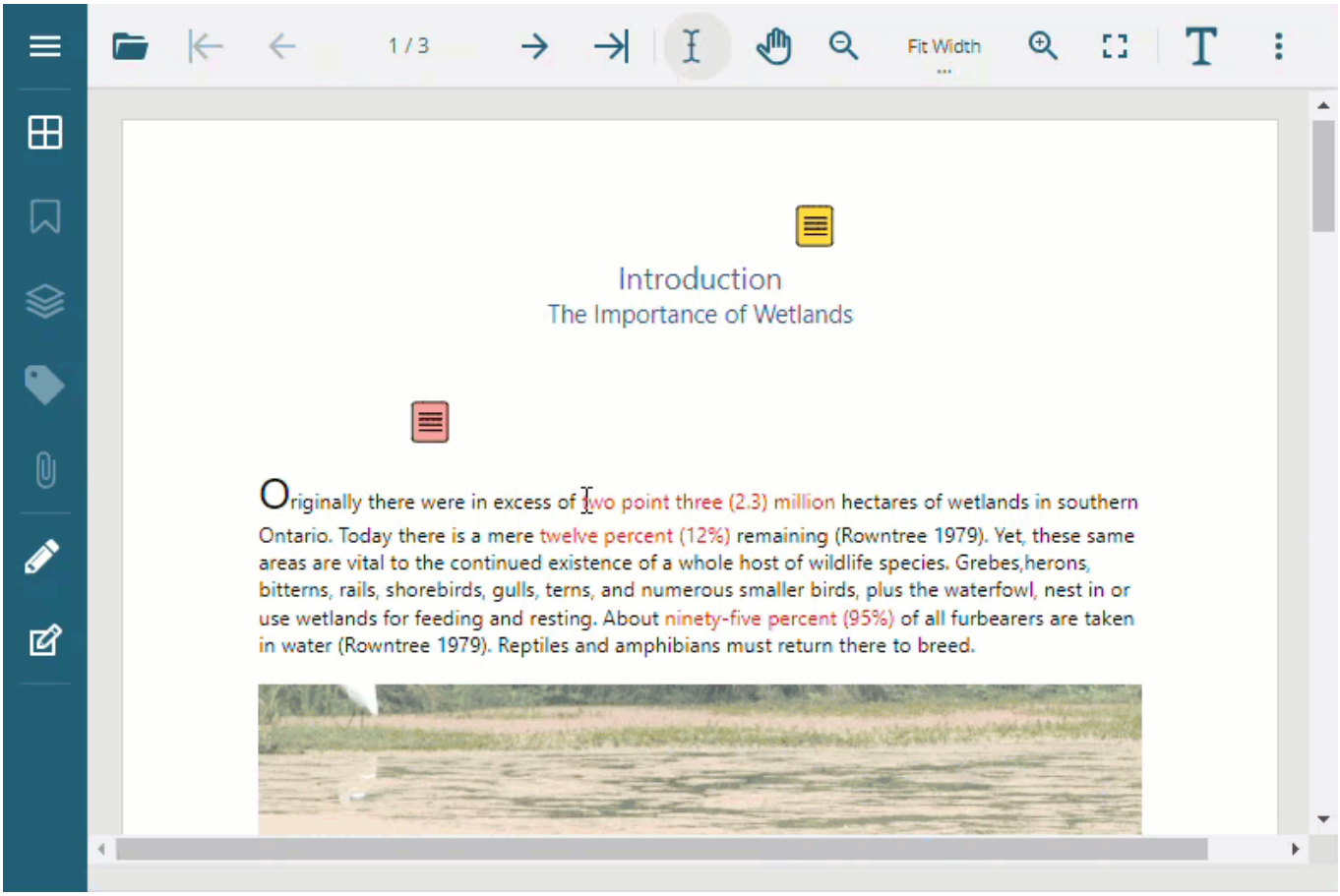
Mark Text Menu

The **Mark Text Menu** appears as a sub-menu when clicking on the Mark Text option available in the viewer's context menu. This menu allows you to distinguish selected text by applying the following annotations in six different colors by default:

Highlight Annotation		Adds highlight over selected text of PDF
Underline Annotation		Adds underline over selected text of PDF
Squiggly Annotation		Adds squiggly annotation over selected text of PDF
Strike-out Annotation		Strikes-out selected text of PDF

When applying an annotation to the selected text using this menu, a comment corresponding to the annotation gets added to the Comment Panel, which activates the Comment Panel, focusing on the new comment in the list. This comment can then be used to provide additional information about the marked text. For more information, refer to Comment Panel.

Refer to the following GIF image that depicts the described behavior:



You can also access these markup annotations through Text tool options on the quick editing toolbar of the PDF viewer as well as from Annotation Editor.

DsPdfViewer lets you disable the Mark Text Menu by setting the **textMarkupContextMenu** property to false. You can also change colors available for these annotations.

Refer to the following example code to disable the Mark Text Menu:

C#

```
// Disable the Mark Text Menu.
var viewer = new DsPdfViewer(selector, {textMarkupContextMenu: false});
```

Refer to the following example code to change the default color options available to red and black:

C#

```
// Change the default text markup colors to “Red” and “Black”.
var viewer = new DsPdfViewer(selector, {
textMarkupContextMenu: { colors: [{value: "#ff0000", displayName: "Red"}, {value:
"#000000", displayName: "Black"}] } });
```

Refer to the following example code to change the default color options available for different text markup types in the Mark Text Menu:

C#

```
// Customize available colors for different text markup types in Mark Text Menu.
var viewer = new DsPdfViewer(selector, {
    textMarkupContextMenu: {
      colors: {
          highlight: [
              {value: "#ff0000", displayName: "Red"},
              {value: "#000000", displayName: "Black"}
          ],
          underline: [
              {value: "#ff0000", displayName: "Red"}
          ]
      }
    }
  });
```

Limitation: Appearance of Highlight annotation in DsPdfViewer is slightly different from that in Adobe Acrobat Reader.

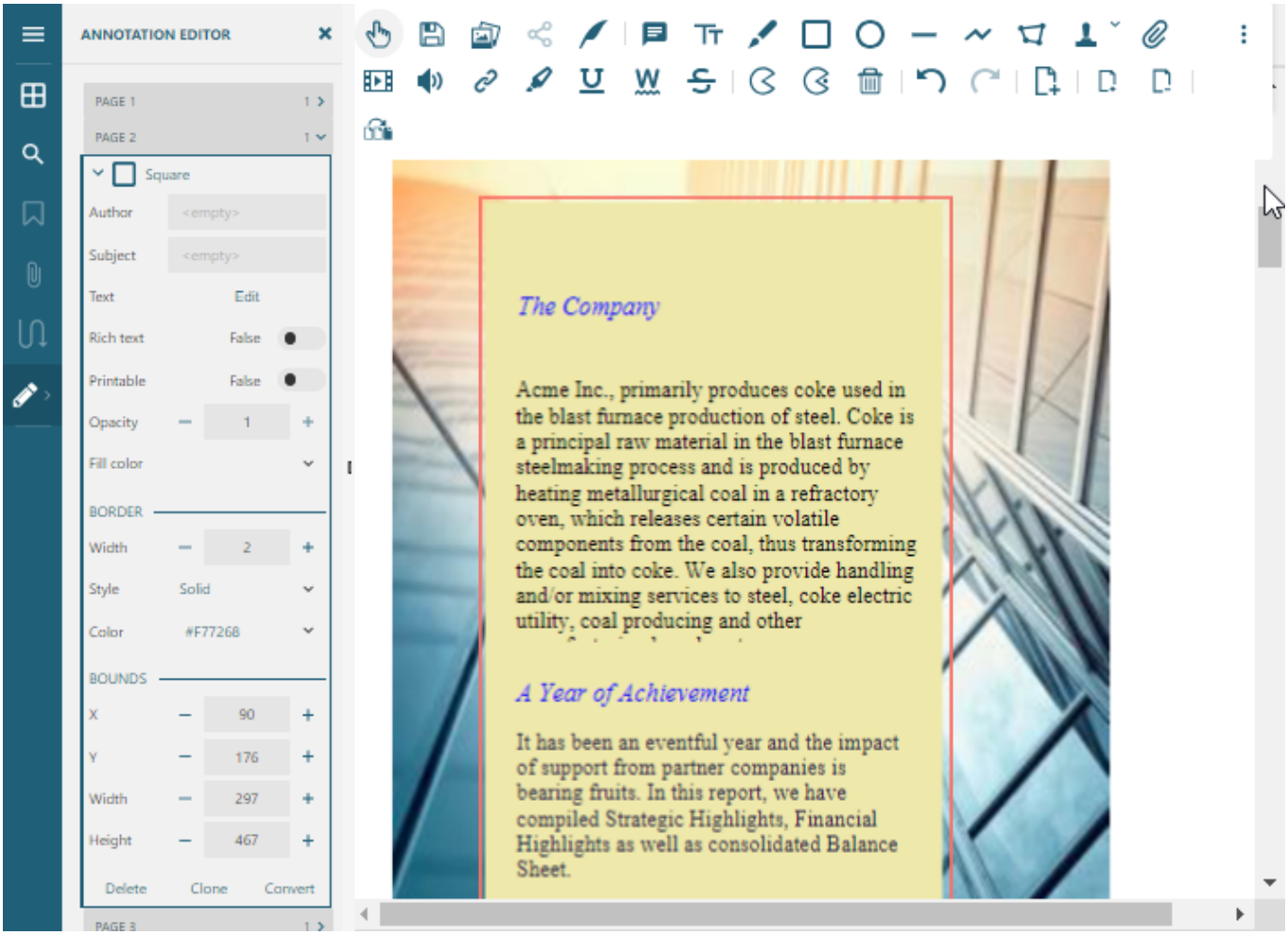
Property Panel of Annotation Editor

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.

The Property panel also provides the following three options, which are commonly available for all annotations:

- Delete: Deletes the annotation.
- Clone: Creates and adds a clone of the annotation to the document.
- Convert: Converts the annotation to PDF content, and a revert button starts appearing in the panel to revert it back to annotation. For more information, see Convert to Content.

The image below shows the properties of a square annotation in the Property panel:



Enable Annotation Editor in DsPdfViewer

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

Index.cshtml

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

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

C#

```
public static DsPdfViewerSupportApiDemo.Models.PdfViewerOptions PdfViewerOptions
{
        get => new DsPdfViewerSupportApiDemo.Models.PdfViewerOptions(

DsPdfViewerSupportApiDemo.Models.PdfViewerOptions.Options.AnnotationEditorPanel |
```

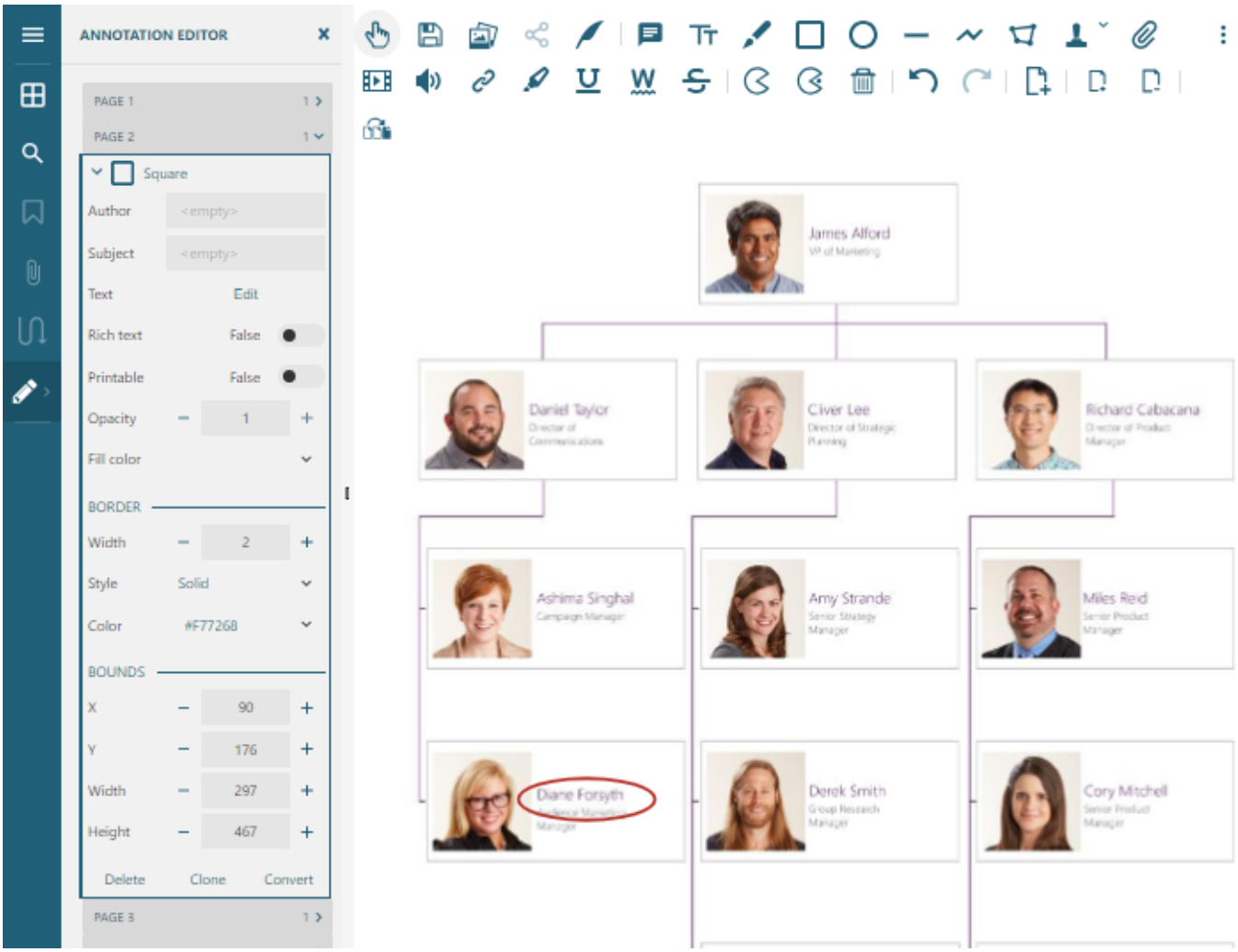
```
DsPdfViewerSupportApiDemo.Models.PdfViewerOptions.Options.ActivateAnnotationEditor,
        annotationEditorTools: new string[] { "edit-select", "$split", "edit-text",
"edit-free-text", "$split", "edit-erase", "$split", "edit-redact", "edit-redact-
apply", "$split", "edit-undo", "edit-redo", "save" });
}
```

Review PDF document using Annotation editor

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

1. Configure the DsPdfViewer 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.



For more information, refer Annotation Editor in DsPdfViewer demos.

Redact Annotation

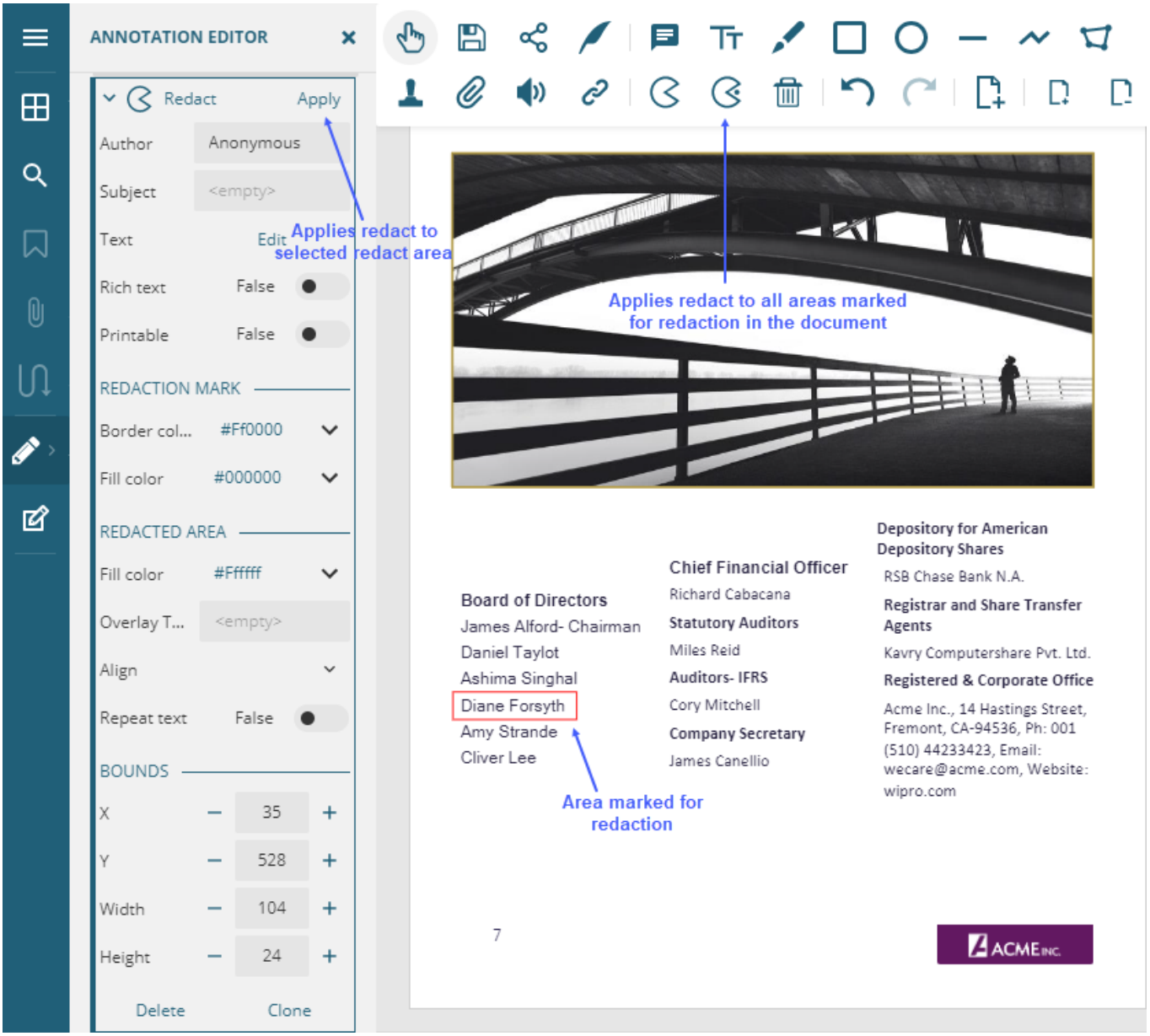
The annotation editor allows you to add Redact annotation. A redact annotation marks an area under which all the existing content should be removed from PDF document. Redact annotation, unlike other annotations, is applied in two phases:

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.

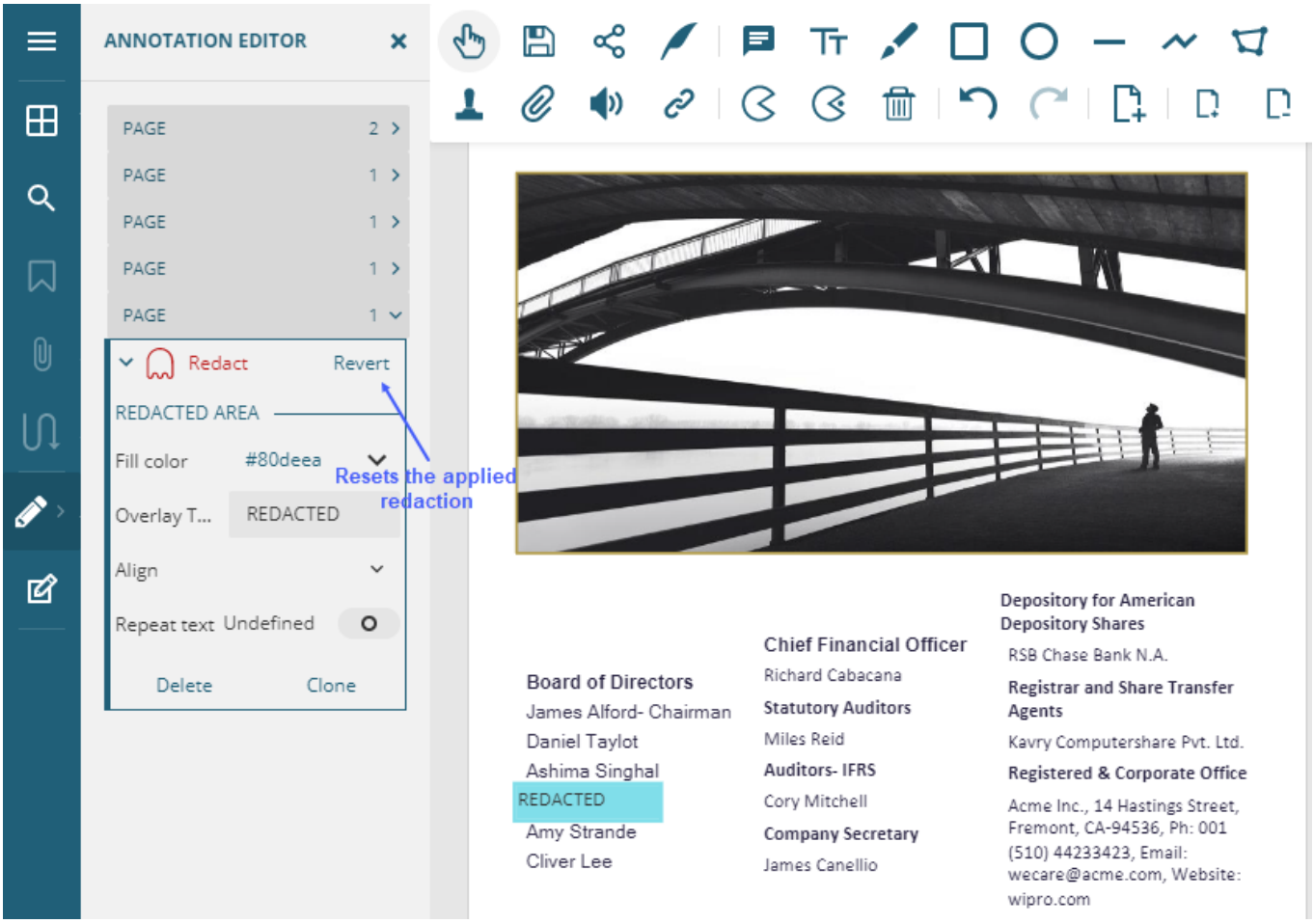
> **Note:** The Redact annotation options can also be accessed through quick editing toolbar. For more information, refer Annotation Editor.

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.

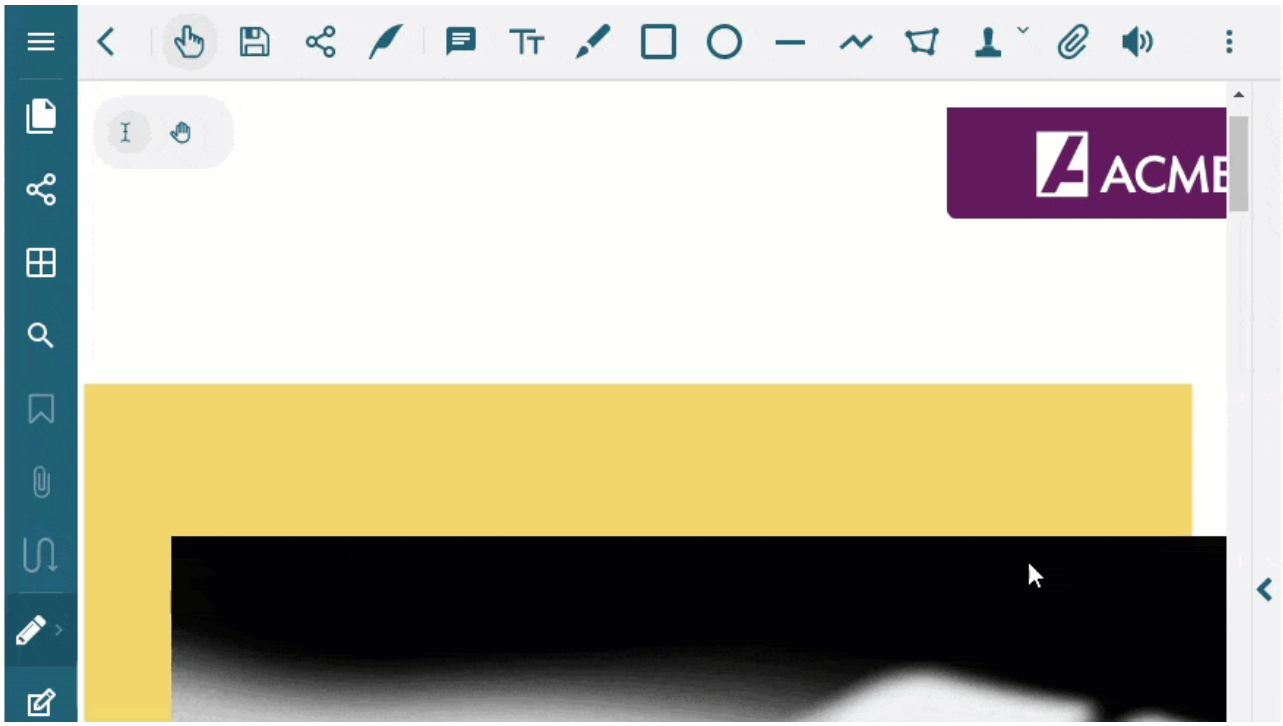
Stamp Annotation

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:

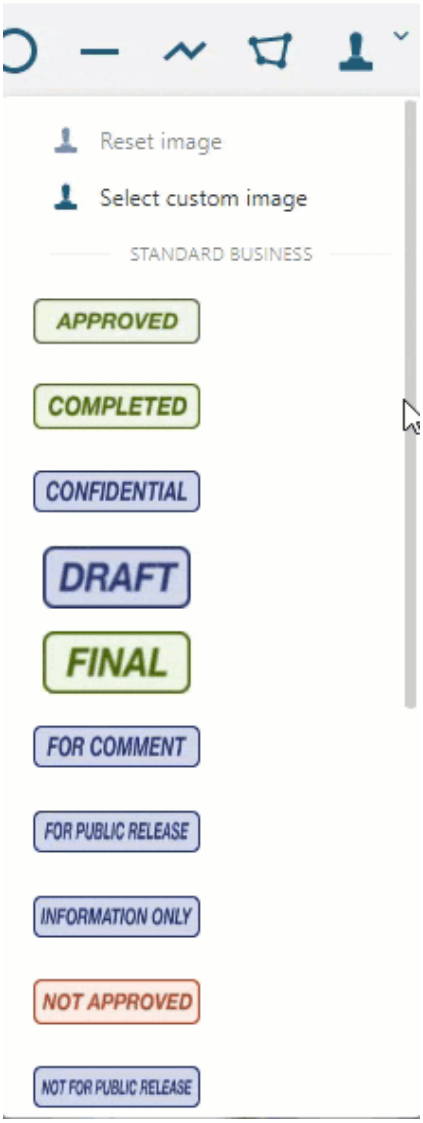
> **Note:** The Stamp annotation option can also be accessed through quick editing toolbar. For more information, refer Annotation Editor.

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 DsPdfViewer:

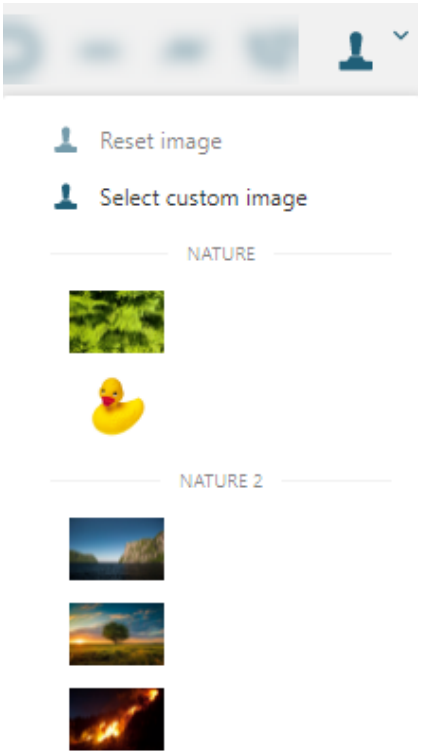


You can also specify your own set of predefined stamps on the client side by using the **stampCategories** option as shown below:

Index.cshtml

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

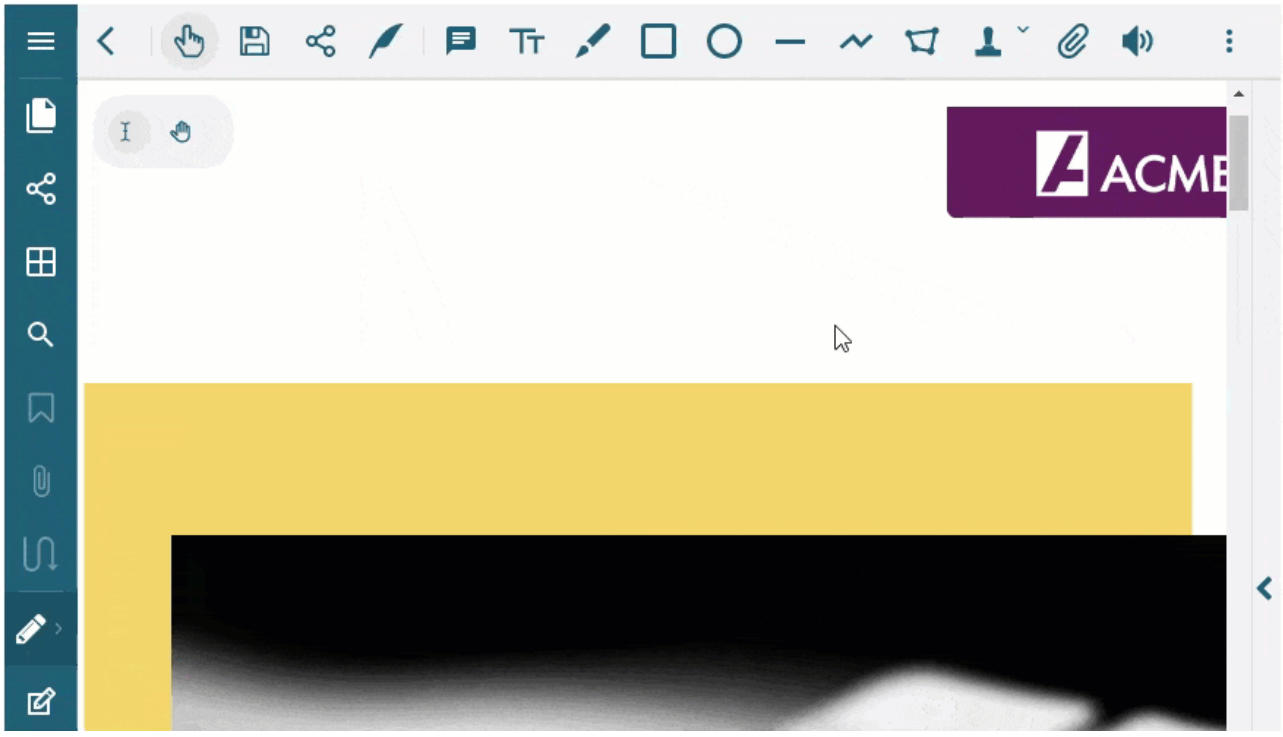
The output of the above code will look like below where the specified stamps will be available in the Stamp annotation's dropdown button:



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:

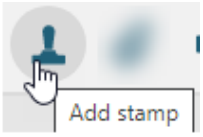


In case you want to add only custom images in a PDF document and no predefined stamps, you can remove the dropdown for adding predefined stamps (visible by default).

Index.cshtml

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

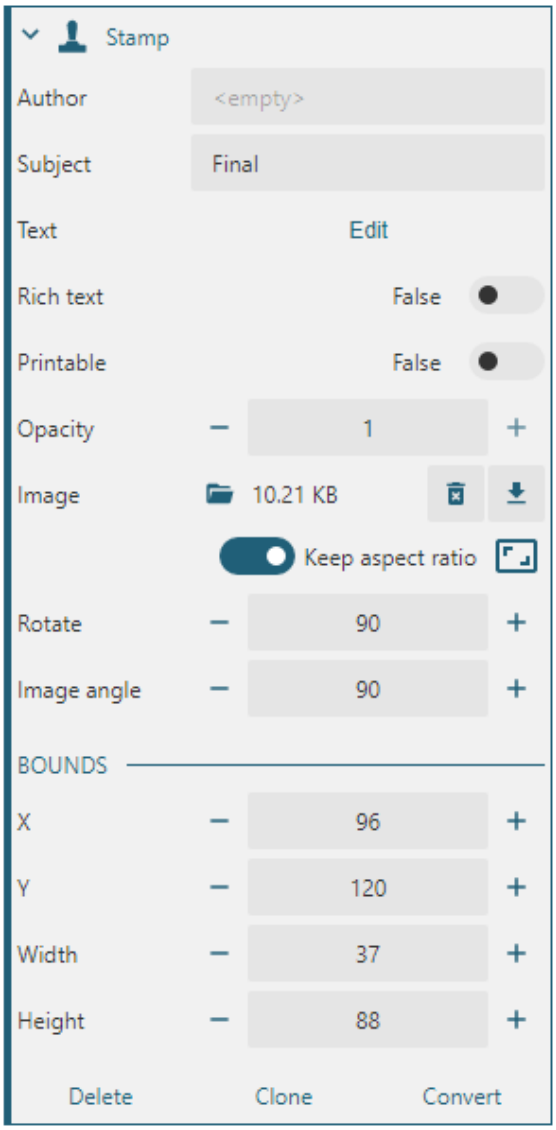
The above code will remove the dropdown of Stamp annotation button:



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.



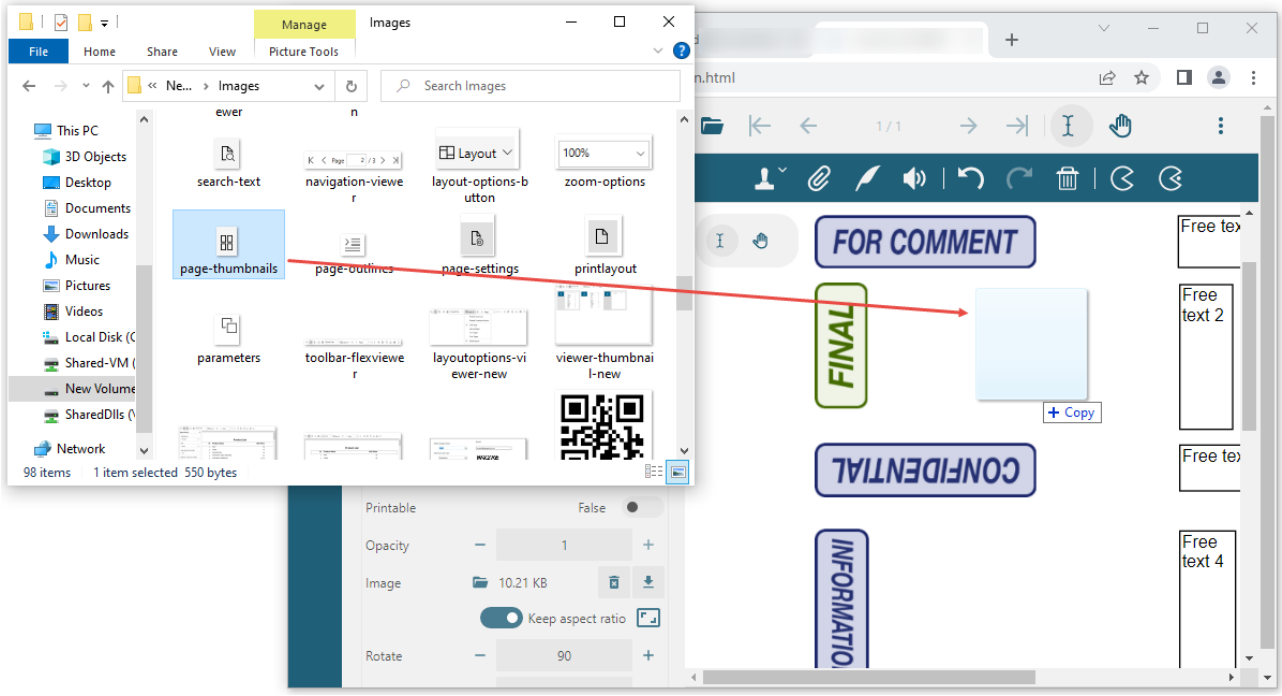
Limitation

Stamp annotation cannot be resized after rotation.

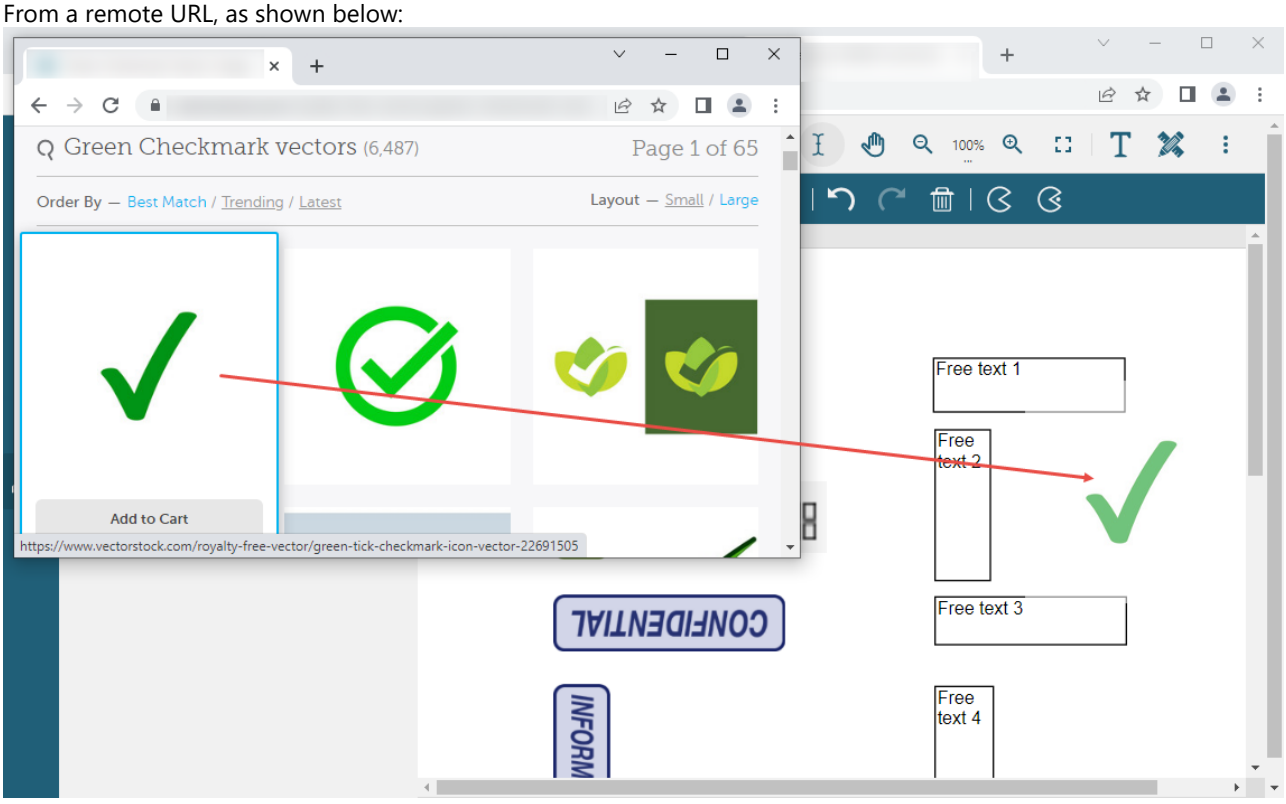
Add Custom Images using Drag and Drop Operation

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

1. From local system, as shown below:



2. From a remote URL, as shown below:

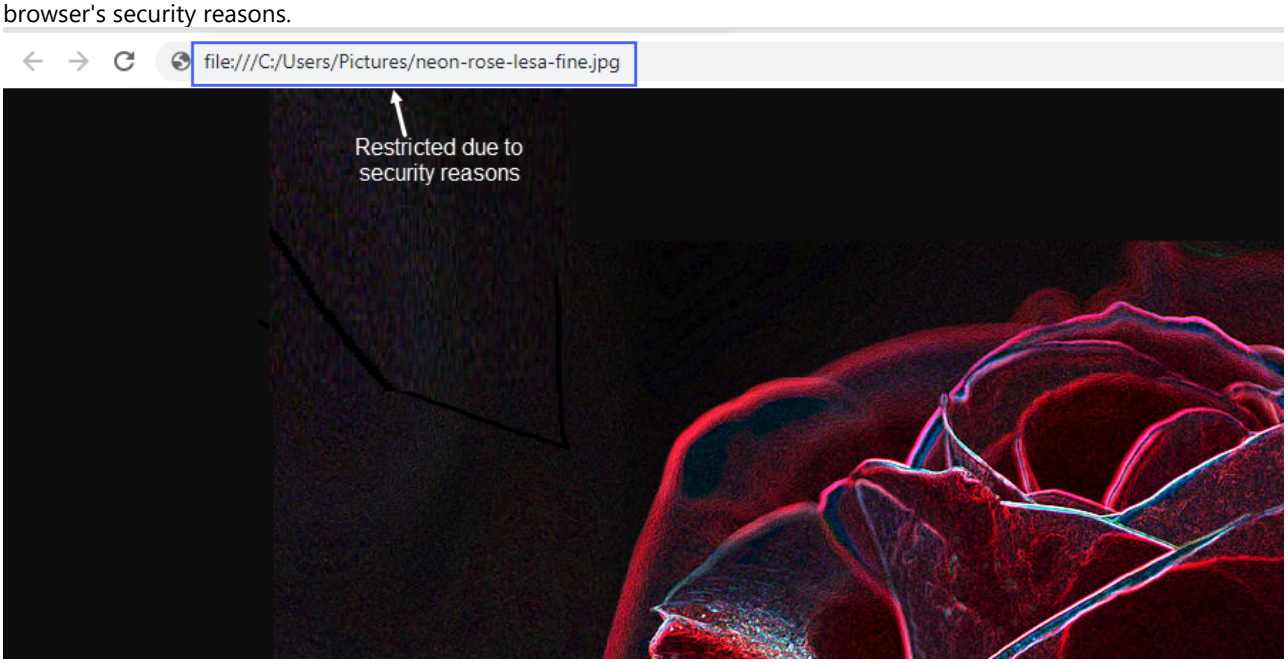


> **Note:** While performing drag and drop operation, the supported image format depends on the browser type. However, all common image formats are supported, to learn more about image formats supported by different browsers, please check here.

For more information, refer Stamp Annotations in DsPdfViewer demos.

Limitation

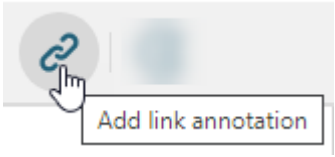
1. A local image opened in browser by using local file system URL is not supported for drag and drop operation due to browser's security reasons.



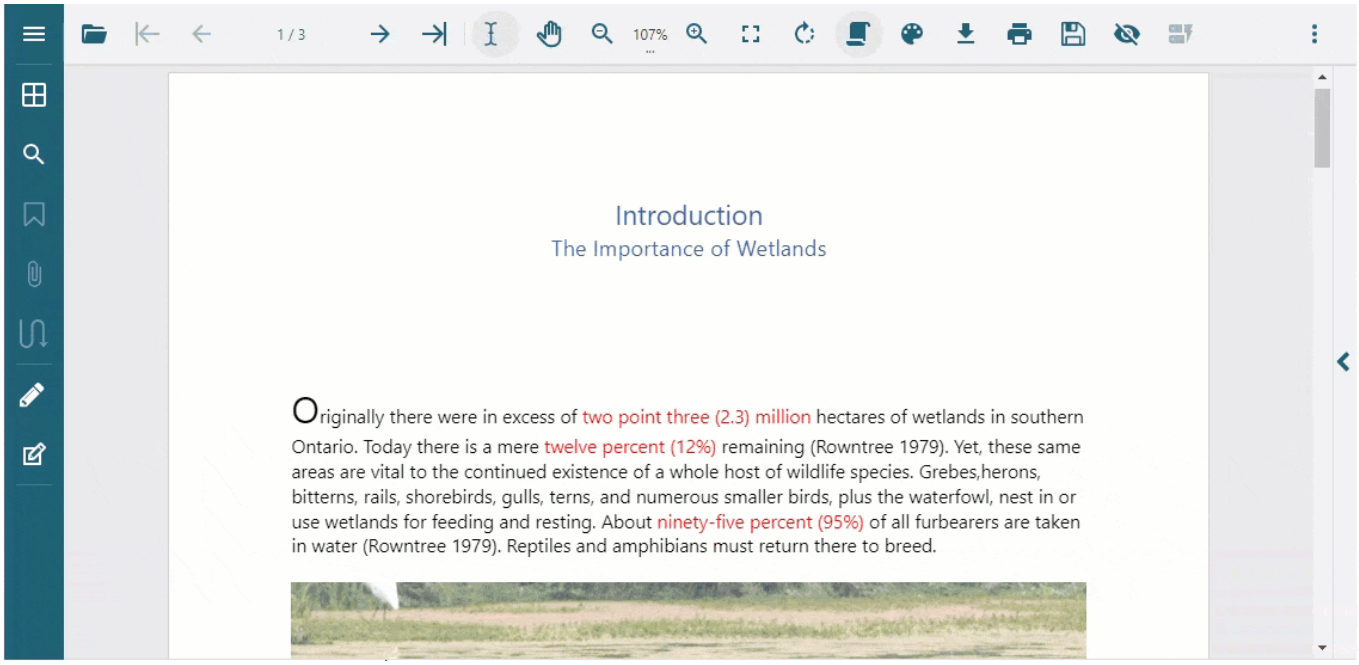
2. Drag and drop operation from a remote URL can be blocked by the host server's cross-domain policy.

Link Annotation

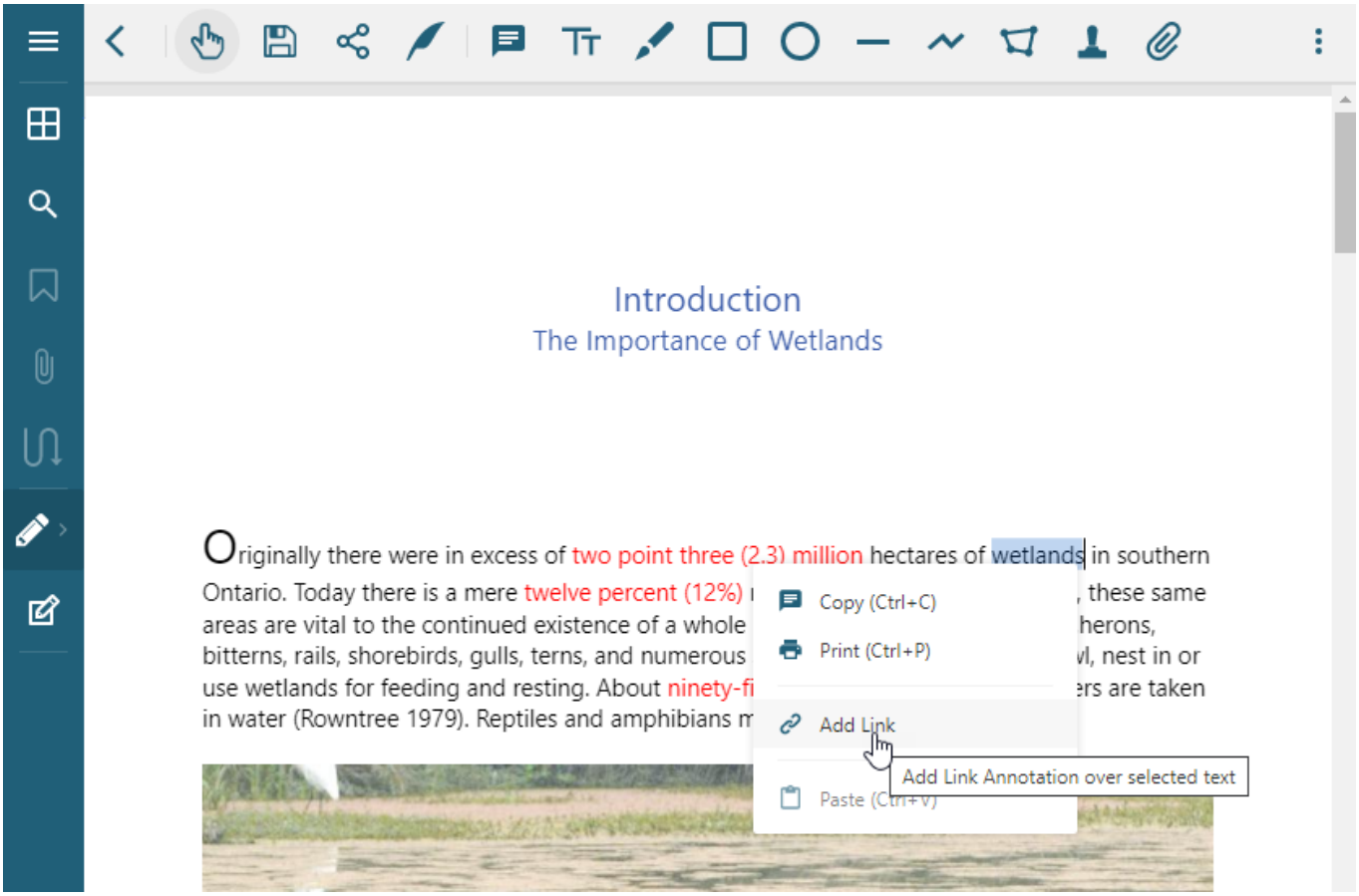
The Annotation Editor provides Link annotation in its toolbar which allows you to add links in a PDF document. The links can be added to any area of a PDF document like text, image or over any other existing annotation or form fields. The below image shows the Link annotation button:



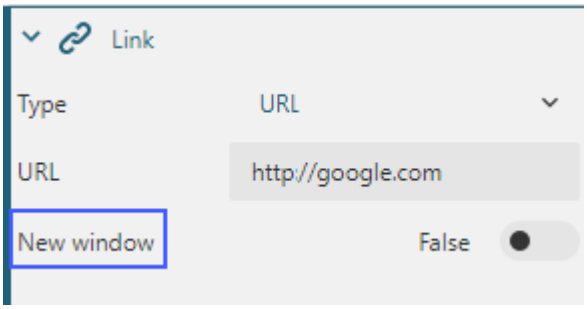
The below GIF demonstrates how to add a link in a PDF document by using Link Annotation. When you hover over the linked area, the link is displayed as a tooltip:



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



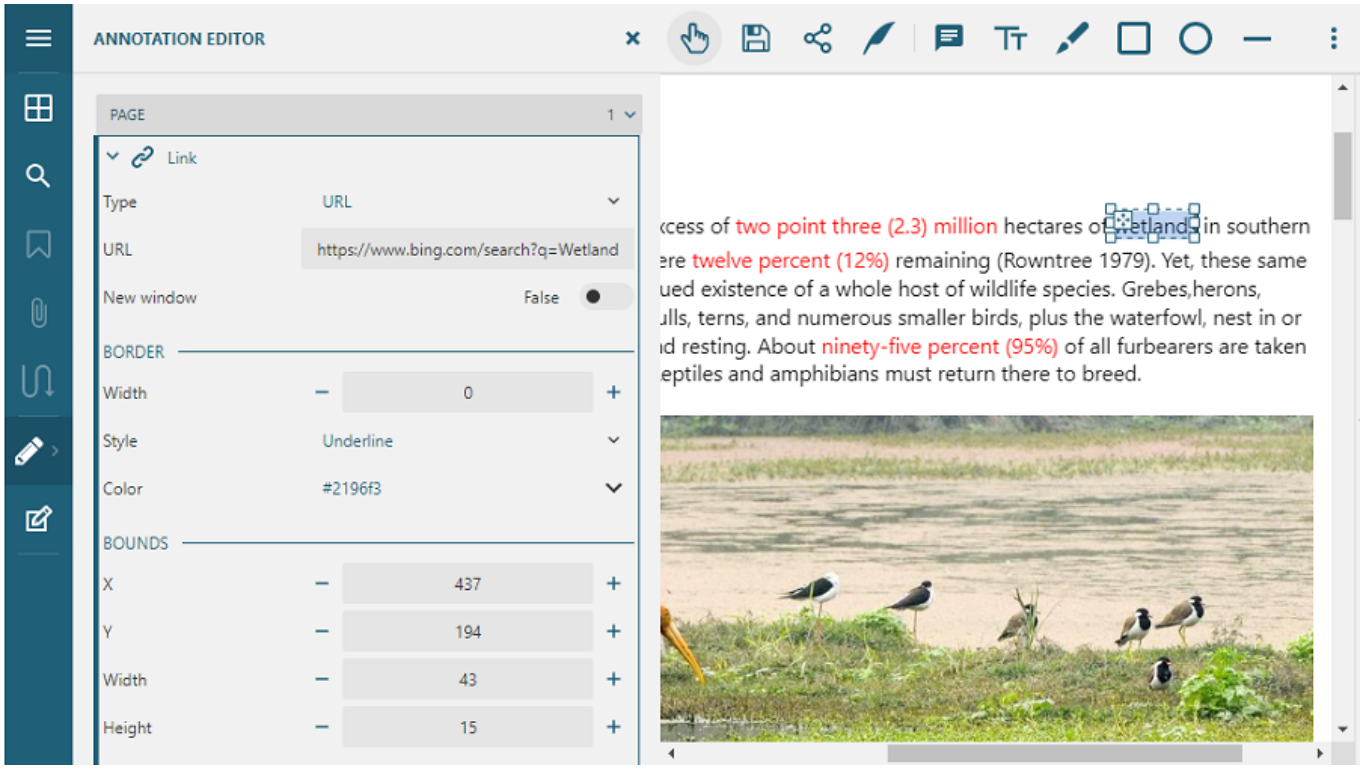
By default, all external links are opened in the same browser window unless the 'New window' option in the property panel is set to true.



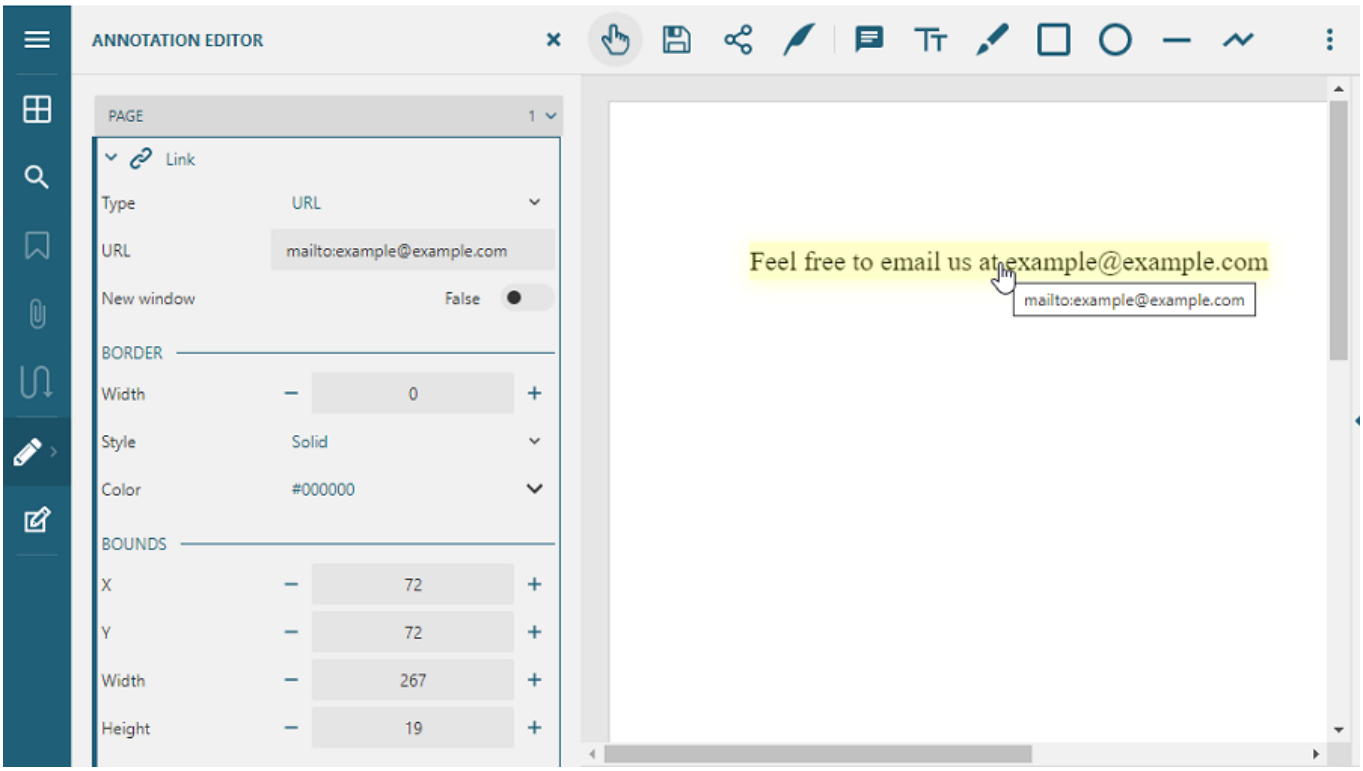
Types of Link Annotations

DsPdfViewer 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':

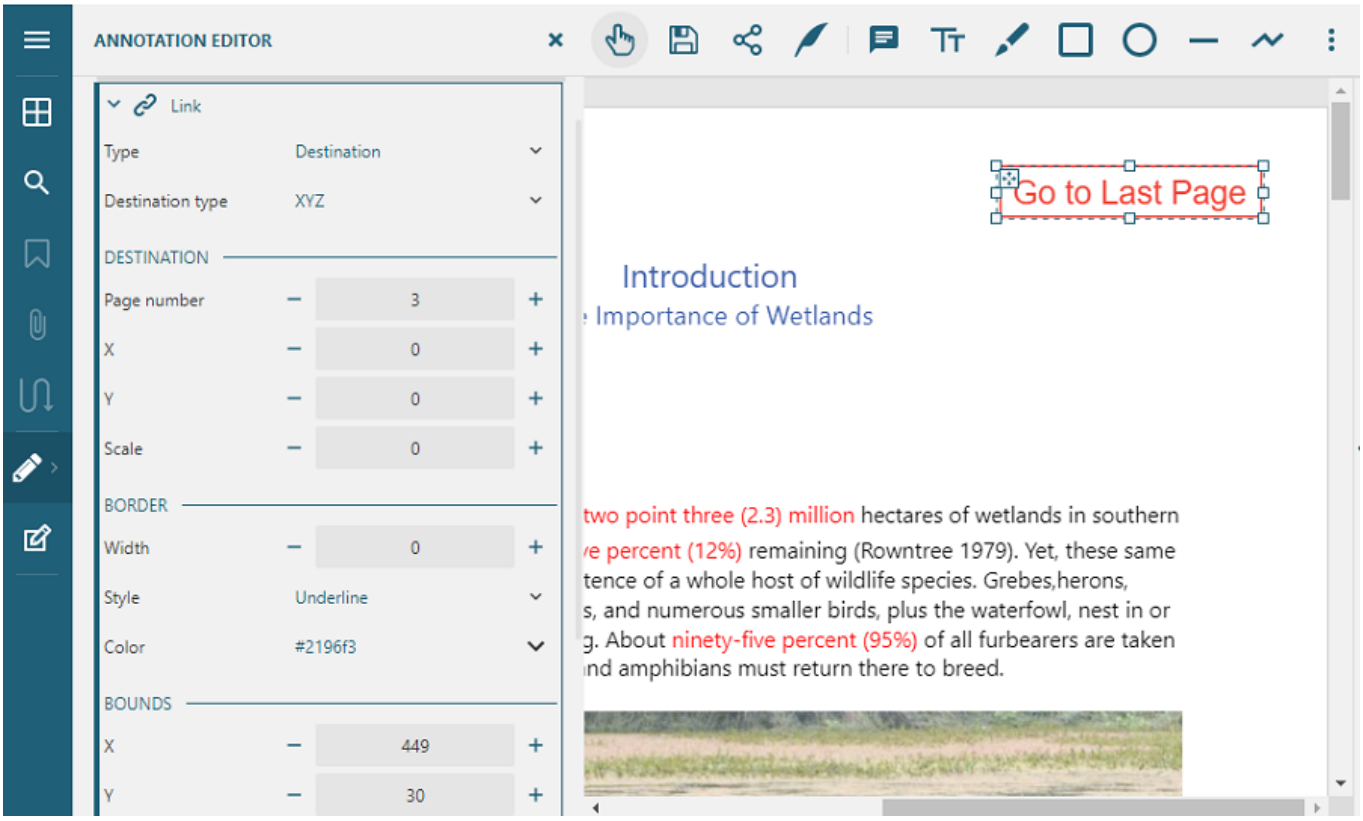


The below image shows adding a URL as a link to open email:

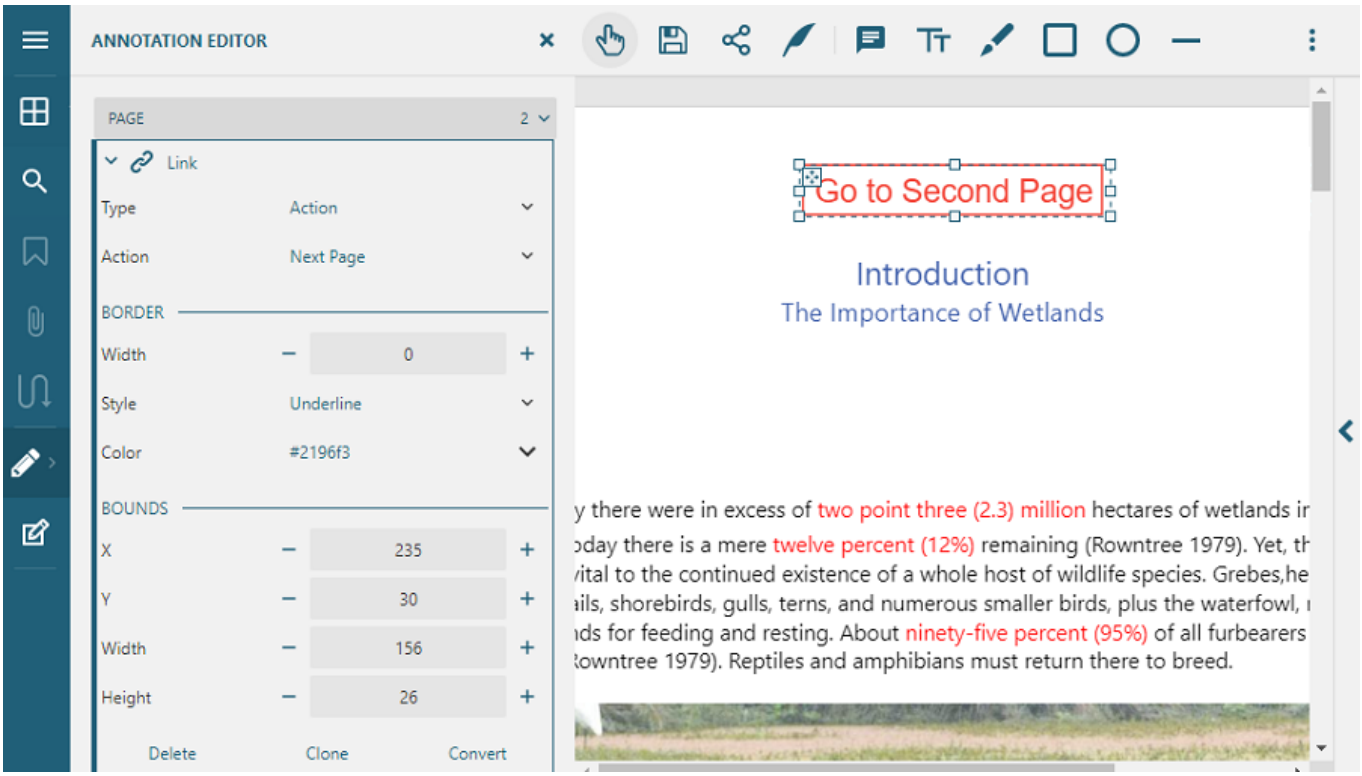


Destination: Specifies the explicit destination in a PDF document to be linked. It can point to any area or page of the PDF document with various options like fit to page, magnified content, specific positioning of vertical or horizontal coordinates etc. The 'Destination type' provides all the 8 options 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.

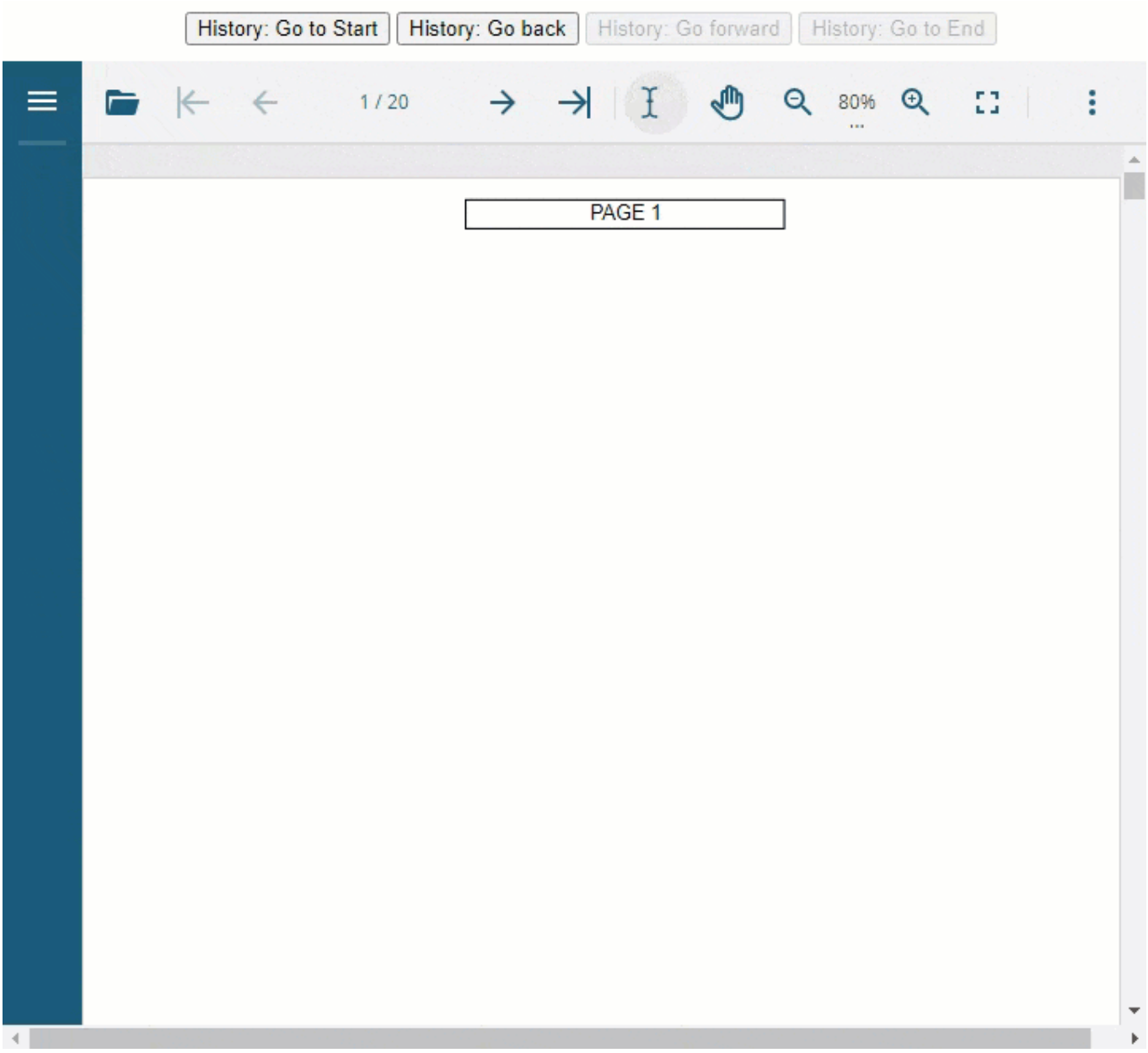


The Go Back and Go Forward actions allow you to navigate the view history of a PDF document similar to a browser's go back and go forward actions. DsPdfViewer provides the below methods to navigate through the document history:

Index.cshtml

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

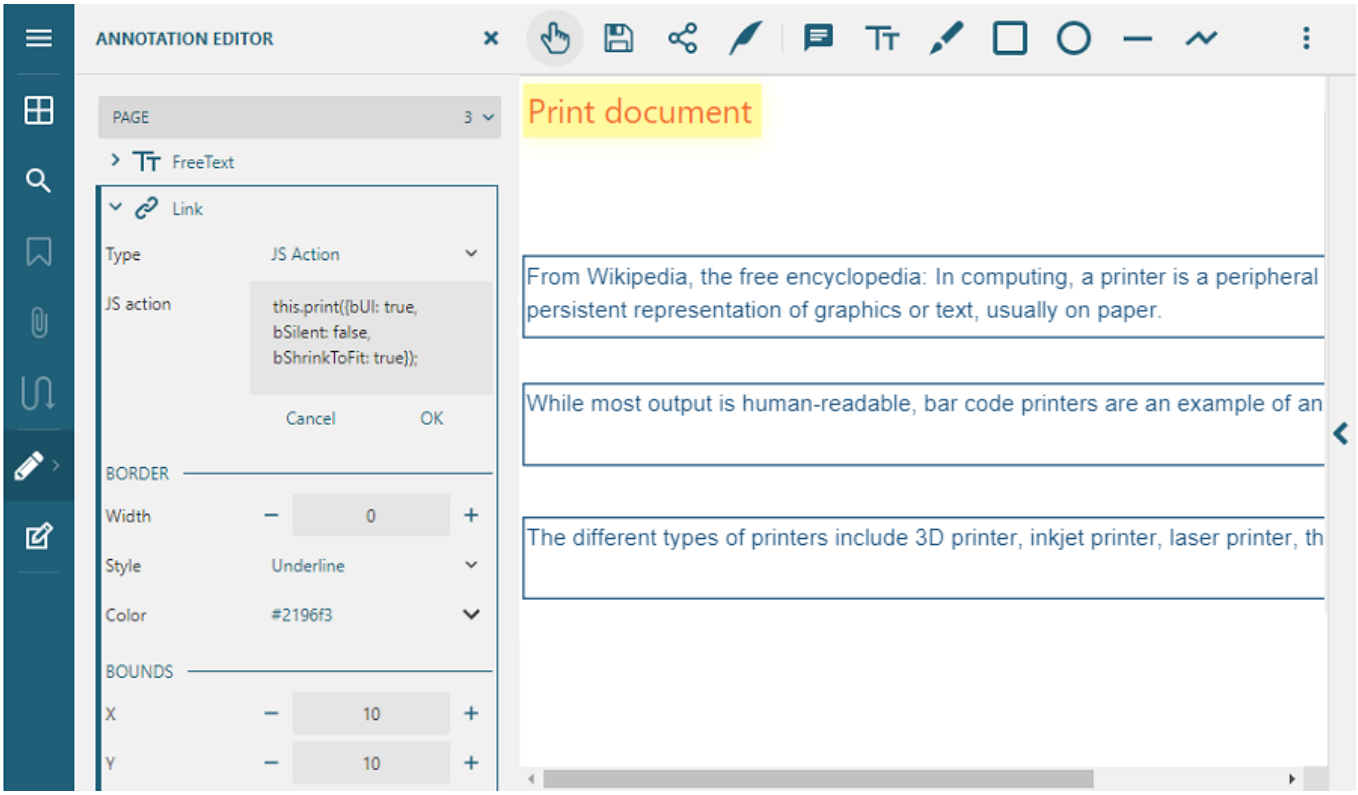
The below GIF shows the functionality of Go to Start, Go Back, Go Forward and Go to End actions using the Link annotation:



DsPdfViewer 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 DsPdfViewer('#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

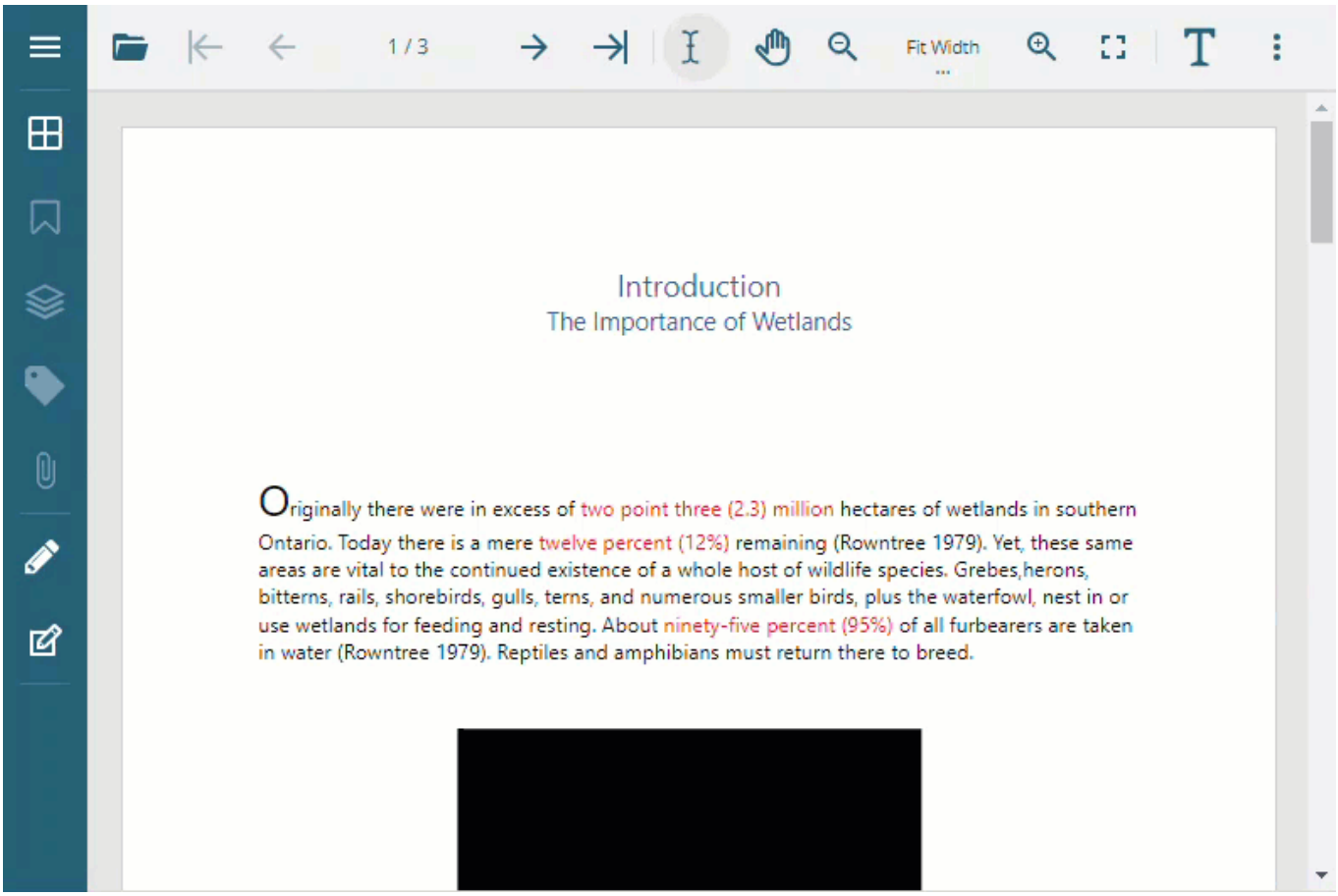
```
var viewer = new DsPdfViewer("#host", {
    externalLinkTarget: 'blank',
    supportApi: 'api/pdf-viewer'
    }
);
```

For more information, refer Link Annotations in DsPdfViewer demos.

RichMedia Annotation

Annotation Editor allows you to add **RichMedia** annotations to a PDF document, allowing you to embed the media resources (audio and video) into a PDF file. You can add the RichMedia annotations using **Add rich media** button  on the toolbar.

Note: You can also access Add rich media button through the quick editing toolbar of **Attachments and stamps** button. For more information, refer to Annotation Editor.

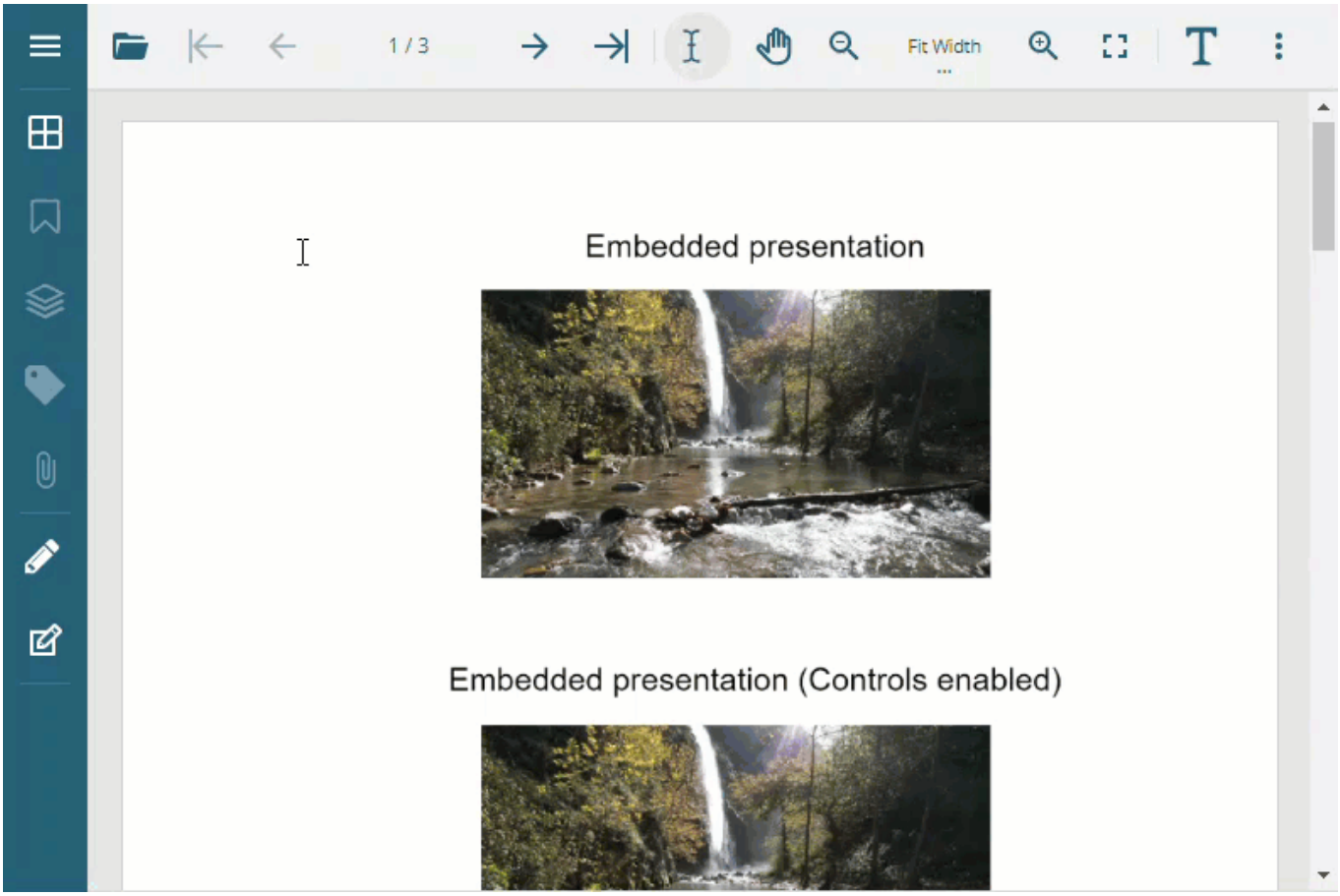


Activation and Deactivation

Playing and pausing the media depends on certain activation or deactivation conditions that you specify when adding the RichMedia annotations. The activation or deactivation of the media resources depends on the following conditions:

Activation or Deactivation Condition Name	Description According to PDF Specification	Working in DsPdfViewer
Content is Clicked	The annotation is explicitly activated or deactivated by a user action.	Occurs when the user clicks on the media.
On Page Open or Close	The annotation is activated as soon as the page that contains the annotation receives the same focus as the current page.	Activation takes place when the page containing the rich media annotation receives the same focus as the current page.

Content is Visible or Invisible	The annotation is activated as soon as any part of the page that contains the annotation becomes visible.	Activation occurs when the media container or any of its parts become visible or invisible in the active view.



Note: DsPdfViewer will prompt you to allow auto-playing of media if you set the activation or deactivation condition to On Page Open or Close or Content is Visible or Invisible.

Supported Formats

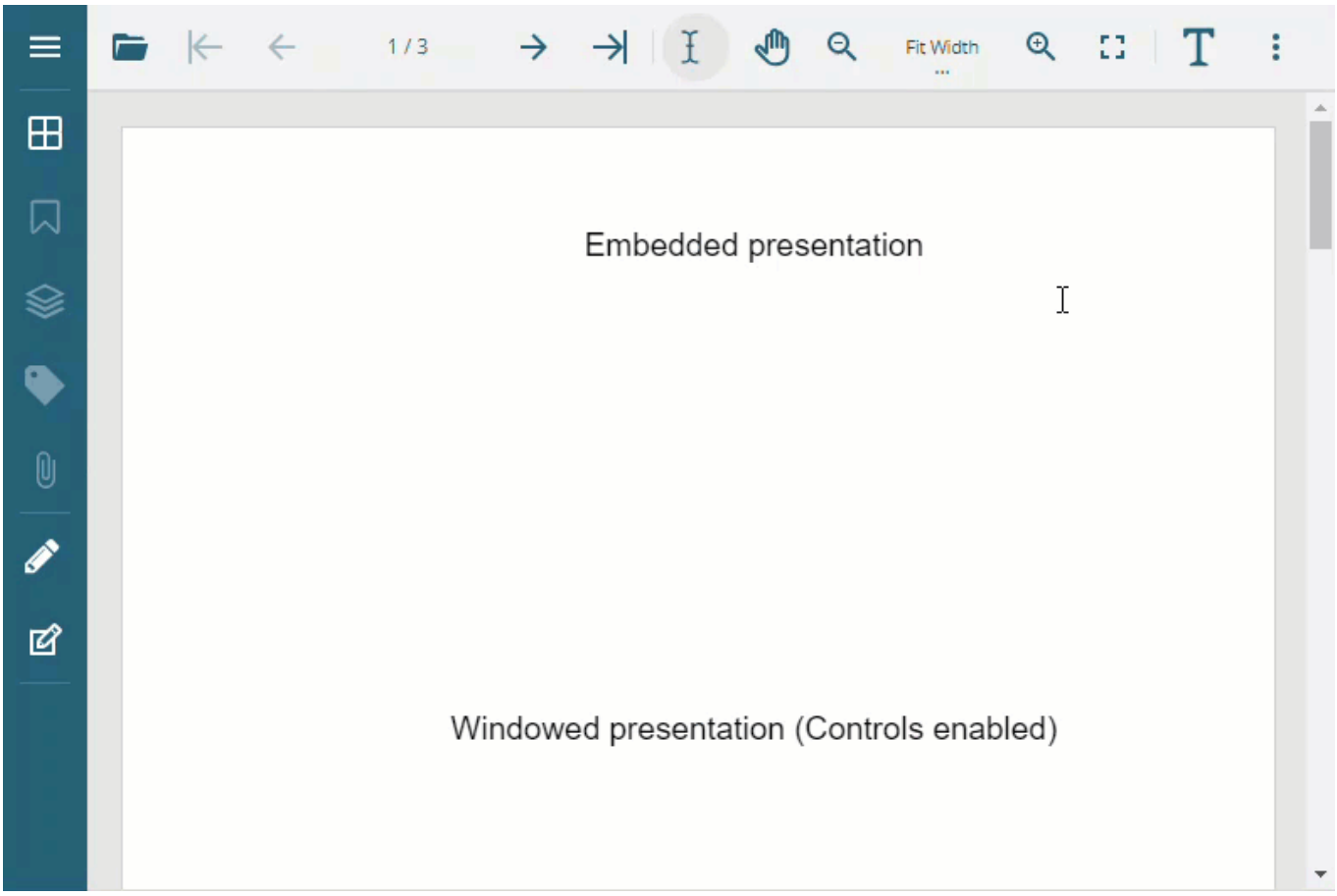
DsPdfViewer supports the following audio and video formats for the media resources:

Audio	Video
MP3	MP4
OGG	SWF
WAV	FLV
-	WebM

Note: To ensure optimal compatibility and widespread support across various devices and platforms, choose MP3 files when dealing with audio and H.264 (AVC) video format when dealing with video. The H.264 (AVC) video format is commonly associated with files having the .mp4 extension.

Add RichMedia Annotation

You can add RichMedia annotations using **Add rich media** button on the Annotation Editor's toolbar or Attachments and stamps button's toolbar. The Browse File dialog will open automatically as soon as the RichMedia annotation is added to the document. Refer to the following GIF image to add rich media annotation in "Embedded" and "Windowed" presentation styles:



Add Rich Media Programmatically

DsPdfViewer also allows you to add RichMedia annotations programmatically using RichMediaAnnotation type. Refer to the following example code to add RichMedia annotation using code:

JavaScript

```
// Fetch bytes from the specified URL.
function fetchBytes(url){
    return new Promise(function(resolve) {
    fetch(url)
        .then(response => response.blob())
        .then(blob => blob.arrayBuffer())
        .then(arrayBuffer => {
            resolve(new Uint8Array(arrayBuffer));
        });
    });
}

// Add RichMedia annotation to the PDF document.
async function addRichMedia(viewer, mediaUrl, posterUrl, args){
```

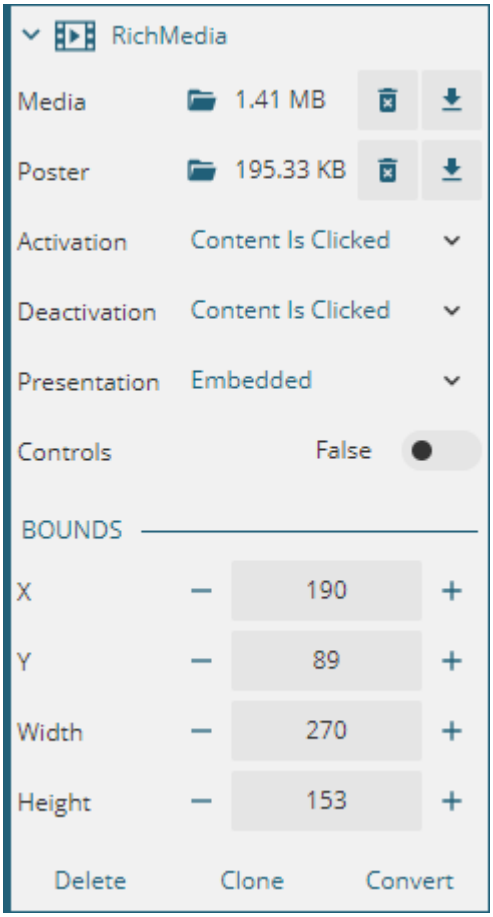
```
    const { width, height, topMargin, mediaFileType } = args;

    const mediaBytes = await fetchBytes(mediaUrl);
    const posterBytes = await fetchBytes(posterUrl);

    const fileId = new Date().getTime() + "." + mediaFileType;
    const fileName = fileId;
    const posterFileId = new Date().getTime() + ".png";
    const posterFileName = posterFileId;

    const pageIndex = 0;

    const pageViewBox = viewer.getViewPort(pageIndex).viewBox;

    const rect = [
        (pageViewBox[2] - width) / 2,
        pageViewBox[3] - topMargin - height,
        (pageViewBox[2] - width) / 2 + width,
        pageViewBox[3] - topMargin
    ];

    viewer.storage.setItem(fileId, mediaBytes);
    viewer.storage.setItem(posterFileId, posterBytes);

    viewer.addAnnotation(pageIndex, {
        annotationType: 90, // RICHMEDIA
        subtype: "RichMedia",
        fileId,
        fileName,
        posterFileId,
        posterFileName,
        activationCondition: "XA",
        deactivationCondition: "XD",
        presentationStyle: "Embedded",
        showNavigationPane: false,
        rect
    });
}

// Load DsPdfViewer.
async function loadPdfViewer(selector){
    var viewer = new DsPdfViewer(selector, {
        supportApi: {
            apiUrl: 'http://localhost:5005/api/pdf-viewer',
            webSocketUrl: 'http://localhost:5005/signalr'
        }
    });
    viewer.addDefaultPanels();
    viewer.addAnnotationEditorPanel();
    viewer.addFormEditorPanel();
```

```
    await viewer.newDocument();

    // Use addRichMedia to insert RichMedia annotation.
    addRichMedia(viewer, "http://localhost/City_Noises.mp3",
"http://localhost/posterFile.png",
                                { width: 300, height: 150, topMargin: 100,
mediaFileType: "mp3" });
}
```

RichMedia Annotation Property Panel

The property panel of RichMedia annotation provides various properties and settings such as **Media**, **Poster**, **Activation**, **Deactivation**, **Presentation Style**, and **Navigation Controls**. The following table lists the editable properties of RichMedia annotation and their descriptions:

Property Name	Description
Media	Specifies the multimedia content associated with the annotation. You can also remove or download the media resources using **Remove file** and **Download file** buttons.
Poster	Defines the image that serves as the poster for the multimedia content. DsPdfViewer adds a poster image by default when you add RichMedia annotations. You can also remove or download the media resources using **Remove file** and **Download file** buttons.
Activation	Determines the action to activate the RichMedia annotation.
Deactivation	Specifies the action to deactivate the RichMedia annotation.
Presentation	Specifies the style in which the RichMedia content is presented: "Embedded" or "Windowed."
Controls	Controls whether navigation controls are displayed for interacting with the multimedia content. When set to true, this flag ensures that navigation controls are visible when the media content is initially activated; its default value is false.
Bounds	Specifies the position, width, and height of the boundary box for the RichMedia annotation.



Note: DsPdfViewer will prompt you to allow the playing of video in picture-in-picture mode if you set the presentation to windowed.

Limitations

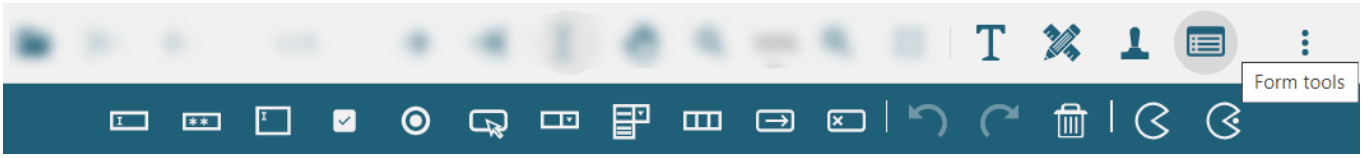
- DsPdfViewer does not support the following:
  - RichMedia presentation styles:
    - Transparent
    - Toolbar
    - PassContextClick
  - Position of the window in the "Windowed" presentation style
  - PlayCount and Speed animation styles
  - RichMedia activation through script
- Automatic playback of videos with sound when the page opens or the content is visible, as modern browsers do not allow this. You can change this behavior via the browser settings or preferences. For automatic playback to work, click anywhere in the browser window before scrolling to the page with the activated RichMedia annotation "Open on page/content visible." This action signals to the browser that you are interacting with the document so that it can start the automatic playback.

Form Editor

The Form Editor in DsPdfViewer 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

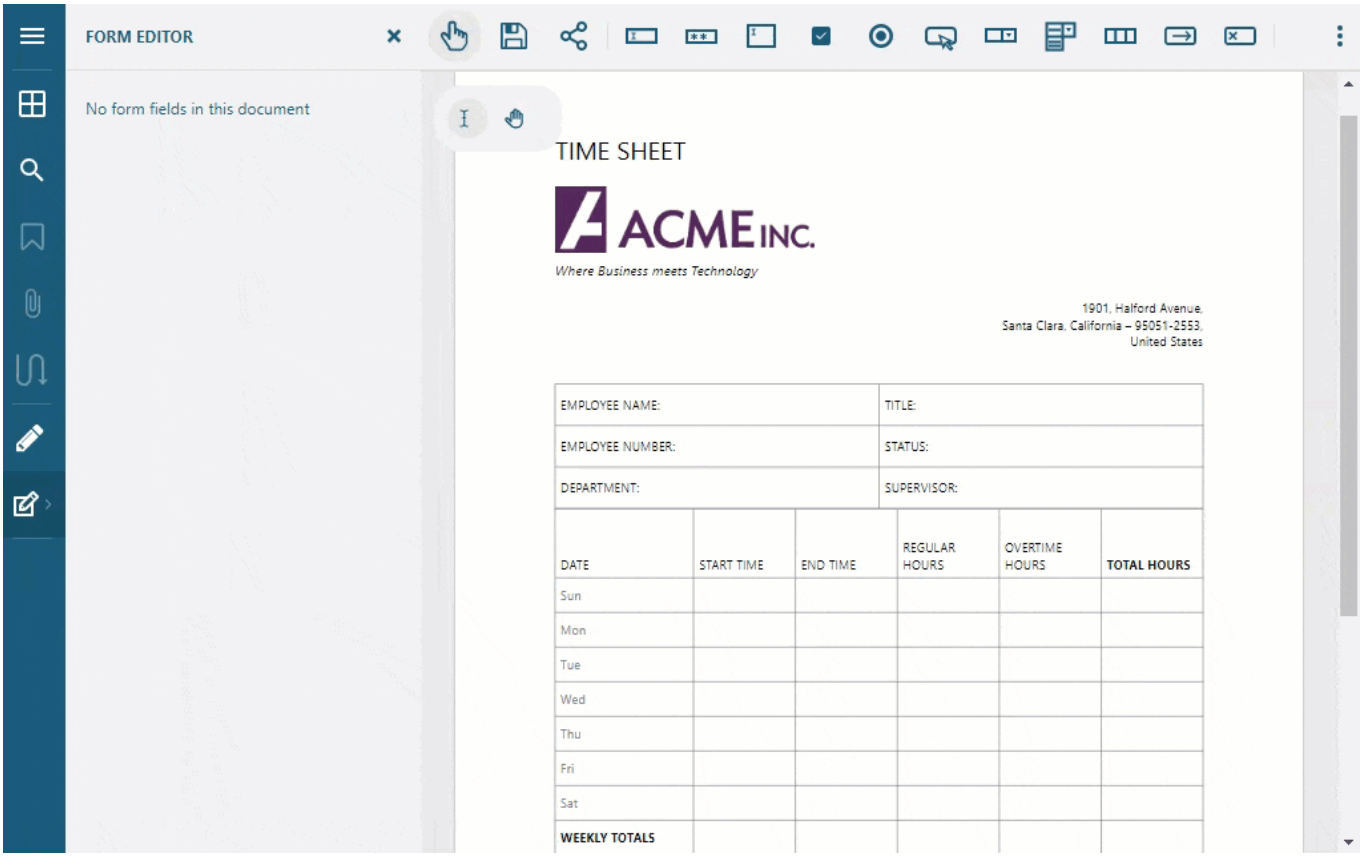
the main toolbar. It provides Form tools button which displays various form field buttons like text field, password field, radio button, checkbox etc. As can be observed, the 'Undo, Redo and Delete' and Redact options are also available in the quick editing toolbar.



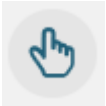
You can also configure which tools should be displayed in the quick editing toolbar of Form tools by using the **secondToolbarLayout** property. Also, you can use Page tools  to add a new blank document, insert a blank page, delete current page and perform undo or redo operations. To know more, refer 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 DsPdfViewer 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 ButtonCheckbox and Radio Button		Deletes the form field

Apart from the different types of form fields described above, DsPdfViewer 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
	Saves the modified document as images on client

	Creates a new blank document
	Inserts a blank page
	Deletes current page
	Reorders and reorganizes the pages in a PDF document. For more information, see PDF Organizer.

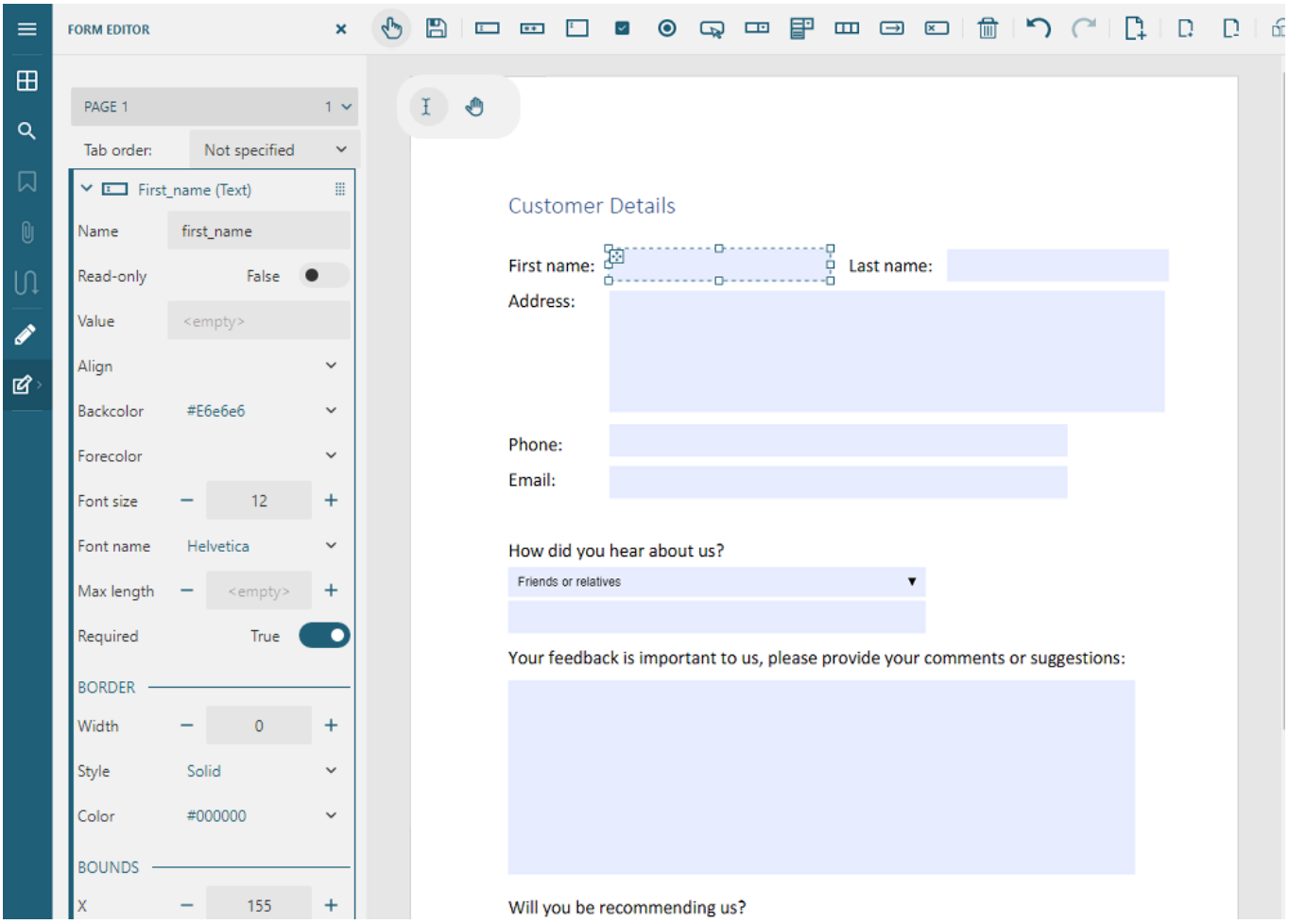
Note: You can view, print or download the original PDF document at any point of time by using the 'Hide Annotations' button on the main toolbar.

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 DsPdfViewer

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

Index.cshtml

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

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

C#

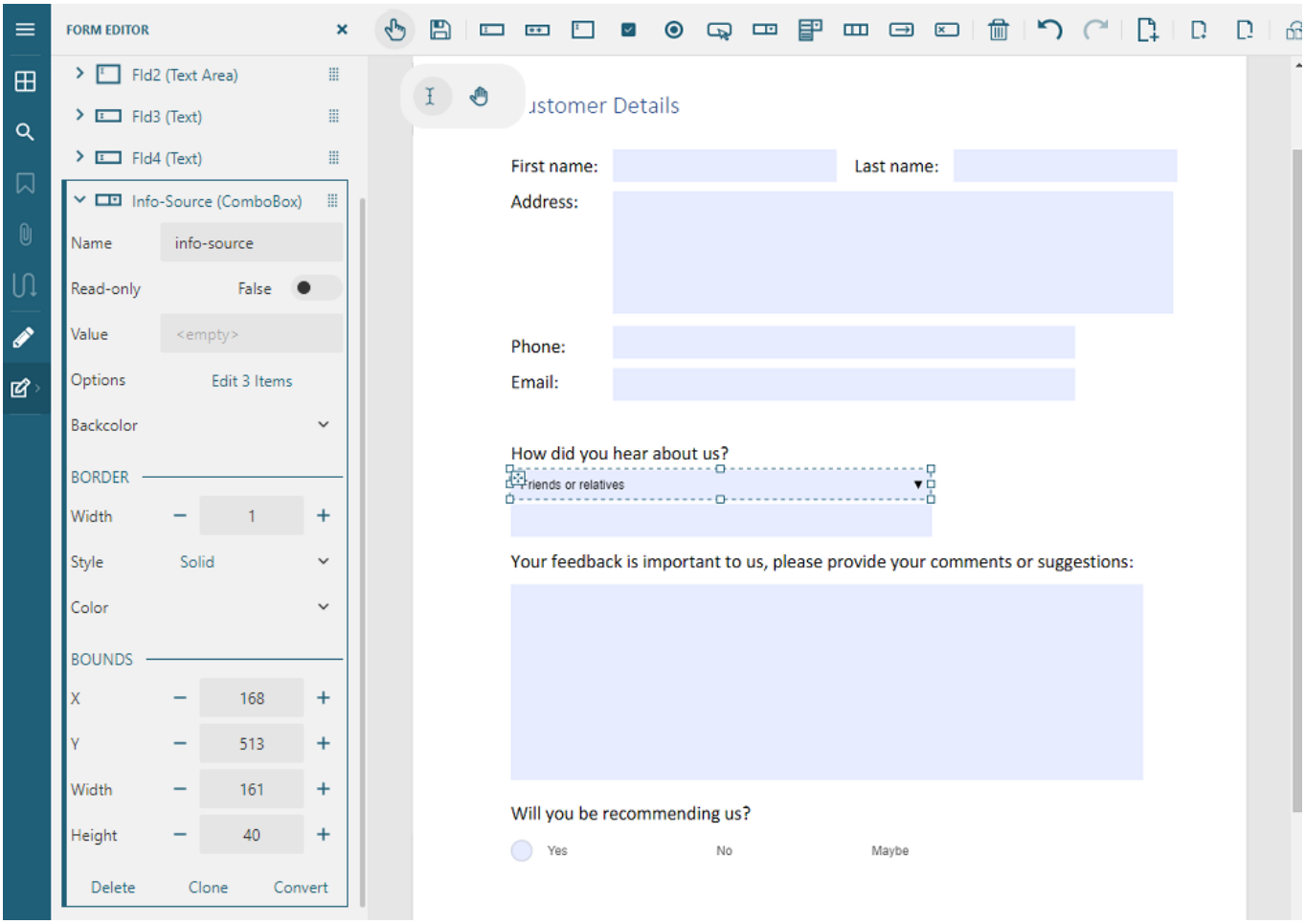
```
public static DsPdfViewerSupportApiDemo.Models.PdfViewerOptions PdfViewerOptions
        {
            get => new DsPdfViewerSupportApiDemo.Models.PdfViewerOptions(

DsPdfViewerSupportApiDemo.Models.PdfViewerOptions.Options.FormEditorPanel |

DsPdfViewerSupportApiDemo.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 DsPdfViewer:

1. Configure the DsPdfViewer 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.



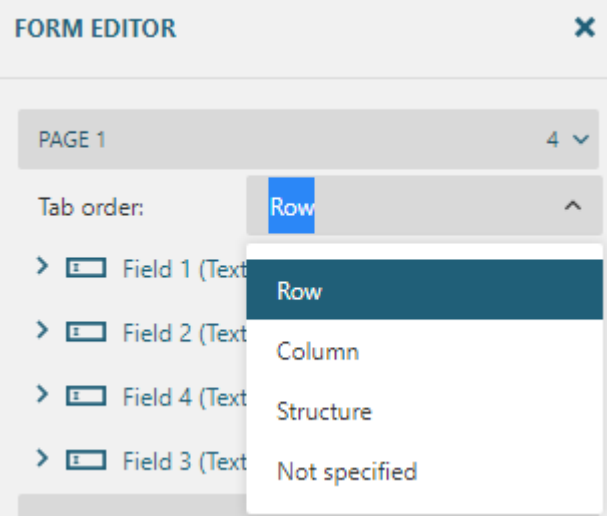
Tab Order of Form Fields

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. DsPdfViewer 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 DsPdf 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 DsPdfViewer	A detailed blog on how to use the DsPdfViewer to develop a Health Intake form for an online yoga class.
Form Editor	The online sample browser demonstrating the AcroForm editing features of DsPdfViewer.
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, DsPdfViewer 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;
```

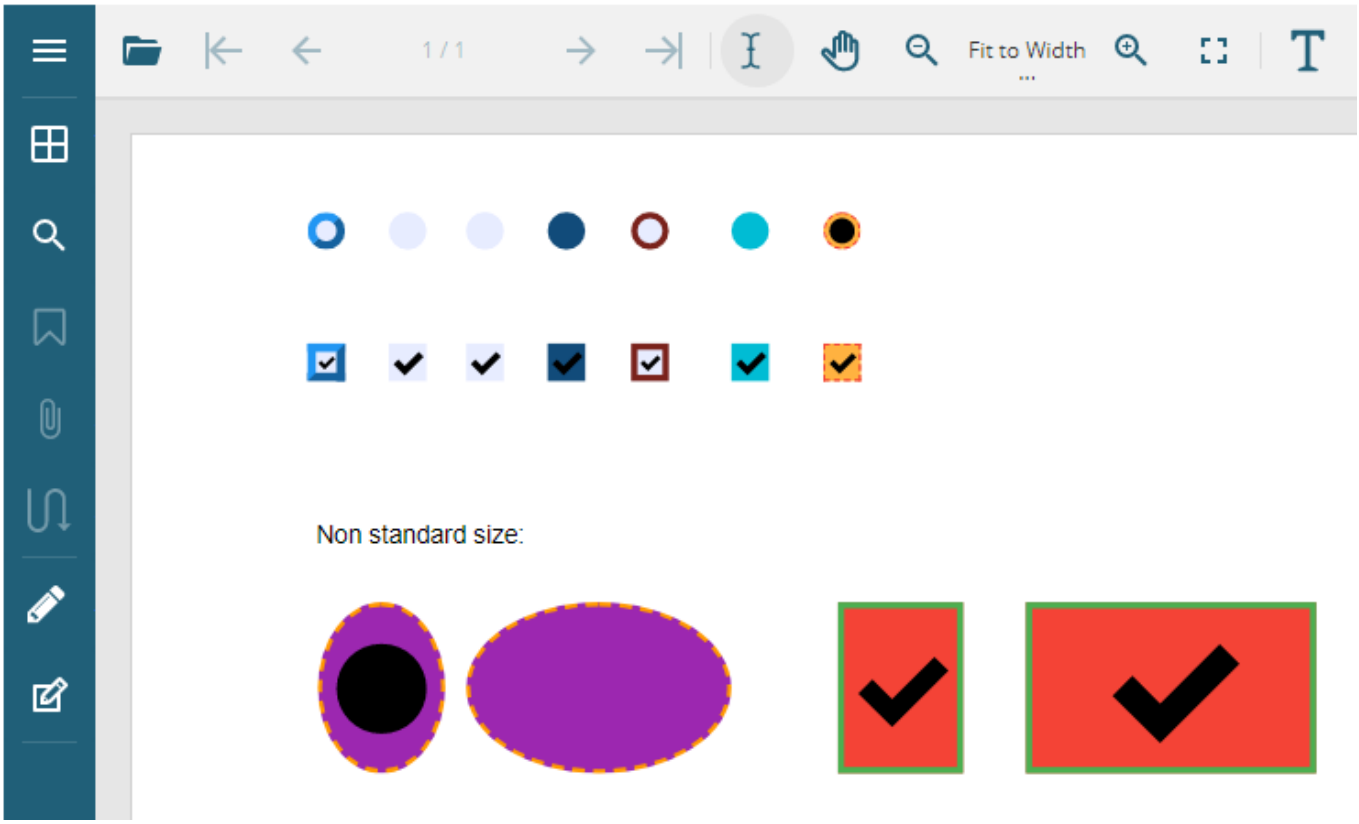
Checkbox and Radio Button

Checkbox and Radio Button Appearance

DsPdfViewer 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.

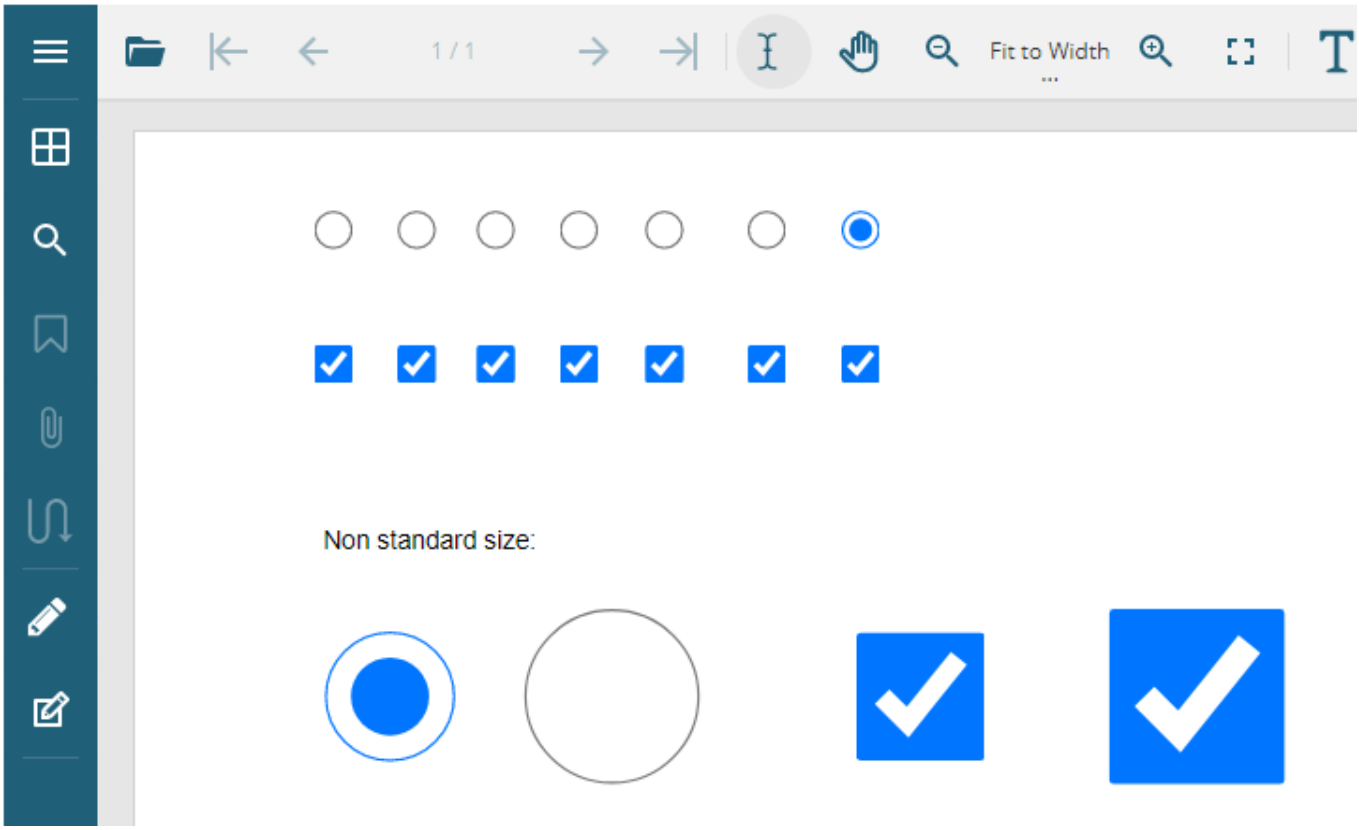
Custom

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:

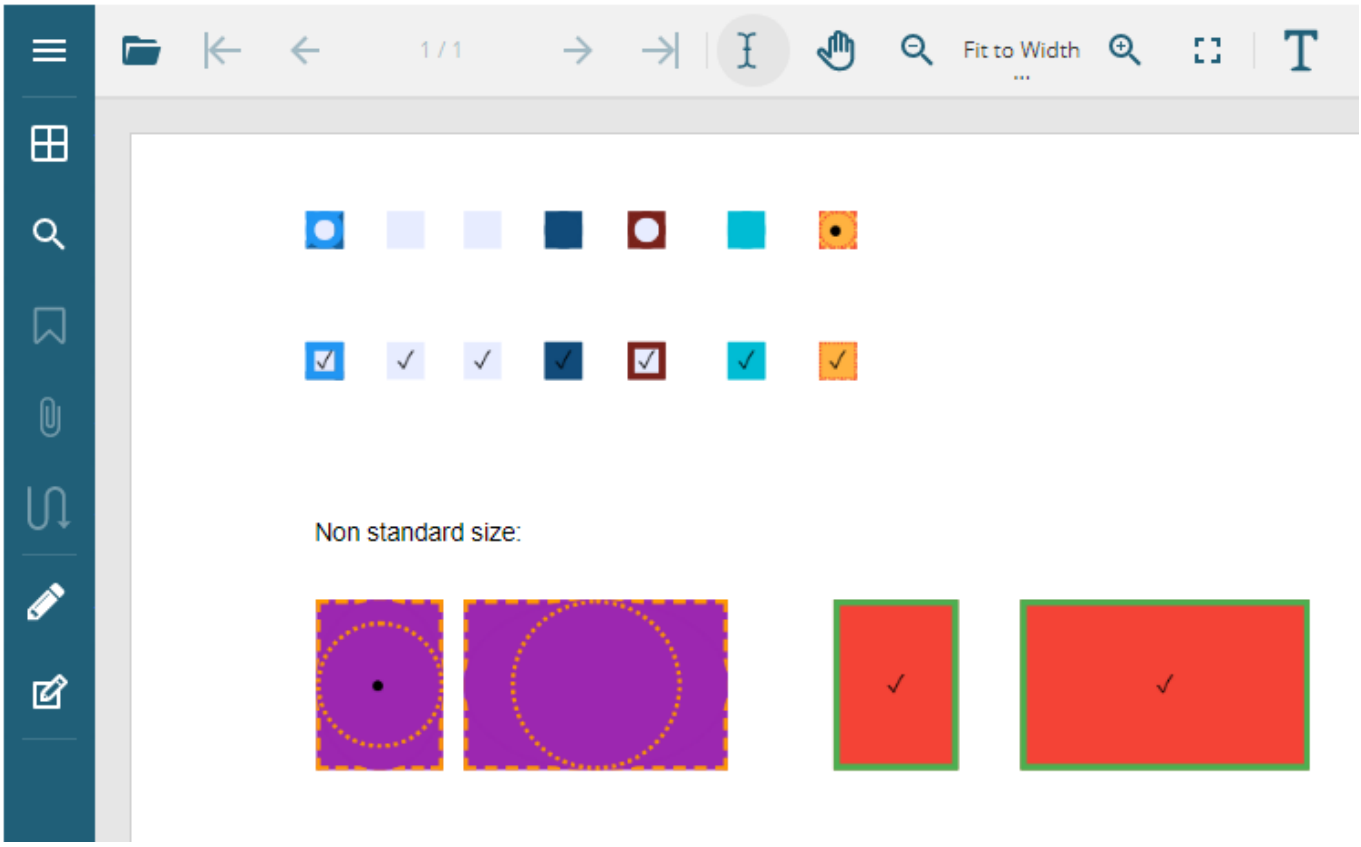


Index.cshtml

```
var viewer = new DsPdfViewer("#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

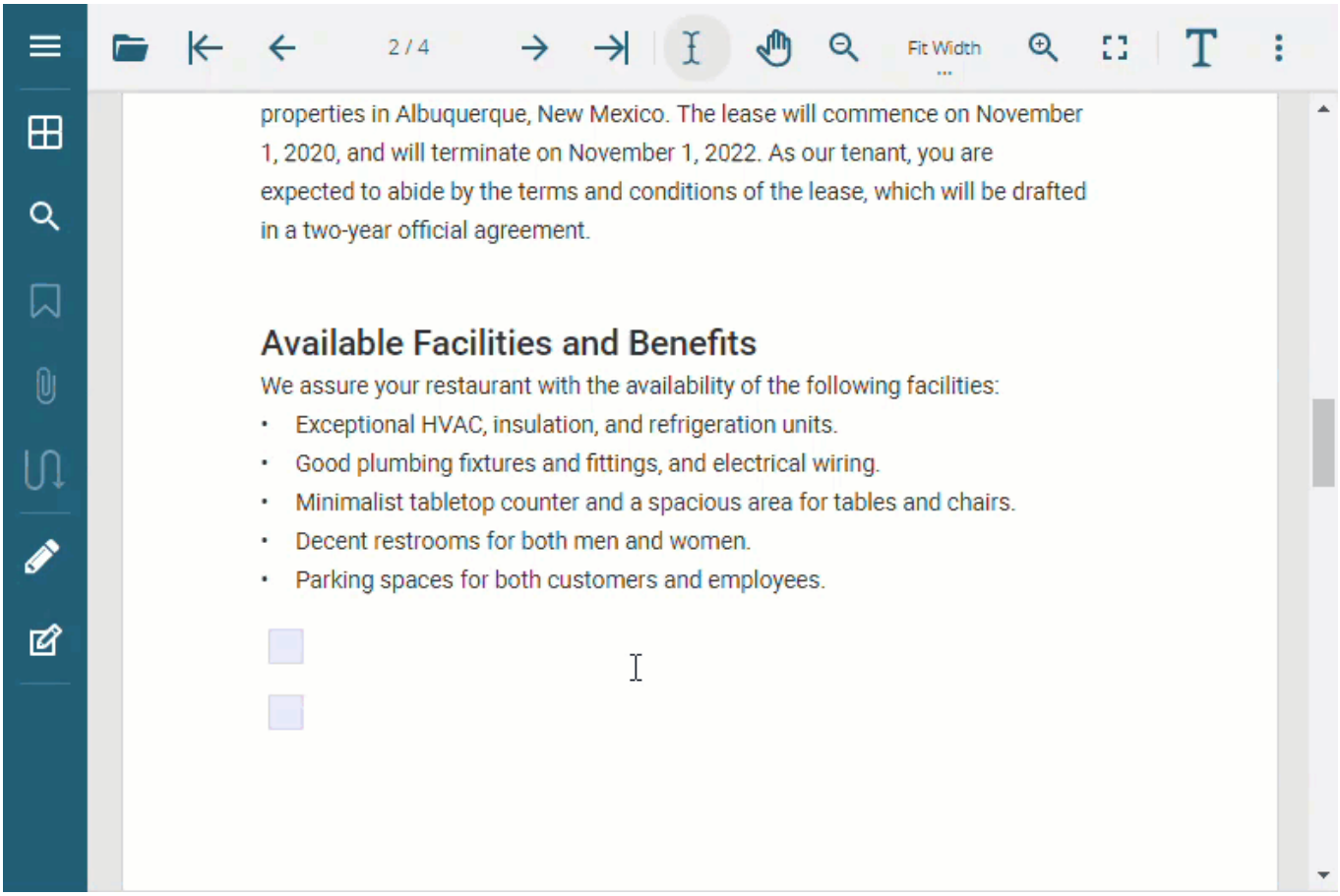
```
var viewer = new DsPdfViewer("#root", { fieldsAppearance: { radioButton:
"Predefined", checkBoxButton: "Predefined" } });
```

Set Checkbox Behavior as Radio Button

When you create checkboxes with the same names, DsPdfViewer allows you to set the behavior of each checkbox, enabling you to check only one checkbox at a time using the **Value** and **Export Value** properties. The Value property refers to the **fieldValue** in client script code. This property is shared between checkboxes with the same name. The Export Value property refers to the **exportValue** property in the client script code. The checkbox is considered to be checked when the values of the Value and Export Value properties are equal. The Value property is set equal to the Export Value property as soon as the user checks a checkbox. Hence, if we assign a unique value to the Export Value property of each checkbox having the same name, then only one of them will be checked at a time.

For example, if there are two checkboxes with the same name, Fld1, the initial value of the Value property is set to Off, and the value of the Export Value property is set to Yes for the checkboxes. When you check a checkbox, the value of Value property of the checkbox becomes equal to the value of Export Value property, i.e. Yes. Since the checkboxes with the same name share the Value property, the value of Value property of the other checkbox is set to Yes, i.e., equivalent to the value of Export Value property, and all the checkboxes become checked.

So, to achieve Radio Button like behavior, set the value of Export Value property of each checkbox to 1 and 2, respectively, and then checking one checkbox sets the value of Value property of both the checkboxes to 1. But the value of Value property is equivalent to the value of Export Value property only for the first checkbox. Hence, only the checkbox you checked remains checked, while the other checkbox remains unchecked. This happens because as soon as you check any of the checkboxes, the value of Value property of all the checkboxes is set to be equivalent to the value of Export Value property of the checked checkbox. Since each checkbox has a unique value in the Export Value property, the value assigned to the Value property of the remaining checkboxes does not match the value of Export Value property, resulting in an unchecked box.

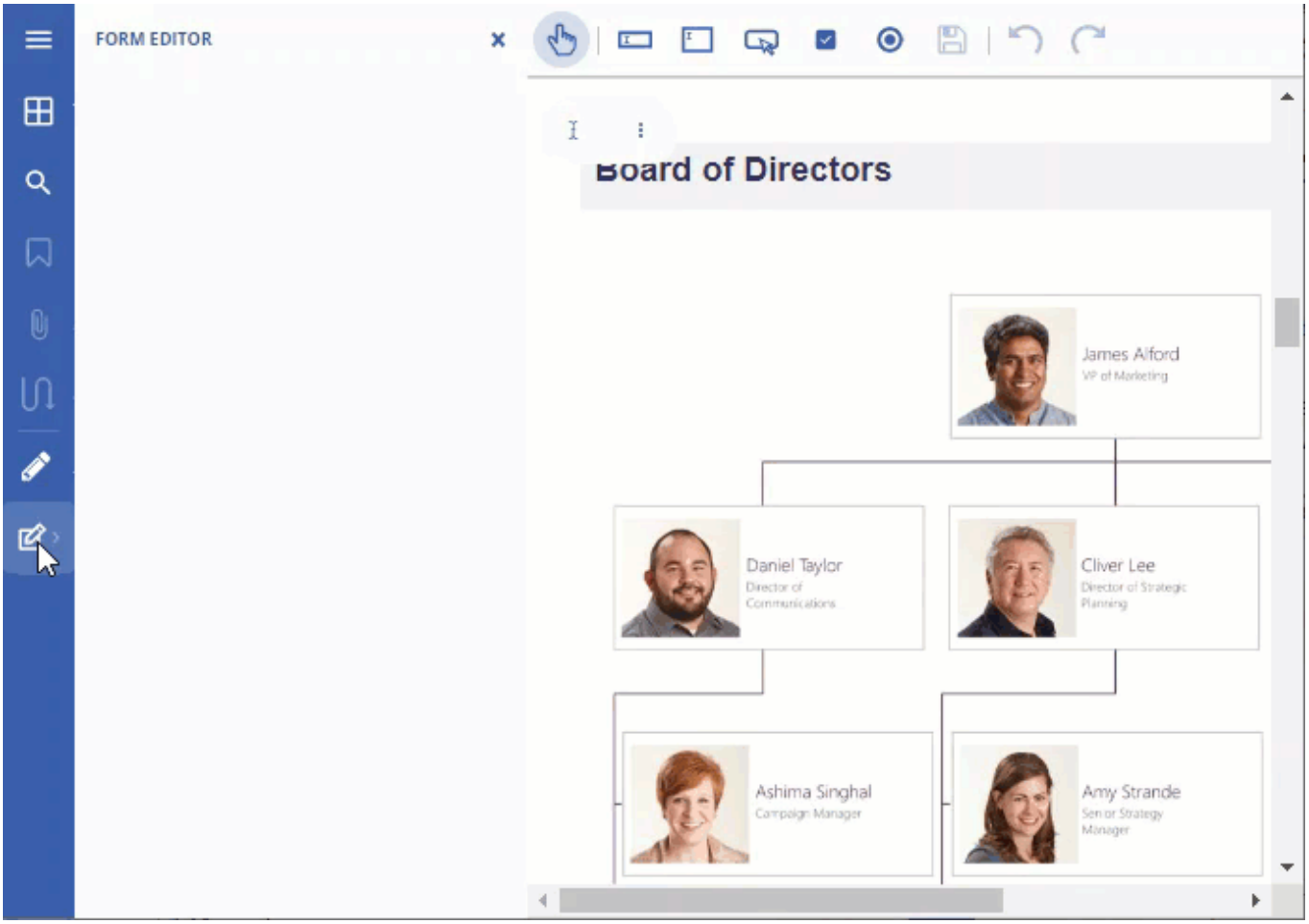


You can also check and uncheck a checkbox via client script code. Refer to the following example code to check and uncheck a checkbox:

```
// Set the Export Value to Yes.
checkBox.exportValue = "Yes";

// Set the Value to equal to Export Value to check the checkbox.
checkBox.fieldValue = checkBox.exportValue;
viewer.updateAnnotation(pageIndex, checkBox);


// Set the Value to Off to uncheck the checkbox.
checkBox.fieldValue = "Off";
viewer.updateAnnotation(pageIndex, checkBox);
```

Sticky Buttons

DsPdfViewer 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 DsPdfViewer.

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();
```

Custom Fonts

Sometimes you need custom fonts to use your creativity to create or edit a PDF document, which is not possible with the default fonts. With DsPdfViewer, you can add and use a list of new or custom fonts using **editorDefaults.fontNames** option.

To use the editorDefaults.fontNames option, you must meet the following pre-requisites:

- Register the new or custom fonts on the web page
- Configure SupportAPI to associate the client font names with the server fonts.

The following sections will guide you on the same:

Register New or Custom Fonts

Refer to the following example code to register two fonts using CSS:

```
// Define custom fonts.
<link rel="preload" href="budmo_jiggler.ttf" as="font" type="truetype" crossorigin>
<link rel="preload" href="FantaisieArtistique.ttf" as="font" type="truetype"
crossorigin>
<style>
    @font-face {
        font-family: BudmoJiggler-Regular;
        src: url(/documents-api-pdfviewer/demos/product-
bundles/assets/fonts/budmo_jiggler.ttf)  format('truetype');
    }
    @font-face {
        font-family: FantaisieArtistique;
        src: url(/documents-api-pdfviewer/demos/product-
bundles/assets/fonts/FantaisieArtistique.ttf)  format('truetype');
    }
</style>
```

Configure SupportAPI

DsPdfViewer provides three different methods for associating client-side font names with server-side fonts:

- **Using the ClientFonts Setting**
- **Using the DocumentInitialized event and FontCollection property**
- **Using the ResolveFont event for font resolution**

Using ClientFonts Setting

```
public void Configuration(IAppBuilder app)
{
    app.UseCors(CorsOptions.AllowAll);
```

```
    GcPdfViewerHub.Configure(app);
    GcPdfViewerController.Settings.VerifyToken += VerifyAuthToken;
    GcPdfViewerController.Settings.Sign += OnSign;

    // The ClientFonts property holds font mappings associated with their respective
client names.
    GcPdfViewerController.Settings.ClientFonts.Add("BudmoJiggler-Regular",
Font.FromFile("budmo_jiggler.ttf"));
    GcPdfViewerController.Settings.ClientFonts.Add("FantaisieArtistique",
Font.FromFile("FantaisieArtistique.ttf"));
    GcPdfViewerController.Settings.ClientFonts.Add("Yu Gothic UI",
Font.FromFile("yu_gothic_ui.ttf"));
}
```

Using DocumentInitialized Event and FontCollection Property

```
public void Configuration(IAppBuilder app)
{
    app.UseCors(CorsOptions.AllowAll);
    GcPdfViewerHub.Configure(app);
    GcPdfViewerController.Settings.VerifyToken += VerifyAuthToken;
    GcPdfViewerController.Settings.Sign += OnSign;
    GcPdfViewerController.Settings.DocumentInitialized += OnDocumentInitialized;
}

private void OnDocumentInitialized(object sender, DocumentInitializedEventArgs e)
{
    // Resolve the "FantaisieArtistique" font using the GcPdfDocument's
FontCollection property.
    var doc = e.Document;

    // Create a FontCollection instance.
    FontCollection fc = new FontCollection();

    // Get the font file using RegisterFont method.
    string projectRootPath = HttpContext.Current.Server.MapPath("~");
    string fontPath = Path.Combine(projectRootPath,
"assets/FantaisieArtistique.ttf");
    fc.RegisterFont(fontPath);

    // Allow the SupportApi to find the specified fonts in the font collection.
    doc.FontCollection = fc;

}
```

Using ResolveFont Event for Font Resolution

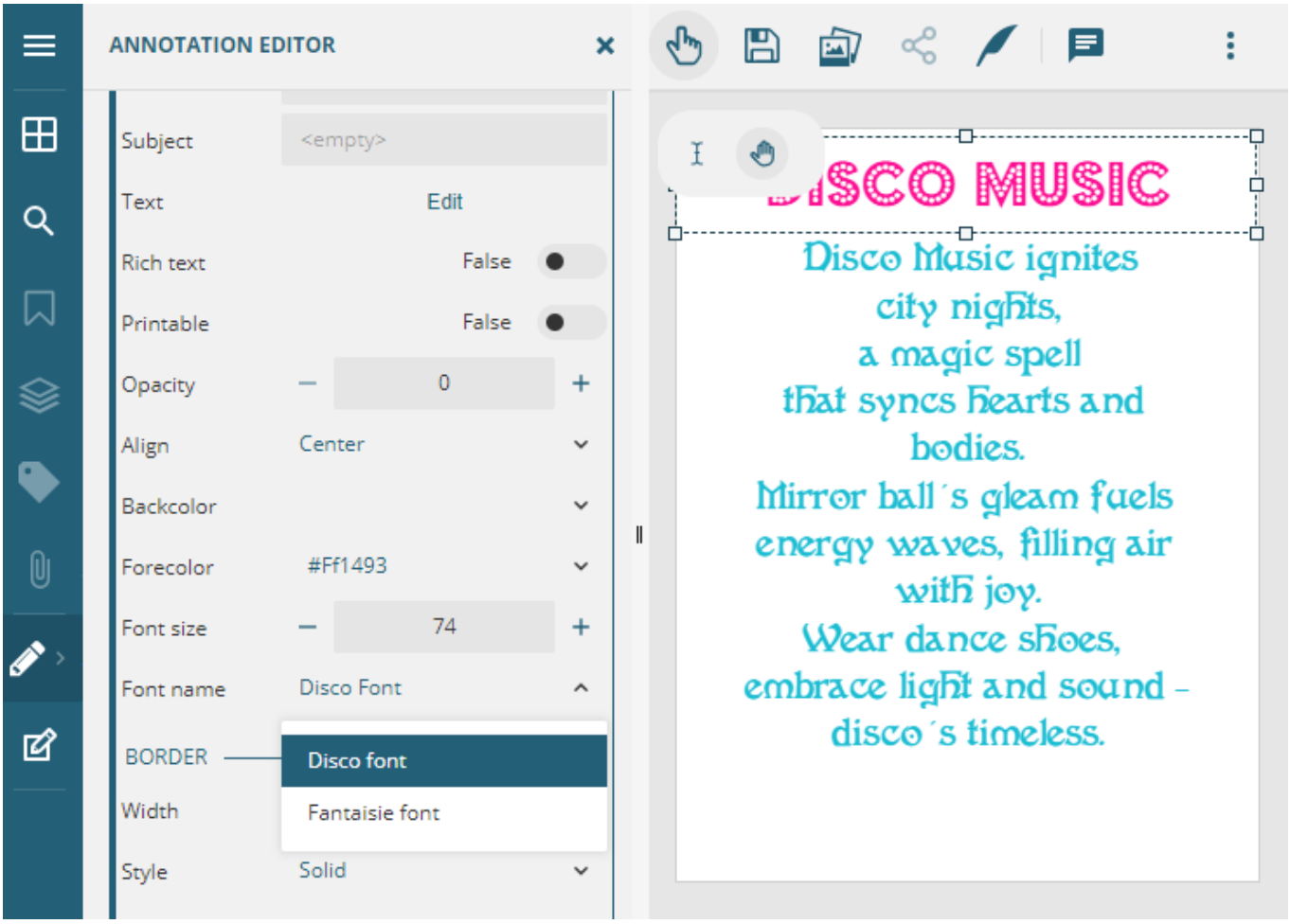
```
public void Configuration(IAppBuilder app)
{
    app.UseCors(CorsOptions.AllowAll);
```

```
    GcPdfViewerHub.Configure(app);
    GcPdfViewerController.Settings.VerifyToken += VerifyAuthToken;
    GcPdfViewerController.Settings.Sign += OnSign;
    GcPdfViewerController.Settings.ResolveFont += OnResolveFont;
}

private void OnResolveFont(object sender, ResolveFontEventArgs e)
{
    // Utilize the ResolvedFont event argument property to resolve the "BudmoJiggler-Regular" font.
    string projectRootPath = HttpContext.Current.Server.MapPath("~");
    if (e.FontName == "BudmoJiggler-Regular")
    {
        string fontPath = Path.Combine(projectRootPath, "assets/budmo_jiggler.ttf");
        e.ResolvedFont = Font.FromFile(fontPath);
    }
}
```

Setting Custom Font

After setting up the pre-requisites as described above, set the custom font using the editorDefaults.fontNames option and apply it to a PDF as described below:

```
// Set editor defaults for free text annotation.
viewer.options.editorDefaults.freeTextAnnotation = { fontName: 'FantaisieArtistique', borderStyle: { width: 0 } };

// Add font names.
viewer.options.editorDefaults.fontNames = [

        { name: 'Disco font', value: 'BudmoJiggler-Regular' },
        { name: 'Fantaisie font', value: 'FantaisieArtistique' }
    ];
```



Features

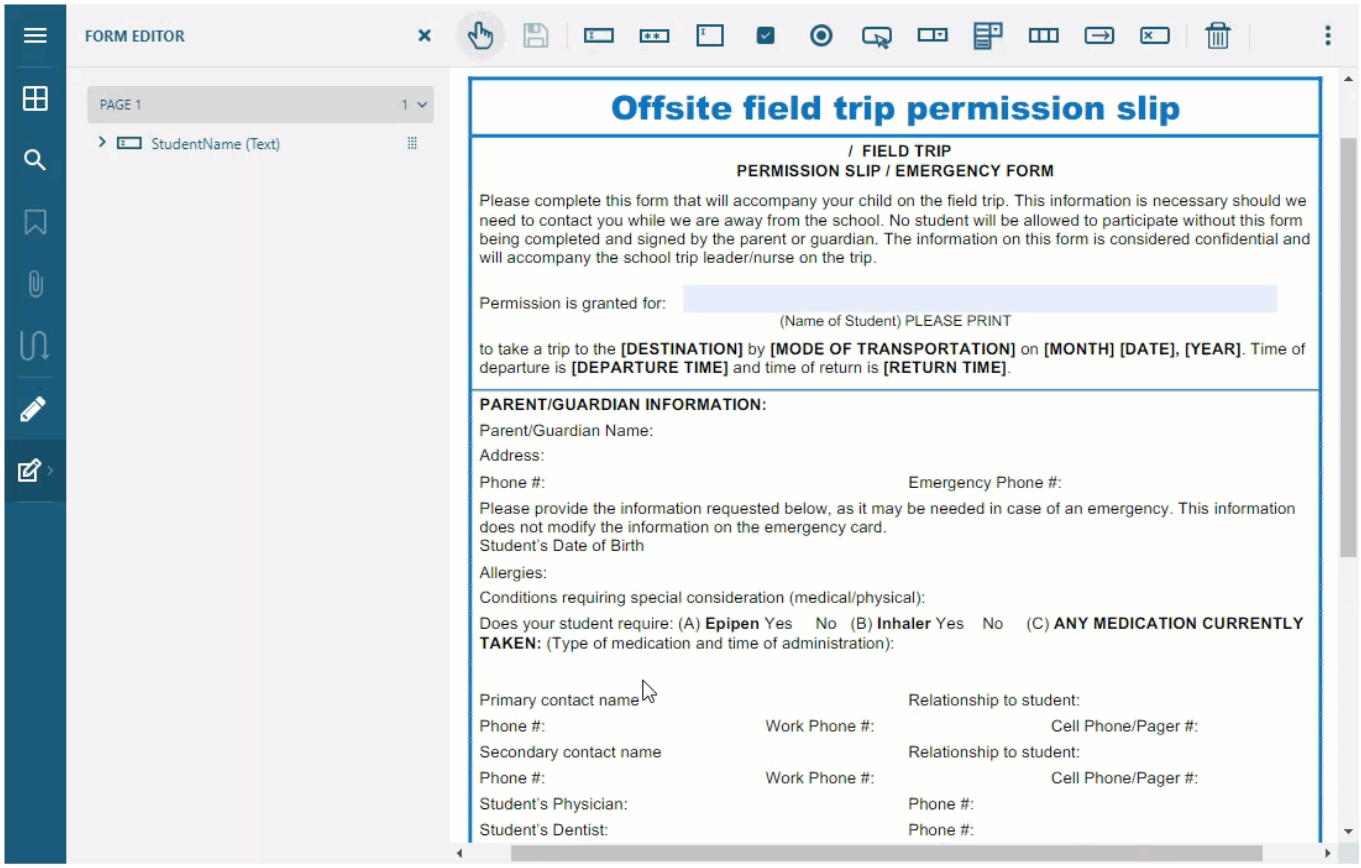
The Annotation and Form editor in DsPdfViewer provides various other useful features while editing PDF documents:

- Align
- Copy or Paste
- Convert to Content
- Default Settings
- Save as Image

Align

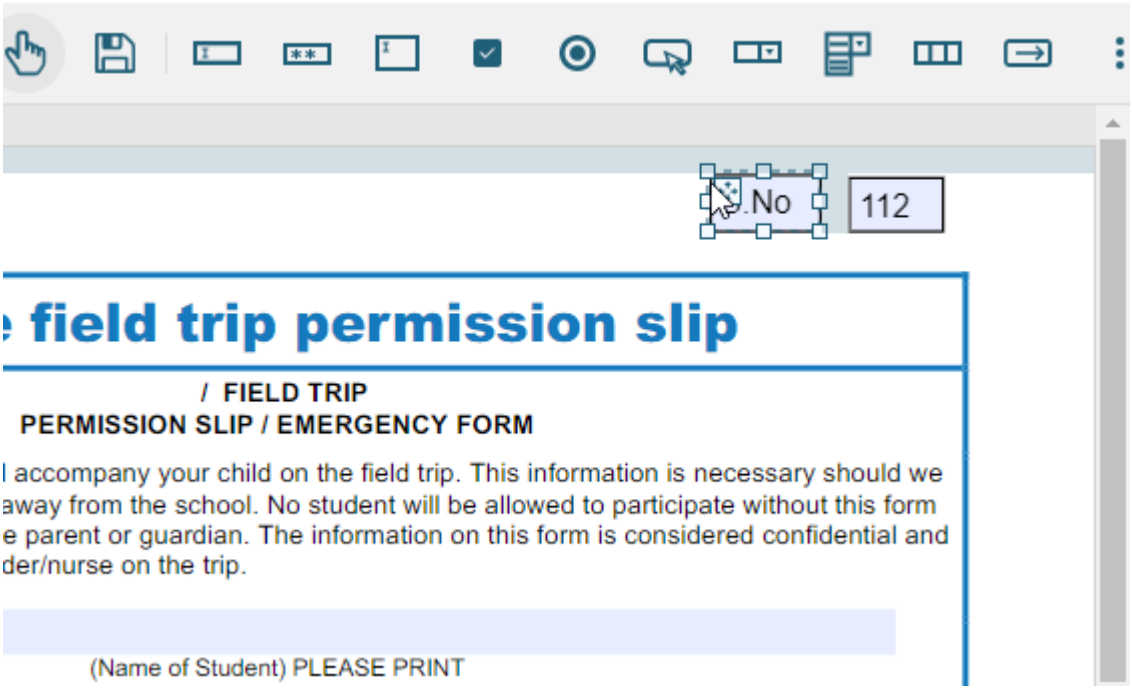
DsPdfViewer allows you to align annotations and form fields with each other. The snap alignment feature, which is enabled by default, can be used to align fields to the left, top, right, center or bottom edge.

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

The horizontal alignment can be disabled by using below code:

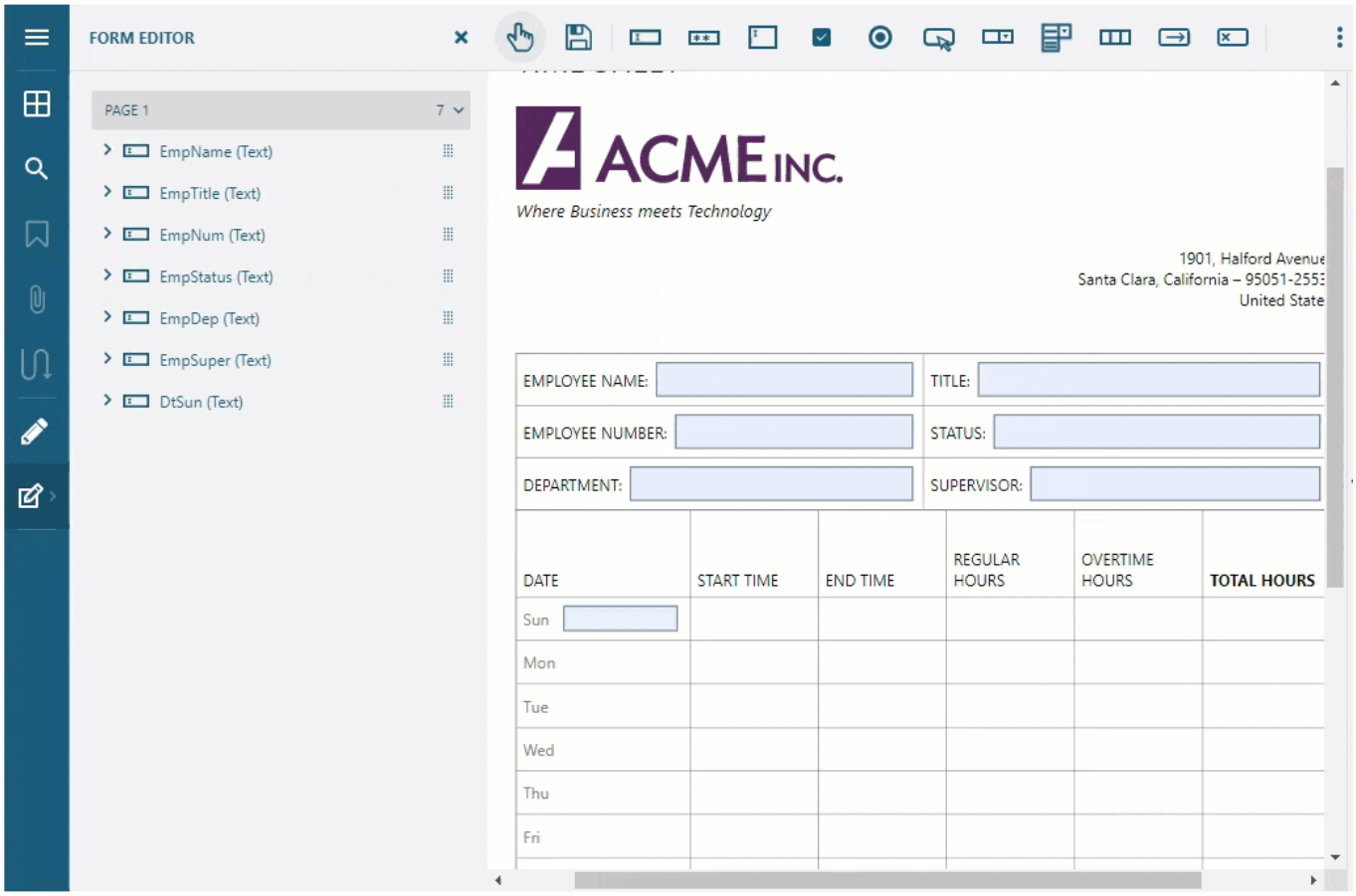
Index.cshtml

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

Copy and Paste

The Annotation and Form Editor in DsPdfViewer 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.



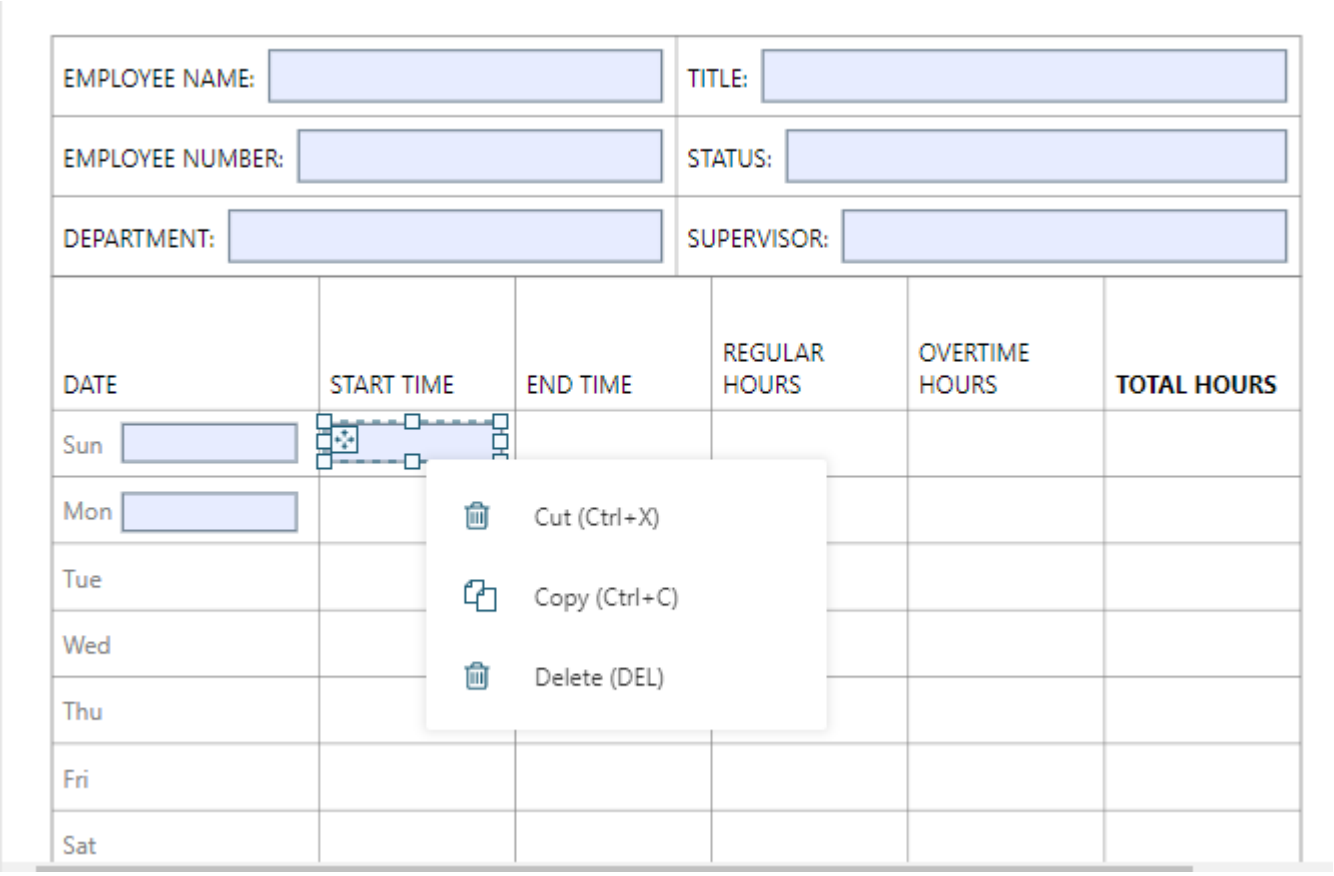
Note: The cloned annotation or form field has a new id and field name.

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.



Using Code

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

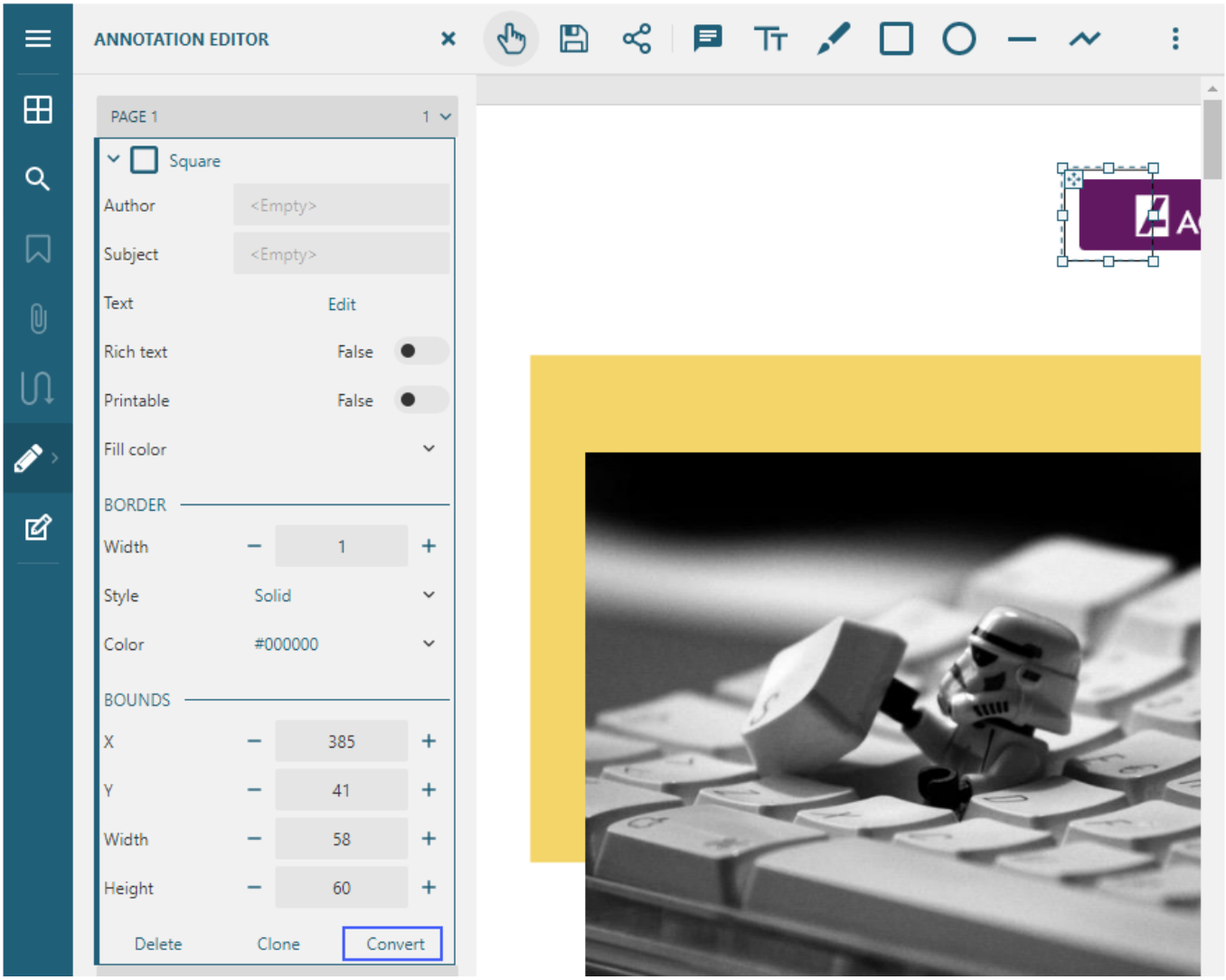
Index.cshtml

```
//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);
```

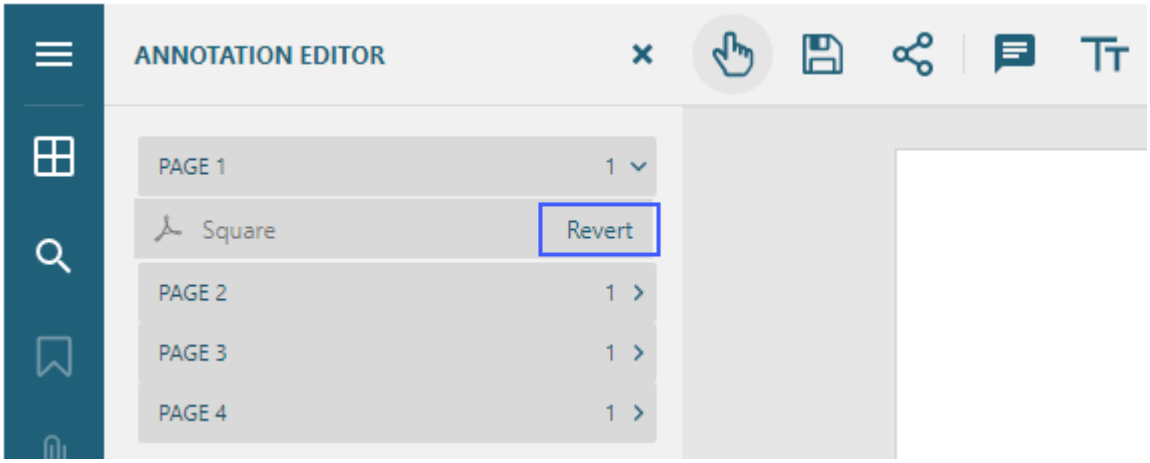
Convert to Content

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

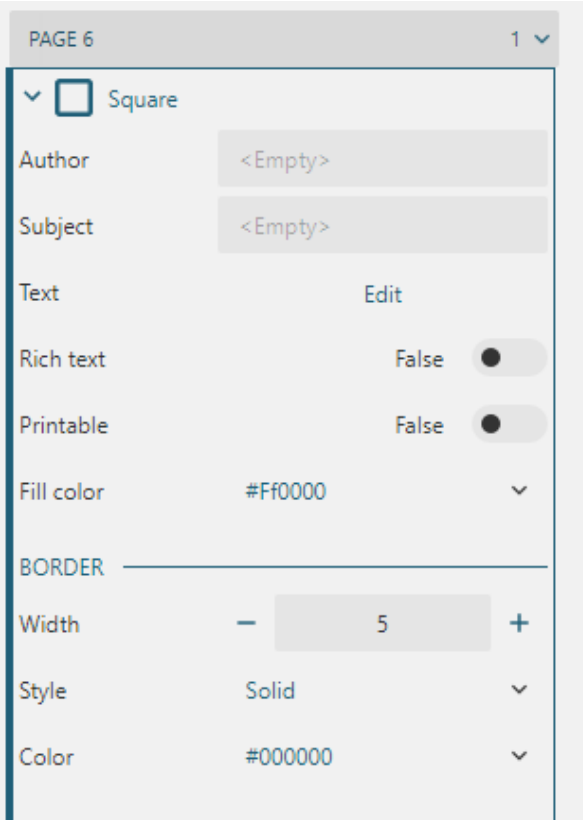
The values for annotation and form field properties can be set by using the properties panel of respective editors. However, you can also set the default value of any annotation or form field property by using the **editorDefaults** option.

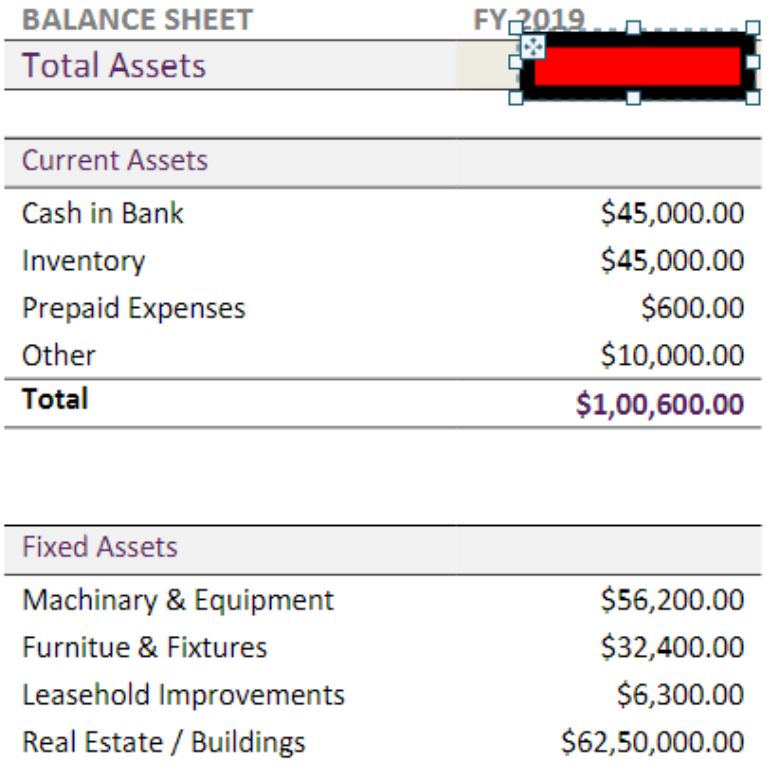
For example, the below sample code sets default values of square annotation properties like border width and interior (fill) color:

Index.cshtml

```
var viewer = new DsPdfViewer("#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:





Opacity

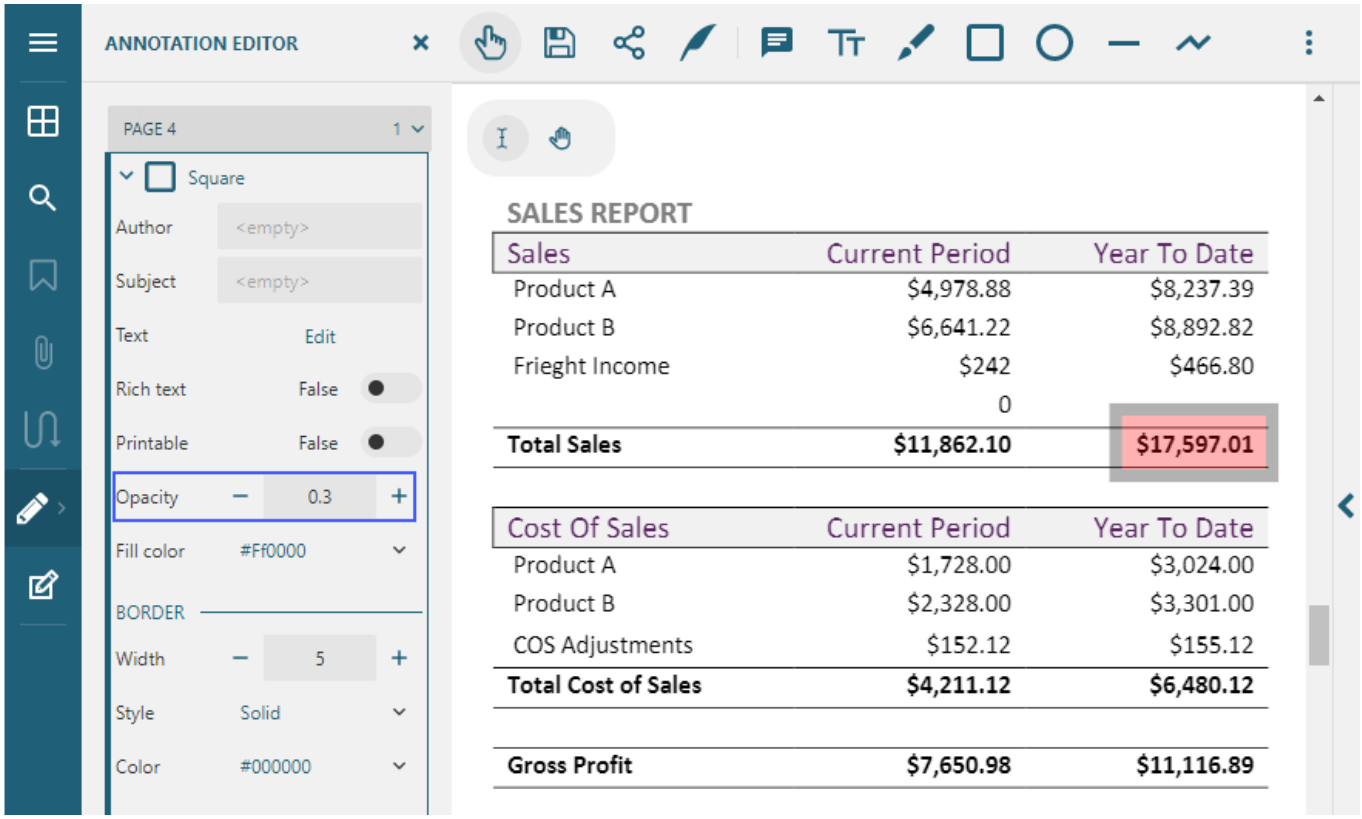
You can set the opacity property for all annotations. The valid value for the property can be within the range of 0.0 (fully transparent) to 1.0 (fully opaque). An annotation will appear as completely opaque if no opacity value is defined.

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

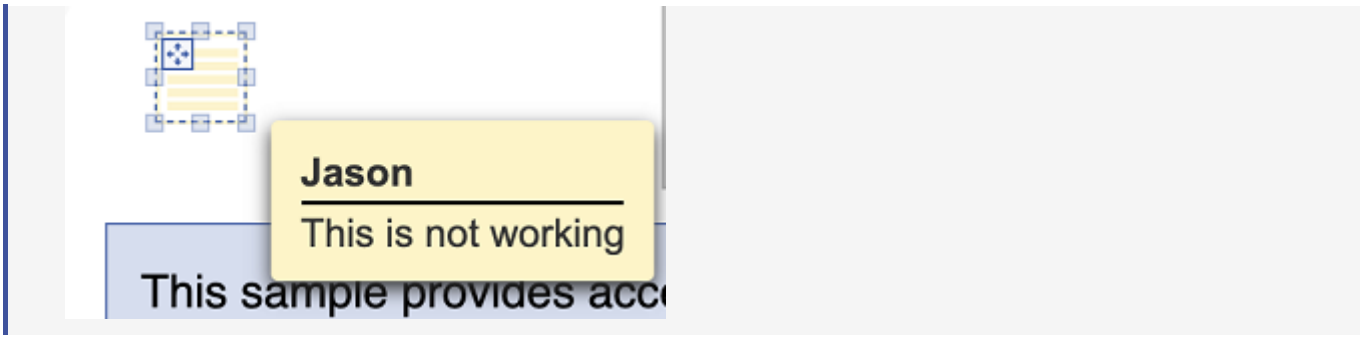
Index.cshtml

```
var viewer = new DsPdfViewer("#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:



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.

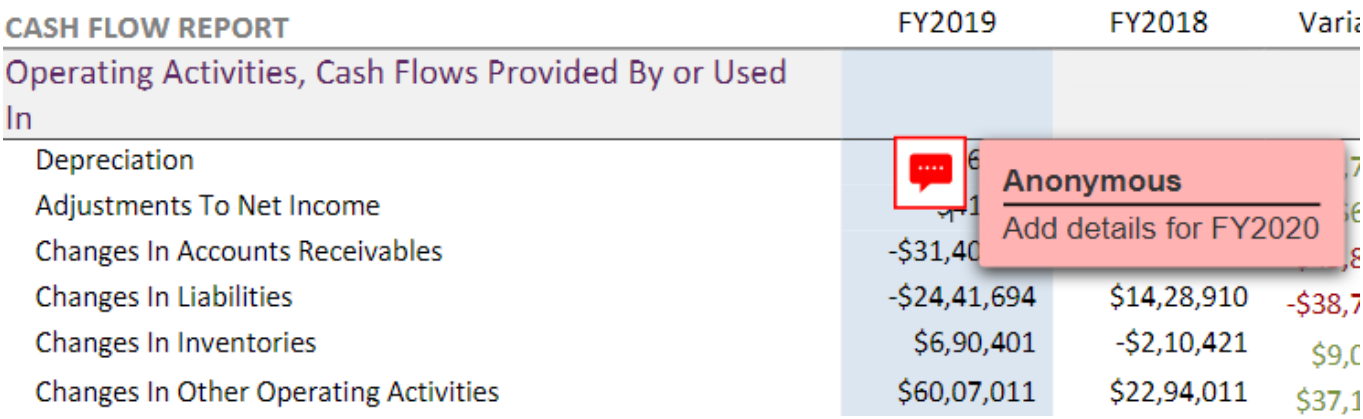
Default Color for Sticky Notes

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

Index.cshtml

```
var viewer = new DsPdfViewer("#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:



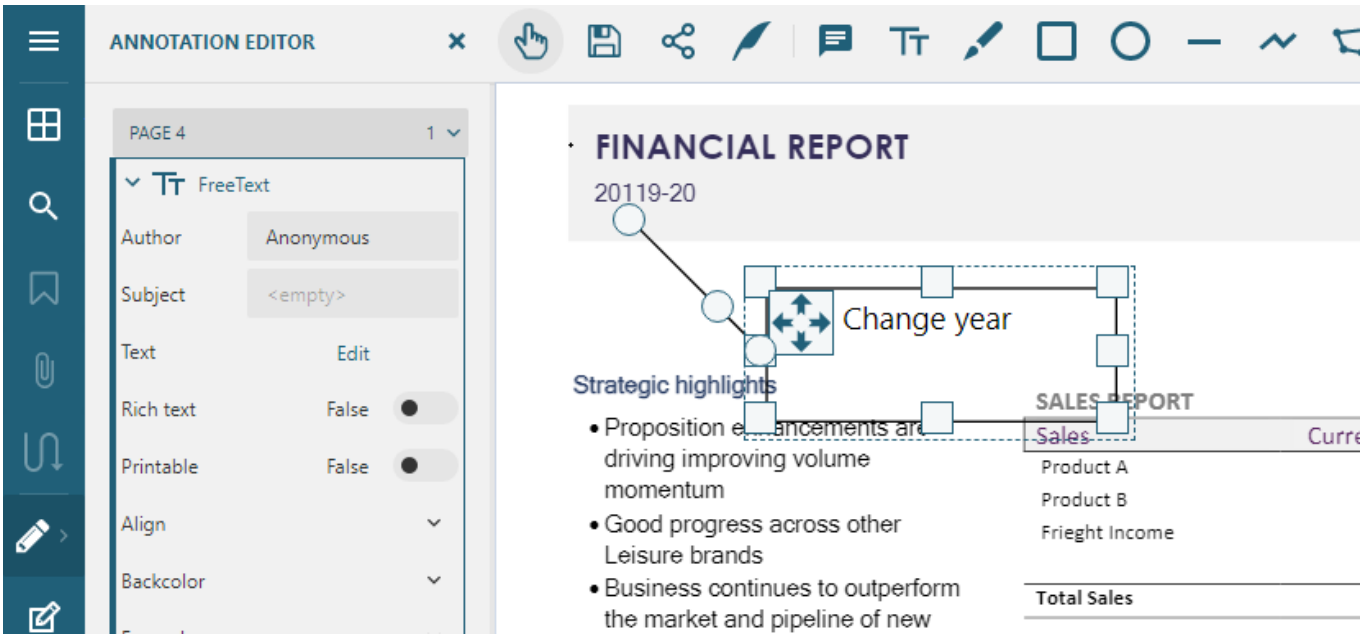
Handle Sizes

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

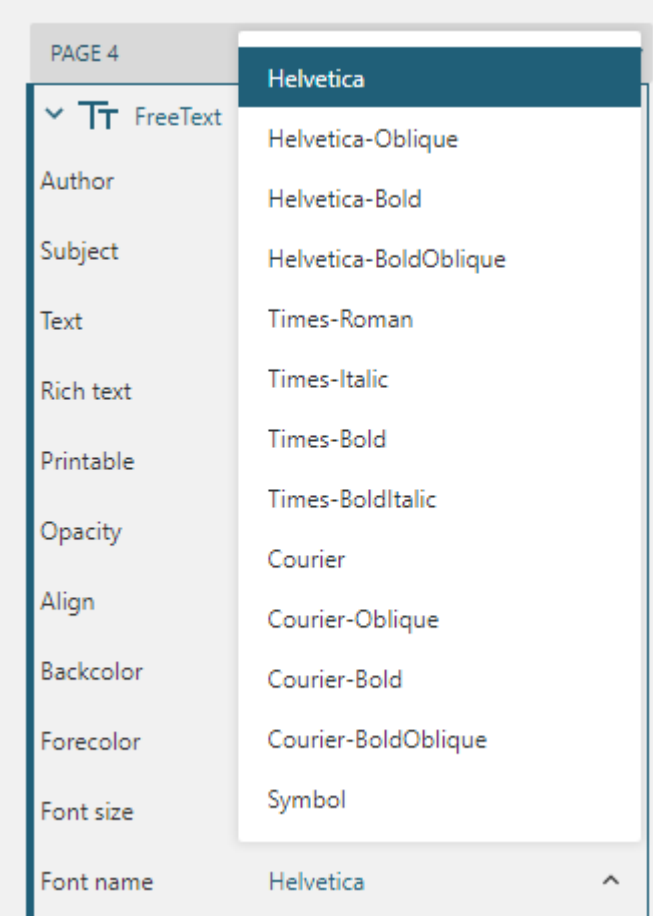
```
<script>
    var viewer = new DsPdfViewer("#host",
        {   editorDefaults: {
                resizeHandleSize: 20,
                moveHandleSize: 40,
```

```
                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 DsPdfViewer, by default.



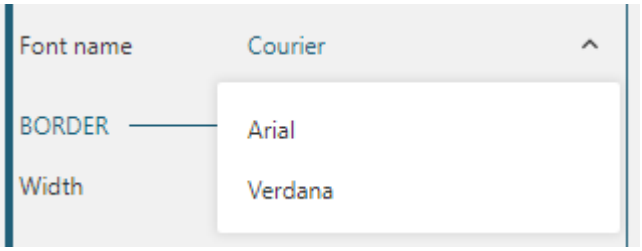
However, you can specify the default font names for the text fields and FreeText annotation by setting the **fontName** property in **editorDefaults** option.

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 DsPdfViewer("#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.

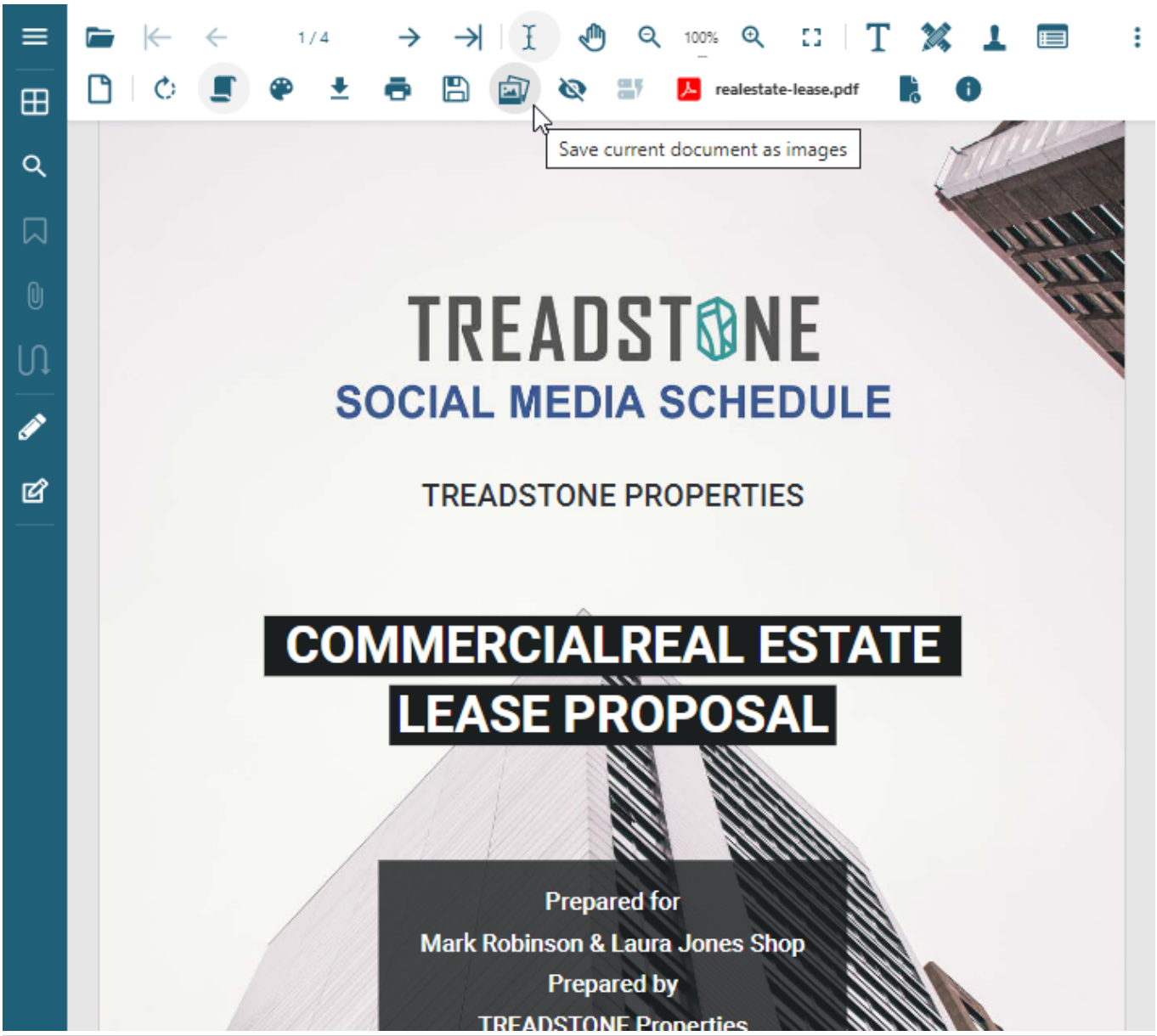
> **Note:** When a custom font is specified by using the fontNames setting, that font should be available in the web browser too.

For more information, refer Editor Defaults in DsPdfViewer demos.

Save as Image

DsPdfViewer allows you to save the modified PDF document as a set of PNG images and downloads the resultant zip archive on the client machine. You can achieve it using the "**Save current document as images**" button which is placed next to the "**Save current document**" button on the "Annotation editor" and "Form editor" toolbar. While working on code, you can use **SaveAsImages** method which accepts name of the output zip archive as parameter and downloads the current PDF document as PNG images in zipped format.

The image below shows "Save current document as images" button in the Annotation and Form editor toolbar:



C#

```
// Save pages of the current PDF document as PNG images to the specified zip file
viewer.saveAsImages('test.zip');
```

PDF Organizer

DsPdfViewer provides **PDF Organizer** toolbar button  that allows a user to reorder and reorganize the pages in a PDF document using individual page numbers or page ranges. This button also allows a user to merge pages from two different PDFs and split a PDF into multiple different PDFs.

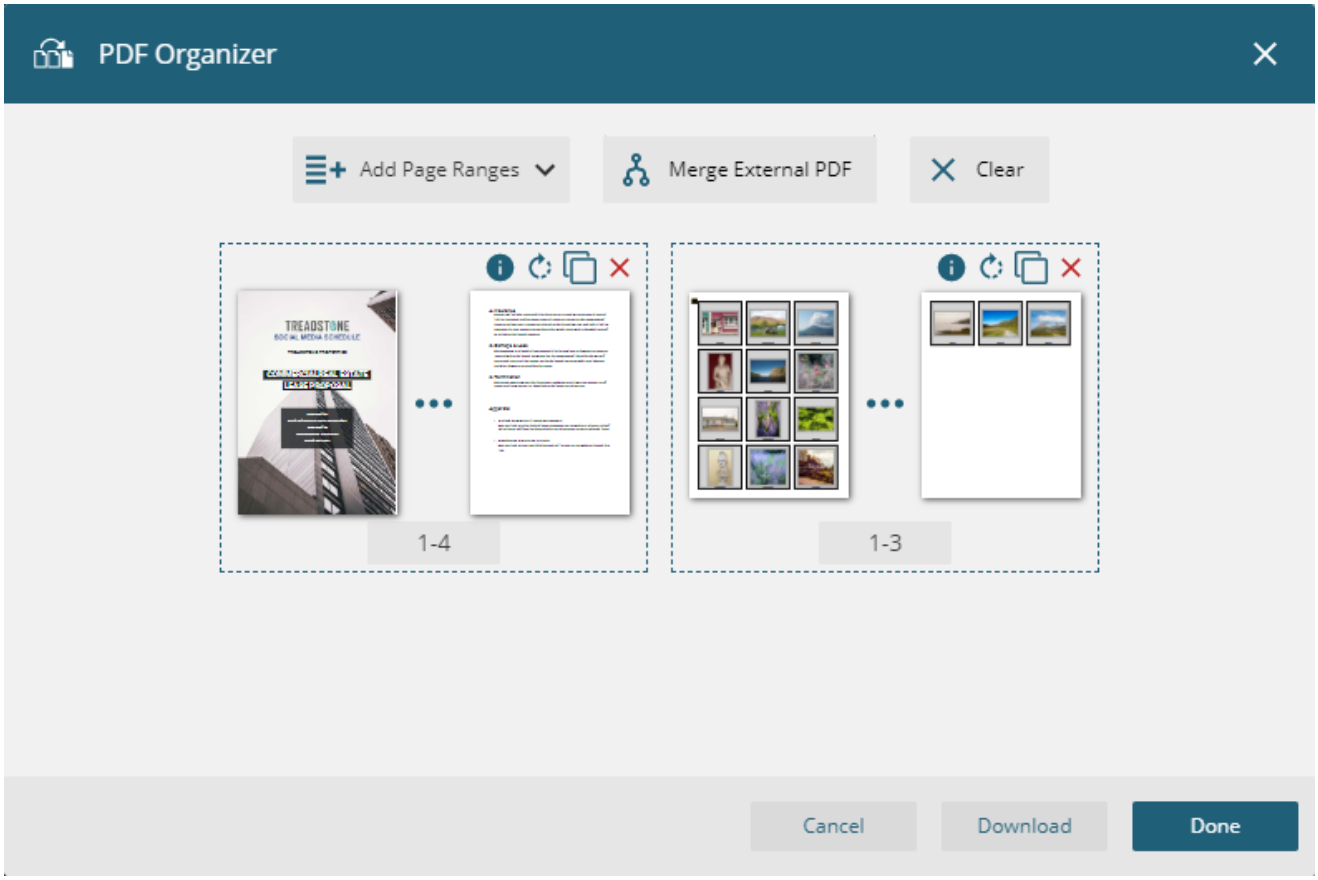
The PDF Organizer toolbar button is present in the following toolbar layouts:

- Page Tools

- Annotation Editor

- Form Editor

When the user clicks the PDF Organizer toolbar button, the following dialog appears:



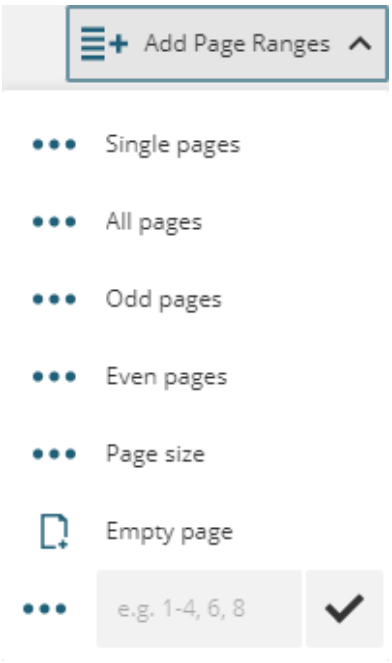
The following table lists the features provided by the PDF Organizer toolbar button.

Features	Icons	Description
Add Page Ranges	Add Page Ranges	Allows a user to add a single page ranges, all pages range, odd pages range, even pages range, each page size/orientation range, an empty page range, or a custom range from the active PDF document.
Merge External PDF	Merge External PDF	Allows a user to add a single page or page ranges from external PDF document to the end of an active PDF document.
Clear	Clear	Allows a user to clear all the ranges.
Download	Download	Allows a user to download the organized PDF document to the computer's local file system.
Done	Done	Allows a user to apply changes and close the PDF Organizer dialog. **Note:** After applying the changes to the active PDF document, it is not possible to revert back to the previous state. The viewer will load the modified PDF document with the new page structure.
Cancel	Cancel	Allows a user to close the PDF Organizer dialog and cancel any changes made to the structure of the PDF document.
Split Range	•••	Allows a user to split a page range into single pages.
Remove	✕	Allows a user to remove a page or a page range from the PDF

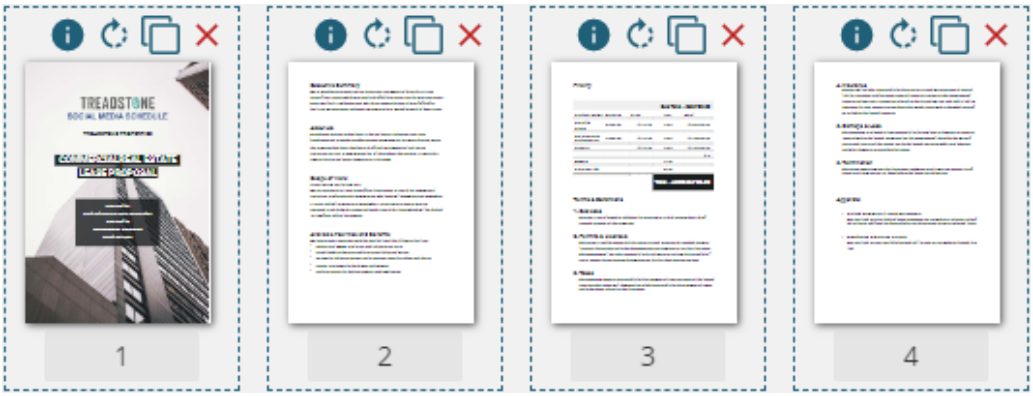
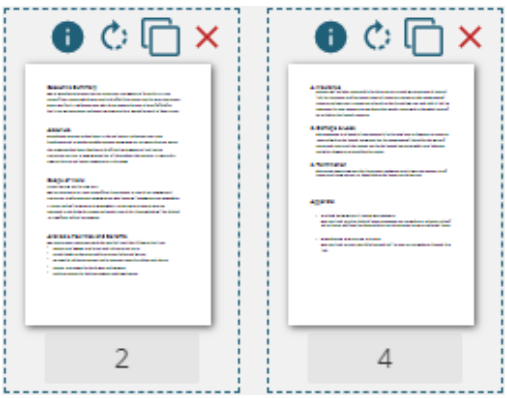
Range		Organizer dialog.
Clone Range		Allows a user to clone the page or page range.
Rotate Range		Allows a user to rotate a page or a page range by 90 degrees from the PDF Organizer dialog.
Information		Displays a user the information about a page or a page range.
Input Field		Allows a user to specify the start and end page numbers for a specific page range. Pressing “Enter” accepts the new value. Pressing “Escape” resets the range value to the previous one.

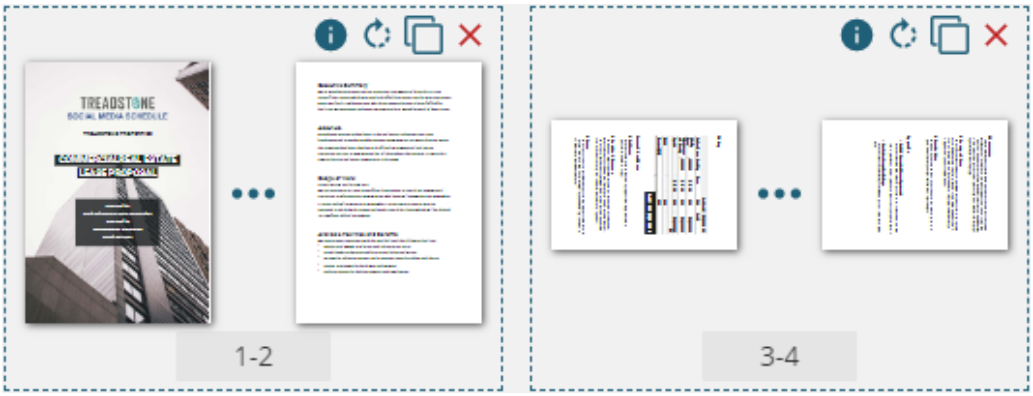
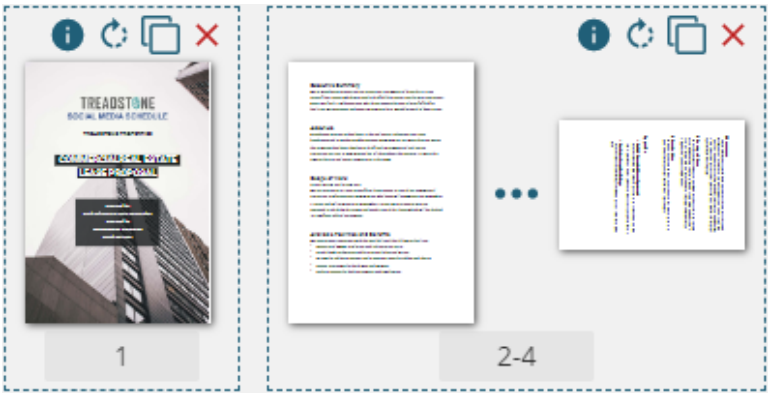
Note: The PDF Organizer also allows a user to reorder a single page or a page range by clicking on the page preview and dragging and dropping the page or range.

The following table lists the Add Page Ranges dropdown menu options:



Add Page Ranges Dropdown Menu Options	Description	Resulting Range
Single	Allows a user	

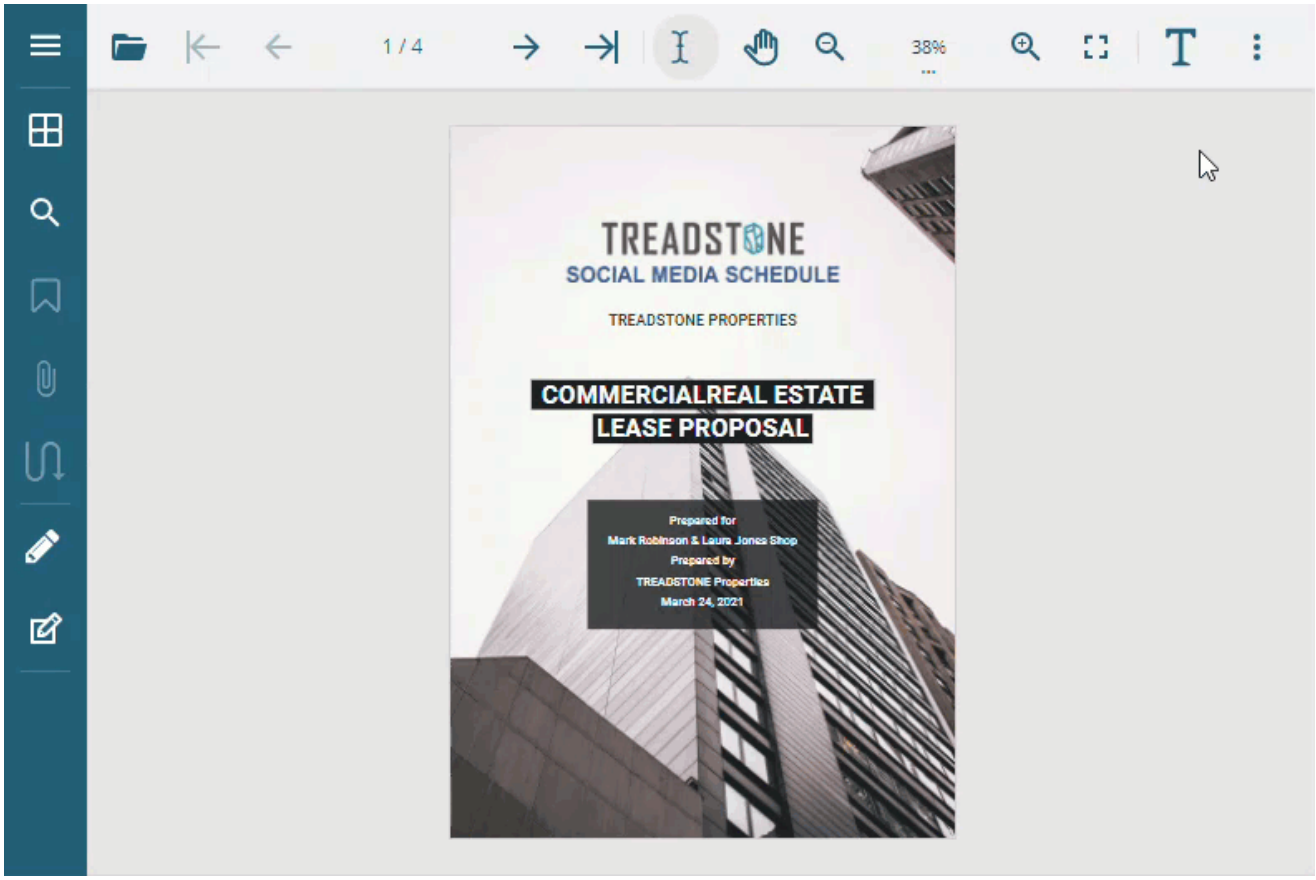
pages	to create a range for each page and provides maximum flexibility to manipulate individual pages.	
All pages	Allows a user to create one range for all pages and also allows manipulating all pages together as a single unit.	
Odd pages	Allows a user to create a range that includes all odd pages.	
Even pages	Allows a user to create a range that includes all even pages.	
Page size	Allows a user	

	to create a range for each page size and orientation.	
Empty page	Allows a user to create a range with a new empty page.	
Custom	Allows a user to create custom ranges from page numbers based on the input.	

Note: The key for PDF Organizer button is **pdf-organizer** which can be used to customize the toolbar.

Reorder PDF Document

PDF Organizer allows a user to reorder and reorganize a PDF document by modifying the page number or page range in the input field or by dragging and dropping a page or page range into the required place.

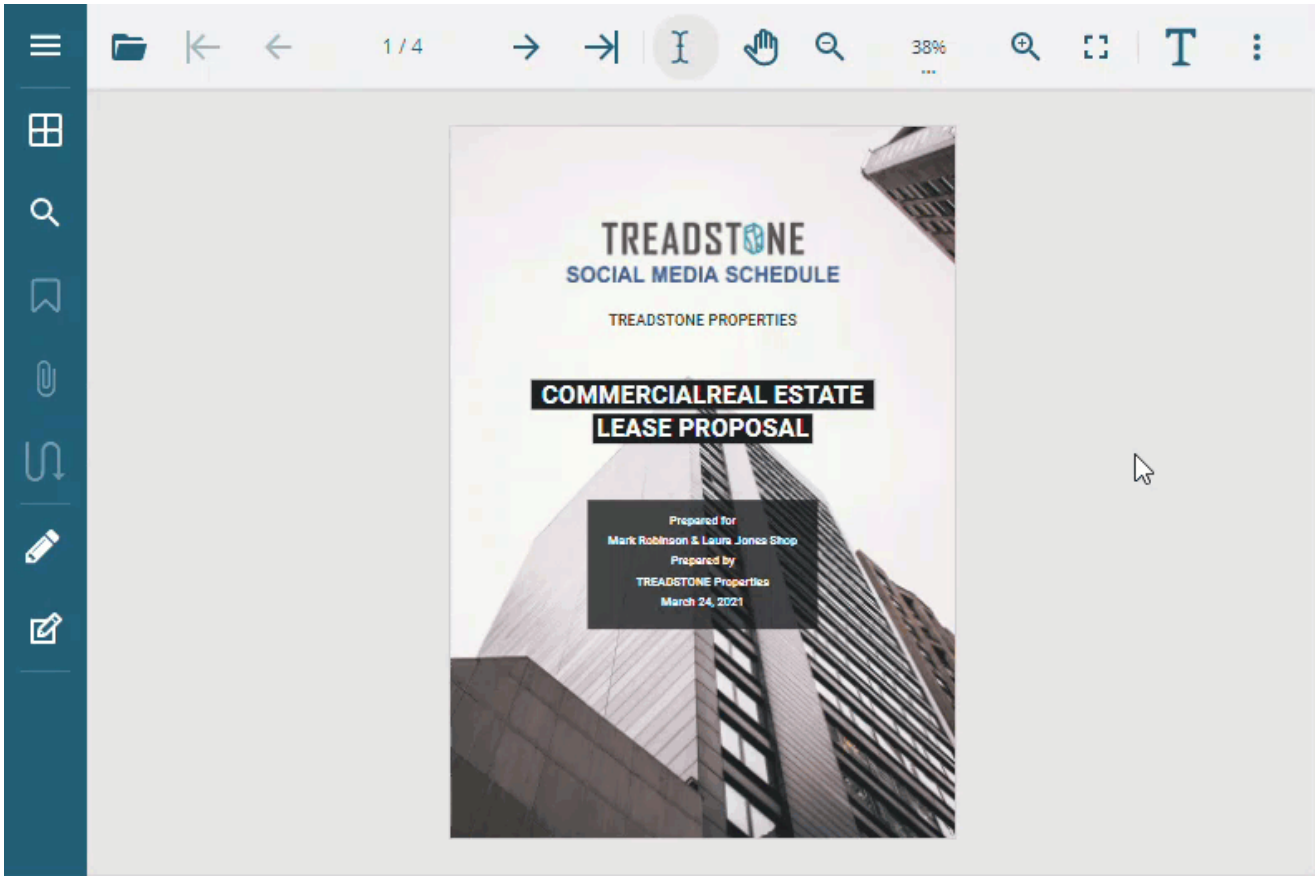


DsPdfViewer also allows a user to reorder a PDF document using code. Refer to the following example code to reorder a PDF document:

```
await viewer.save("test_changed_order.pdf", { pages: "3, 2, 1, 0" });
```

Split PDF Document

PDF Organizer allows a user to split a PDF document into different PDF documents by modifying the page number or page range in the input field.



DsPdfViewer also allows a user to split a PDF document by page numbers or size using code. Refer to the following example code to split a PDF document by page numbers:

```
await viewer.save("part1.pdf", { pages: "0-3" });
await viewer.save("part2.pdf", { pages: "4-8" });
await viewer.save("part3.pdf", { pages: "9-11" });
```

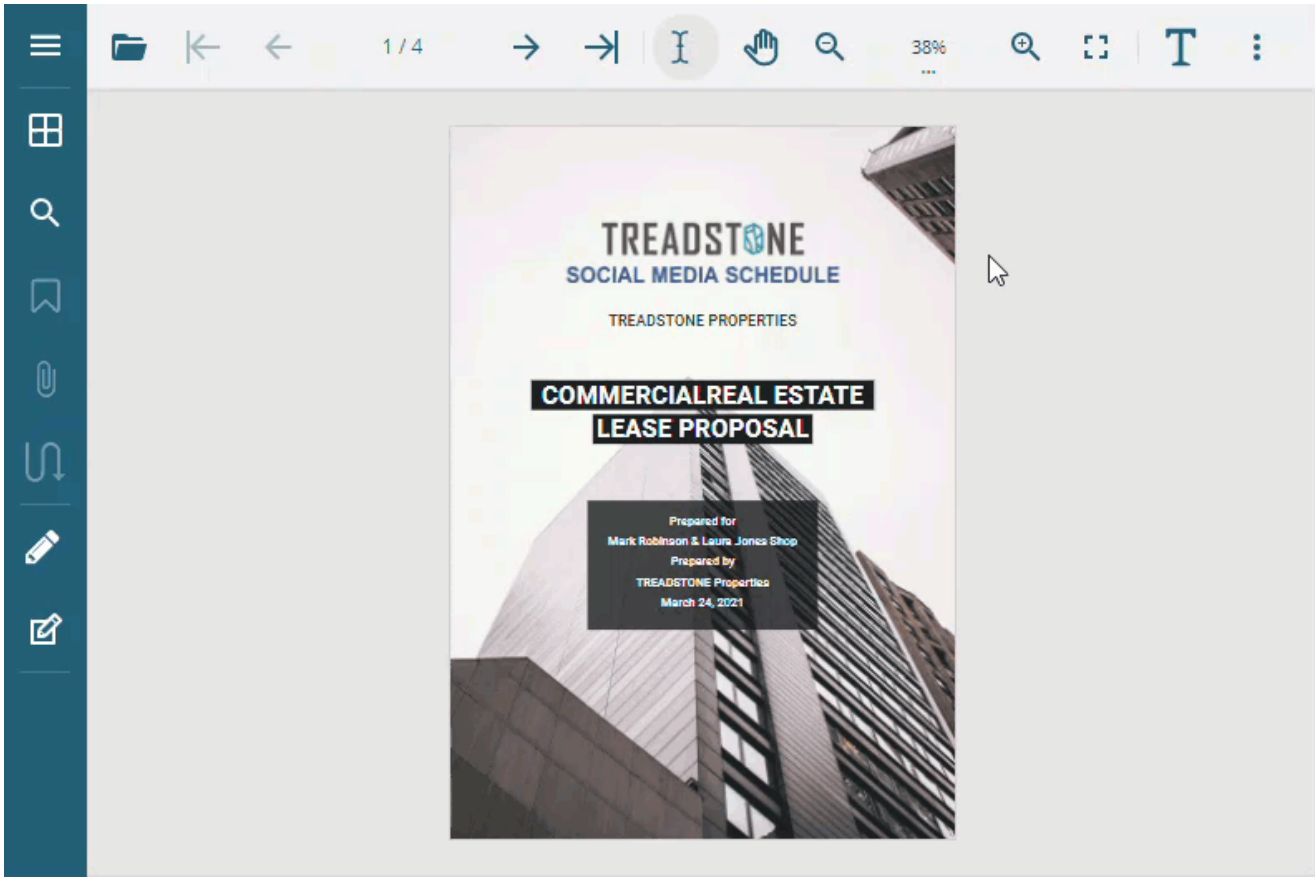
Refer to the following example code to split a PDF document by size:

```
async function splitPdfBySize(partSizeKb)
{
    const fileSizeBytes = await viewer.fileSize, fileSizeKb = fileSizeBytes / 1024,
          partCount = Math.floor(fileSizeKb / partSizeKb), pageCount = viewer.pageCount;
    if (pageCount > partCount)
    {
        let nextPageIndex = 0;
        for (let i = 0; i < partCount; i++)
        {
            const startIndex = nextPageIndex ? (nextPageIndex + 1) : 0;
            nextPageIndex = i === partCount - 1 ? pageCount - 1 : startIndex +
Math.floor(pageCount / partCount);
            await viewer.save(`part -${ i + 1}.pdf`, { pages: `${ startIndex} -${
nextPageIndex}` });
        }
    }
    else {
          await viewer.save("all-in-one.pdf");
```

```
        }
}
// Split the PDF by approximately 300 KB.
await splitPdfBySize(300);
```

Merge PDF Documents

PDF Organizer allows a user to merge an external PDF document with the PDF document opened in the DsPdfViewer. The user can reorder the pages or select the number of pages to be added to the new PDF document.



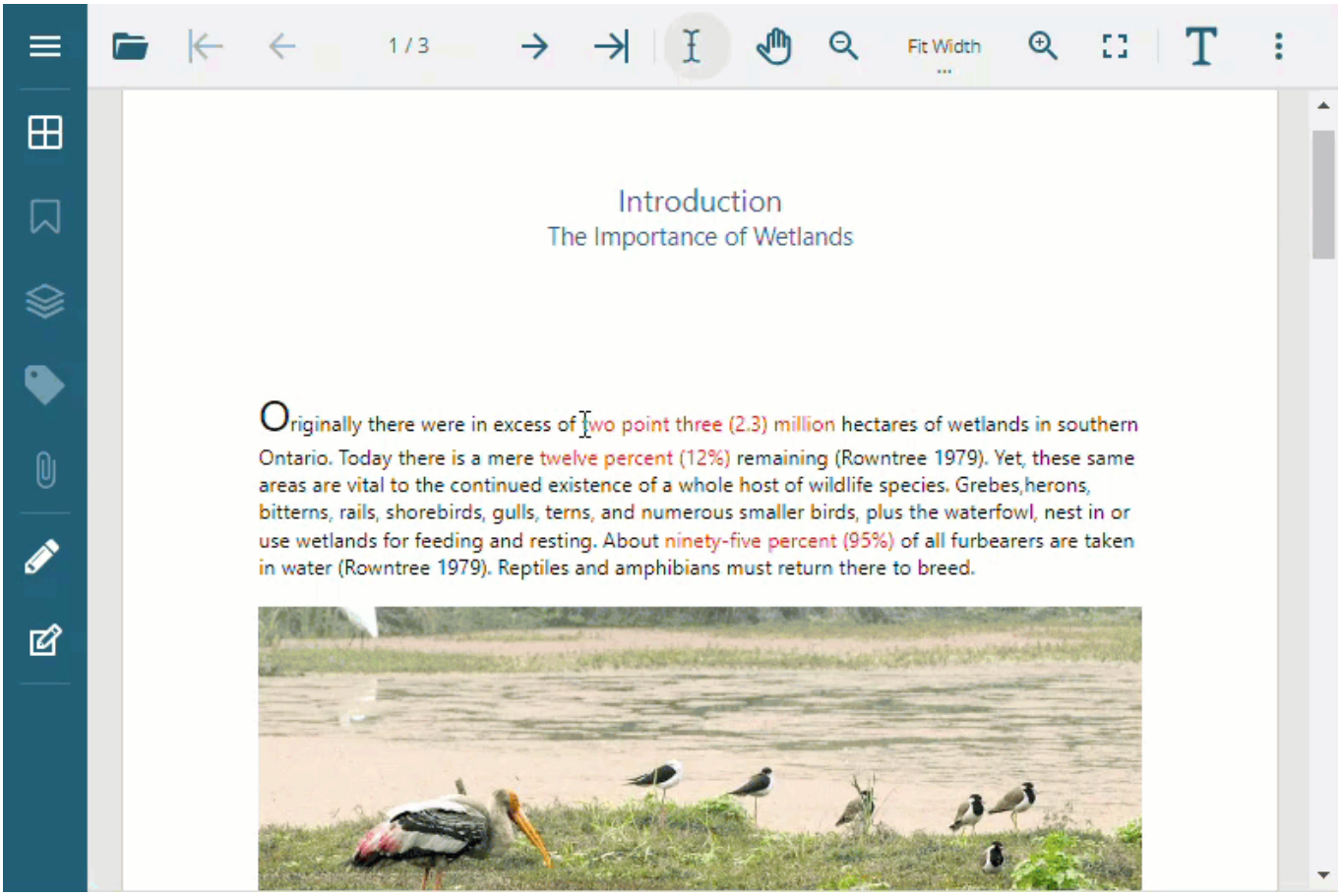
DsPdfViewer also allows a user to merge PDF documents using code. Refer to the following example code to merge PDF documents:

```
// Merge the first page from an external PDF document to the end of the active PDF
document.
const fileId = "uniquefileid1";
viewer.storage.setItem(fileId, new Uint8Array(fileReader.result));
await viewer.save("test_part10.pdf", { pages: `0 -${ viewer.pageCount - 1}, [file:${
fileId}]0` });
```

> **Note:** SupportApi should be configured in order to edit a PDF document using PDF Organizer button, and SupportApi is only available with the Professional License of DsPdfViewer. For more information, see Configure PDF Editor.

Comments

DsPdfViewer allows you to add comments to the PDF document in the form of annotations. These comments are useful for discussion, asking questions, or adding information. You can also use the **Comment Panel** to add replies to the comments, view all comments, edit or delete comments, and assign their review status.



Add Comments

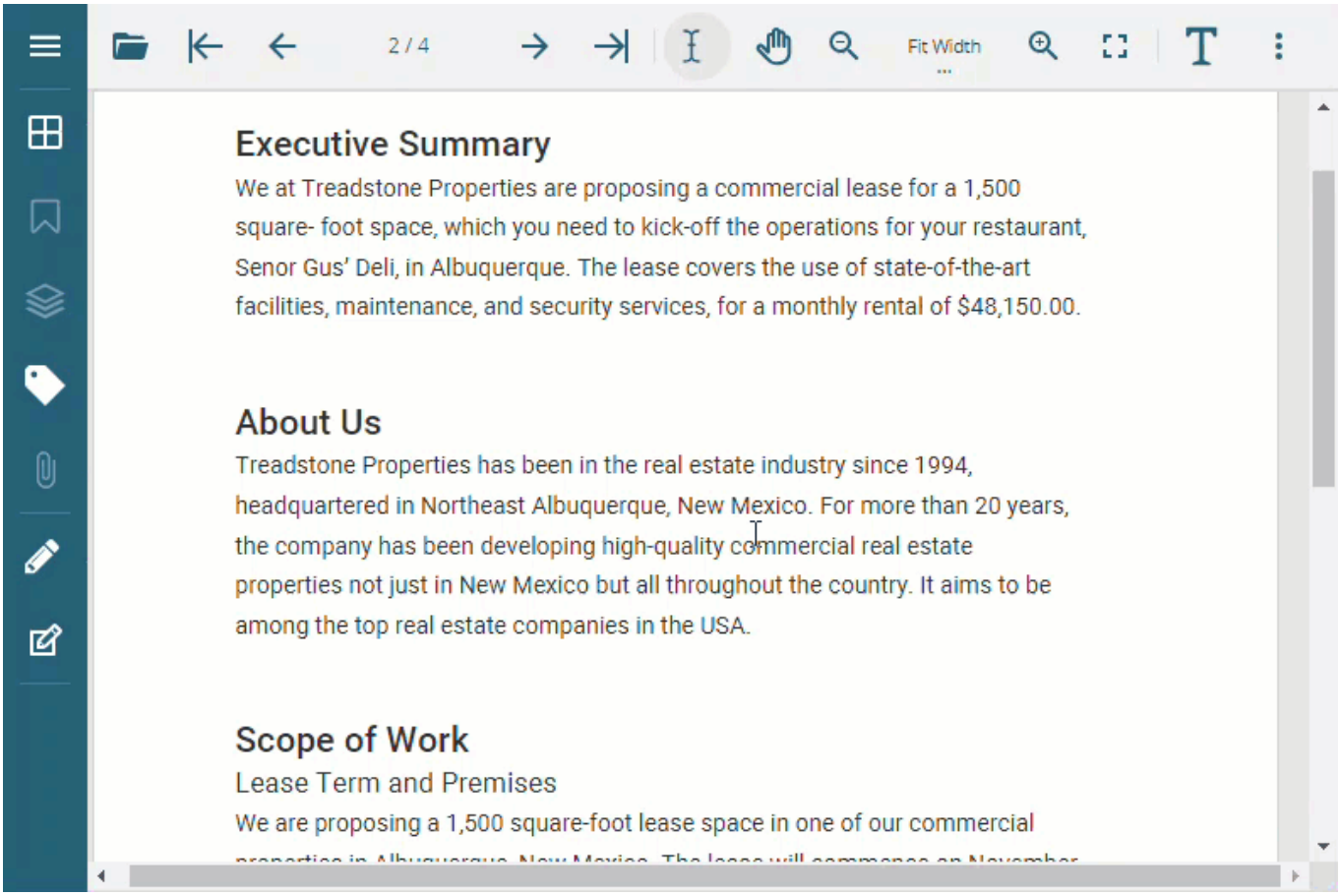
You can add comments to the PDF document directly through the context menu of DsPdfViewer. The context menu enables you to either add a sticky note or add a comment to the selection. These options are available in the context menu by default. You can also view, edit, or delete the newly added comments through the comment panel. The comment panel automatically activates when you add a comment, focusing on the new comment in the comment list. For more information, see **Comment Panel** section.

> **Note:** The context menu can be activated automatically upon text selection. For more information, refer to Customize Context Menu Activation on Text Selection.

Add Comment to Selection

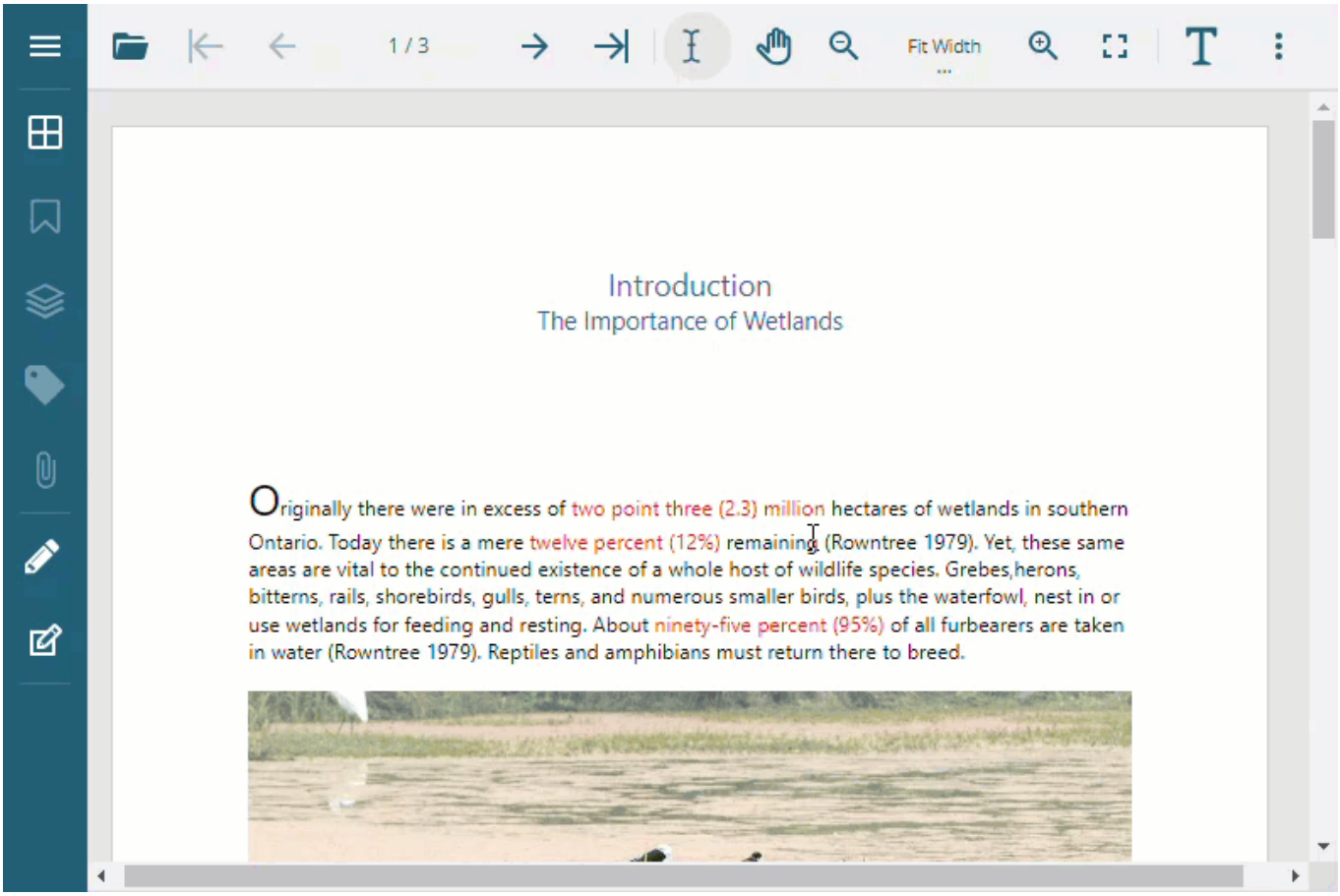
You can add a comment to a specific text selection by using **Add comment** option from the context menu.

Refer to the following GIF image to add a comment to the selection:



Add Sticky Note to PDF Document

You can add a sticky note anywhere in the PDF document directly using **Add sticky note** option in the context menu. Refer to the following GIF image to add a sticky note to the PDF document:



You can also add sticky notes using the quick edit toolbar in Text Tools. For more information, refer to Text Tools.

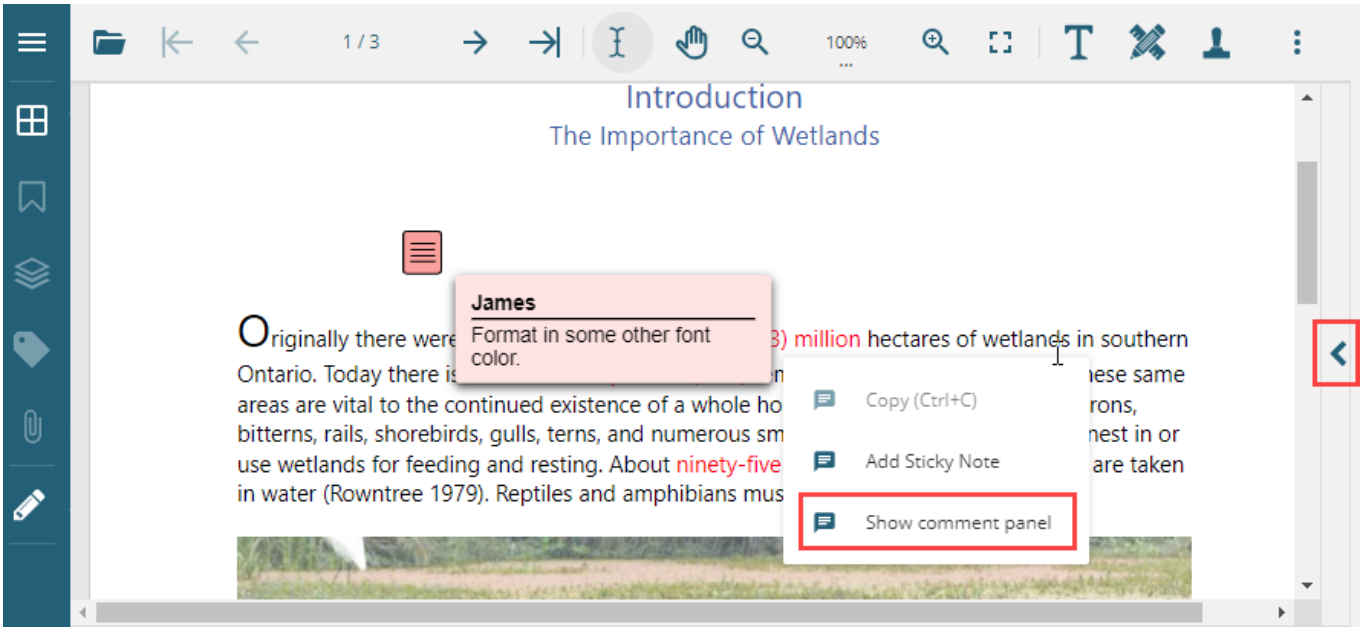
Note: Add sticky note option will be disabled in the context menu if you have selected any text.

Comment Panel

Comment Panel provides various functionalities to interact with comments with ease, allowing you to communicate with other people who are responsible for creating the PDF document. You can add replies to the comments, set the author name, modify or delete a comment, and assign review status. For more information, see **Work with Comments using Comment Panel** section.

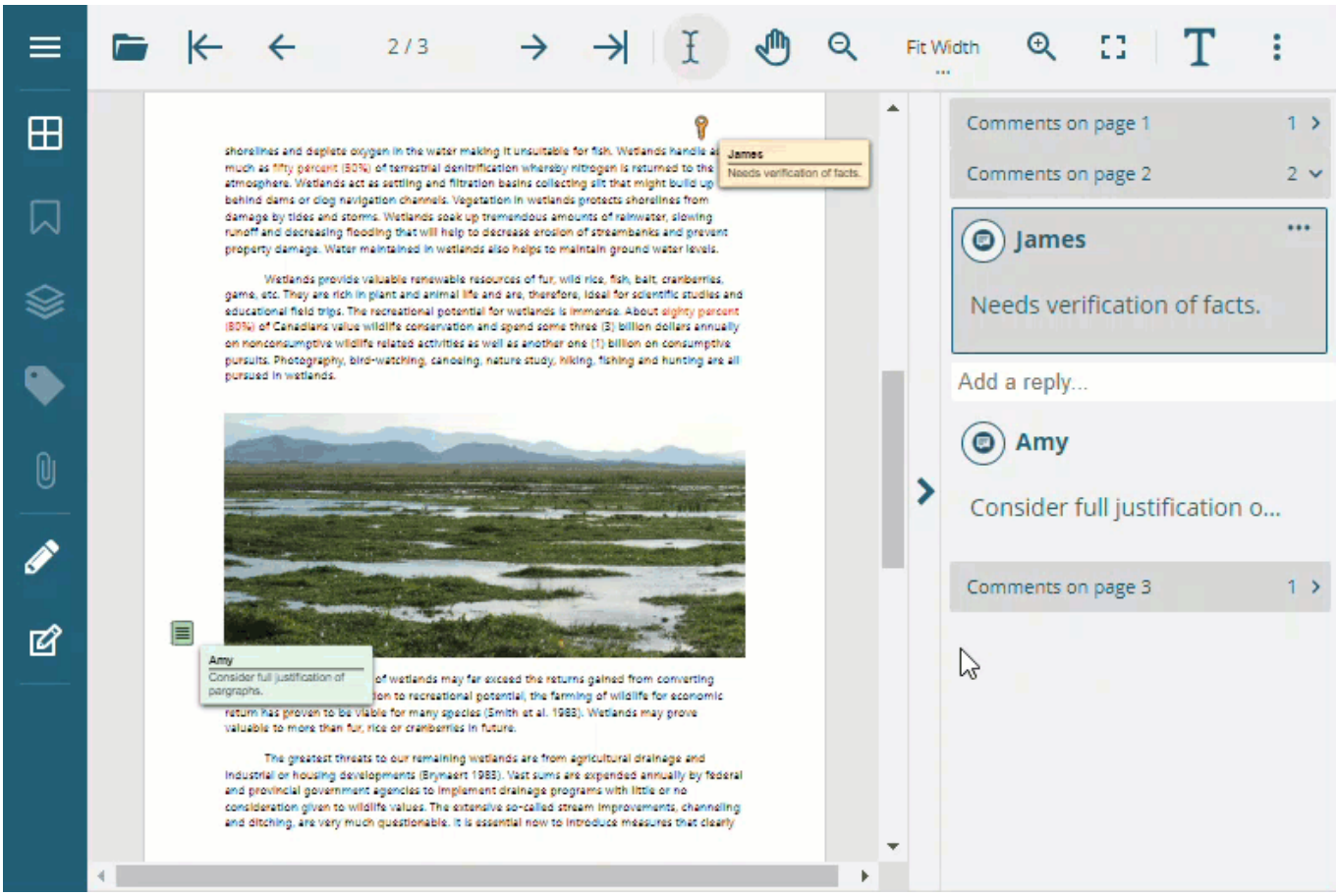
The comment panel automatically activates when you add a sticky note or a comment, focusing on the new comment in the comment list. Once it is open, you can navigate through the comments using the **Tab** button. You can also open the comment panel through the UI using the following methods:

- Right-click on the page and select 'Show comment panel' from its context menu.
- Click on the arrow at the extreme right of DsPdfViewer.



DsPdfViewer also allows you to resize the comment panel. To resize the comment panel:

- Place your cursor at the divider of the comment panel until you see the resize cursor.
- Left-click the mouse button and then move the pointer to the left or right to widen or narrow the sidebar width.



Enable Comments Panel Programmatically

The Comments Panel is hidden by default. However, you can also enable the comment panel programmatically 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

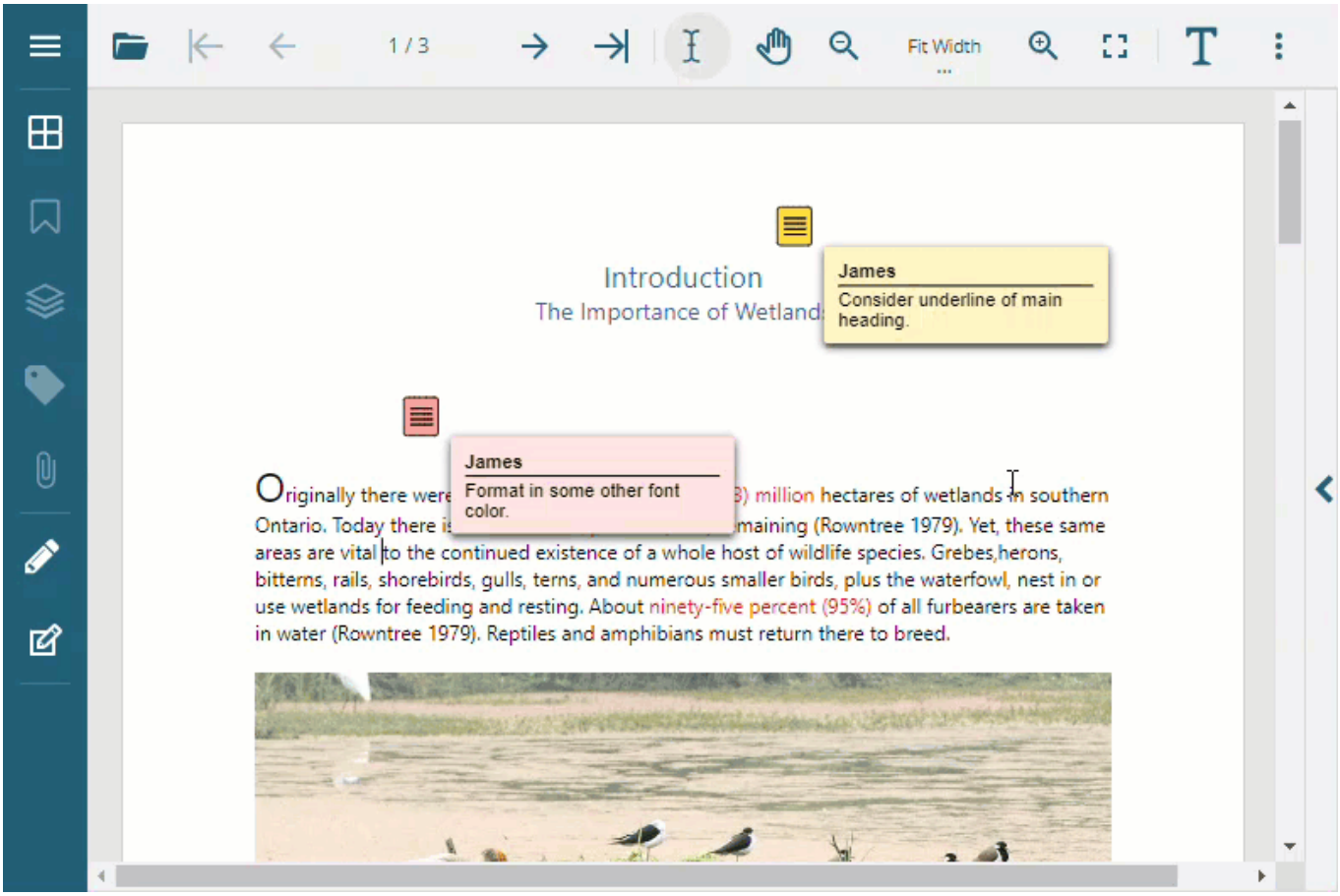
```
viewer.addReplyTool("expanded");
```

Note: To use Comments Panel, 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.

Work with Comments using Comment Panel

Add Reply to Comments

You can add a reply to a comment in the comments panel by clicking on **Reply**.

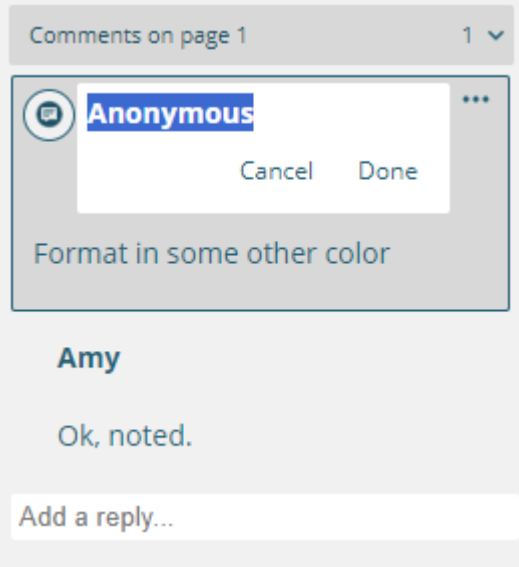


Set Author Name for Comments

While replying to a comment, the author name is displayed as 'Anonymous' by default. However, the desired author

name can be set in following ways:

- 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:

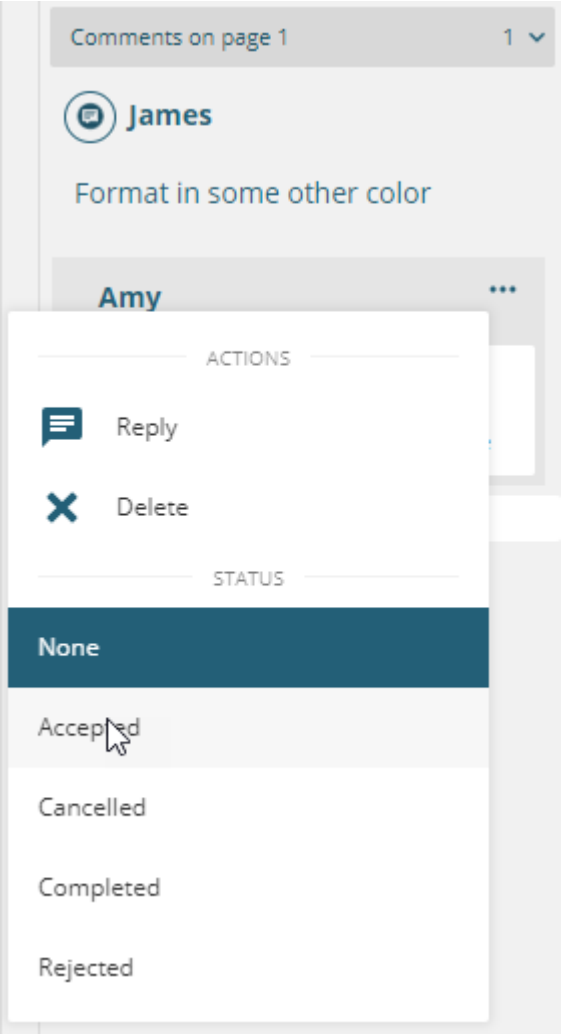
Index.cshtml

```
var viewer = new DsPdfViewer("#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.

Add Review Status to Comments

You can also add review status to a comment 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.

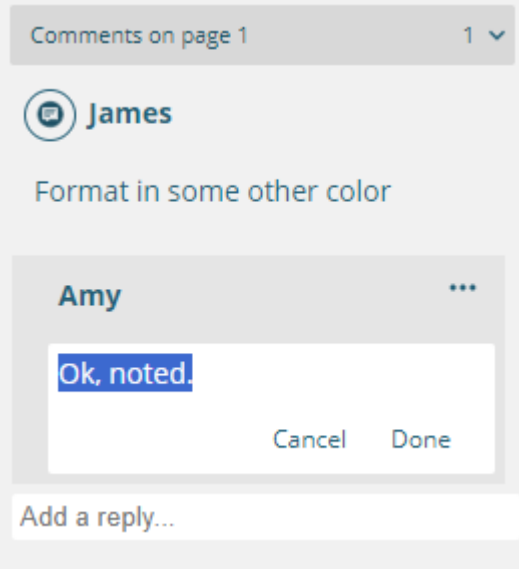
You can also assign status for a text annotation programmatically, by setting following properties: 'title' 'state', 'stateModel', 'referenceType', 'referenceAnnotationId'. The below example code shows how to add a reply to text annotation with id '6R' and assign 'Completed' status:

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);
    });
}
```

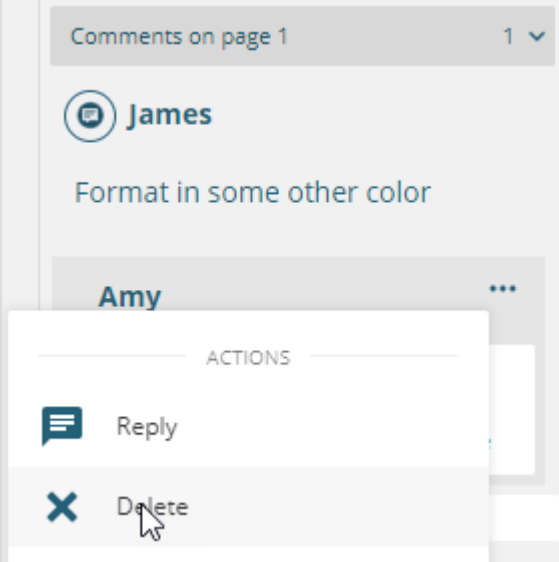
Modify or Delete Comments

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



The comment panel also highlights the comment on the PDF page when you reply to the comment in the comment list.

A comment or reply can be deleted by clicking on **Actions** | **Delete** or by using the **Delete** key.



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

Customize Comment Panel

You can also programmatically customize the behavior of the comment panel. Refer to the following example codes for different customizations:

Example 1: Disable Auto-Activation of Comment Panel

```
JavaScript
```

```
var viewer = new DsPdfViewer("#root", { replyTool: { autoExpandOnCommentAdd: false
}});
```

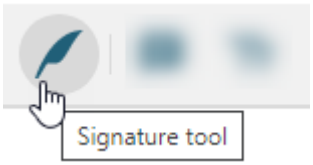
Example 2: Enable Comment Panel Icon Color

JavaScript

```
var viewer = new DsPdfViewer("#root", { replyTool: { useColoredIcons: true }});
```

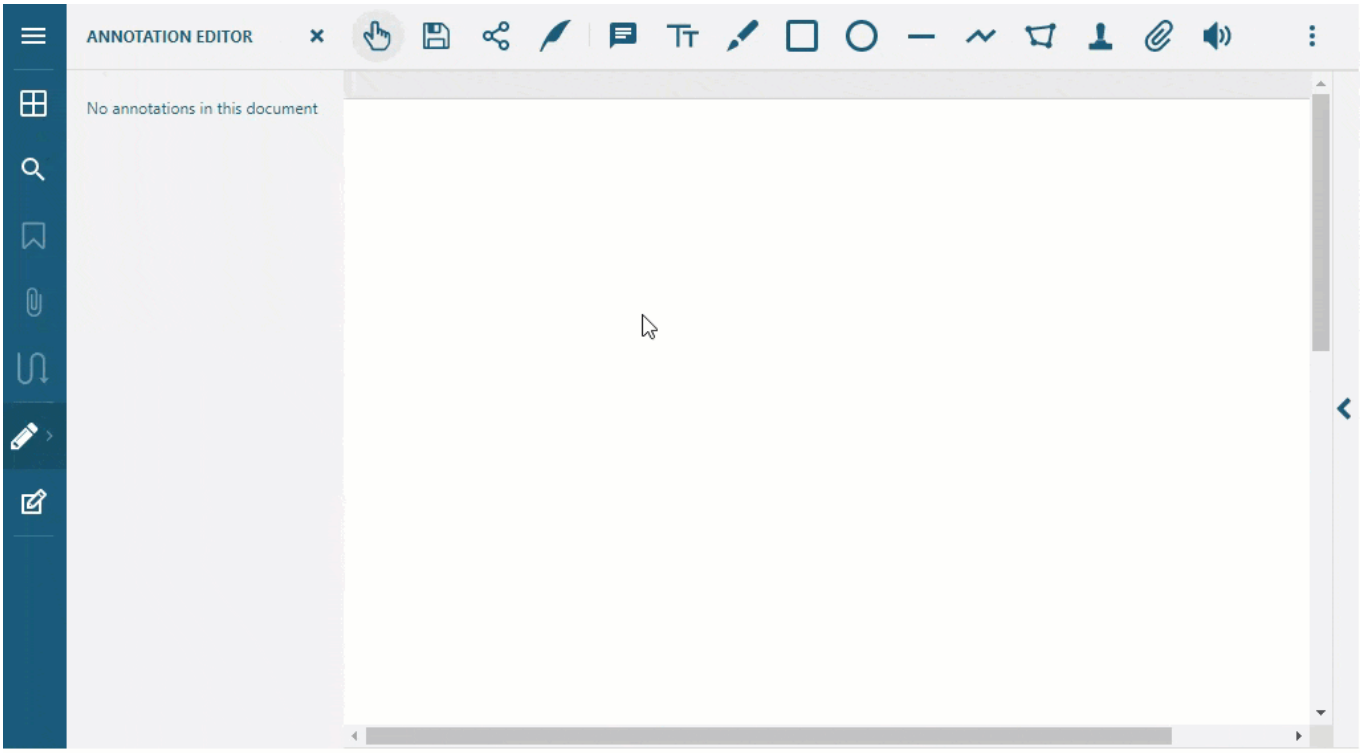
Graphical Signature Tool

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

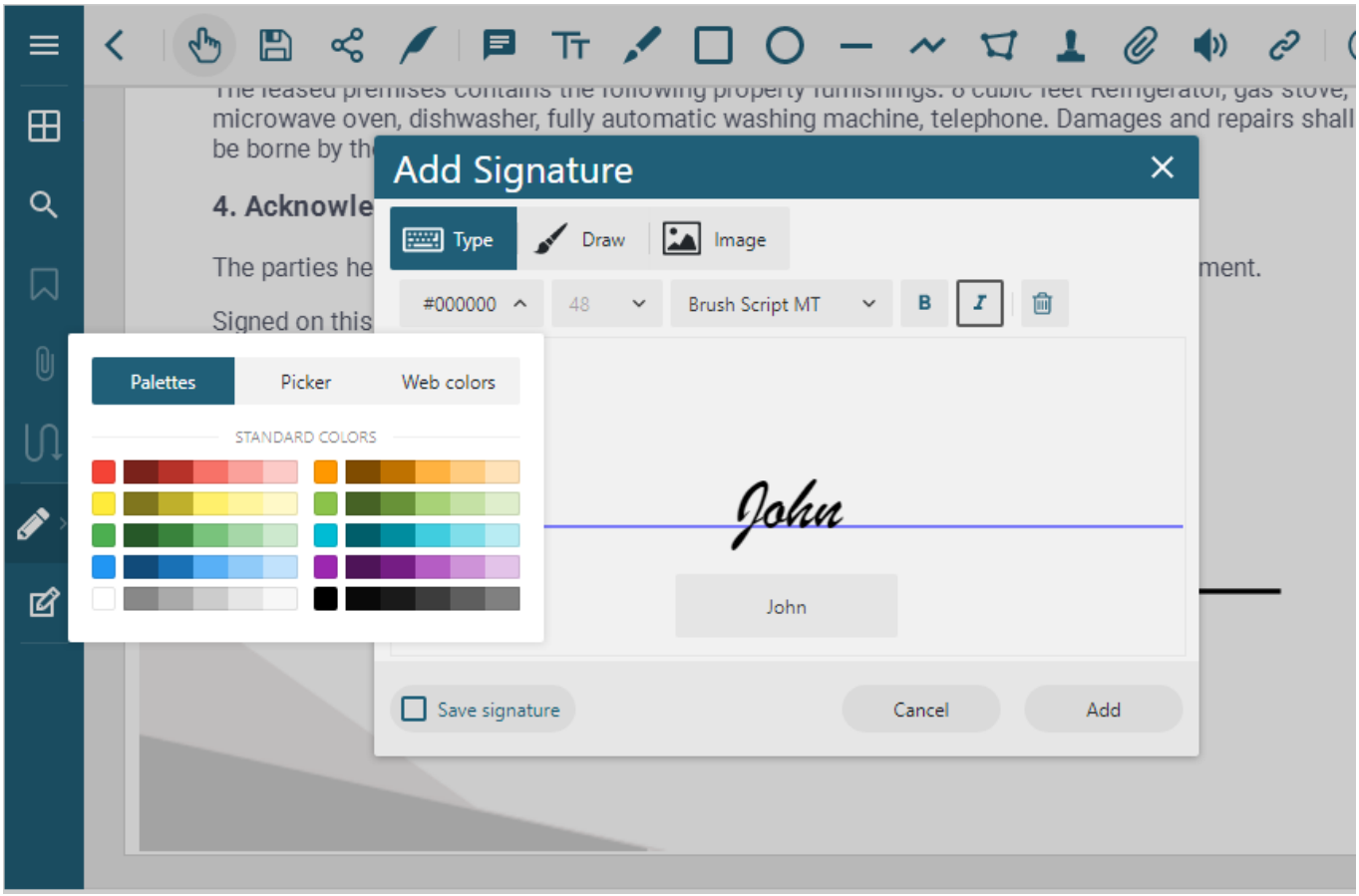


The 'Signature Tool' button opens the 'Add Signature' dialog which lets you type, draw or add the image of a signature in a PDF document. You can also format the signatures by using formatting options while typing or drawing signatures. If location of the signature is not specified, you can move the digital signature to a desired location with the help of a mouse.

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 docment.

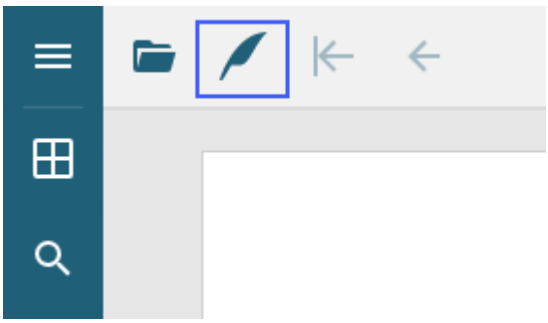
Note: SupportApi should be configured in order to use Signature Tool, as the editing operation is performed in a PDF document.

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.

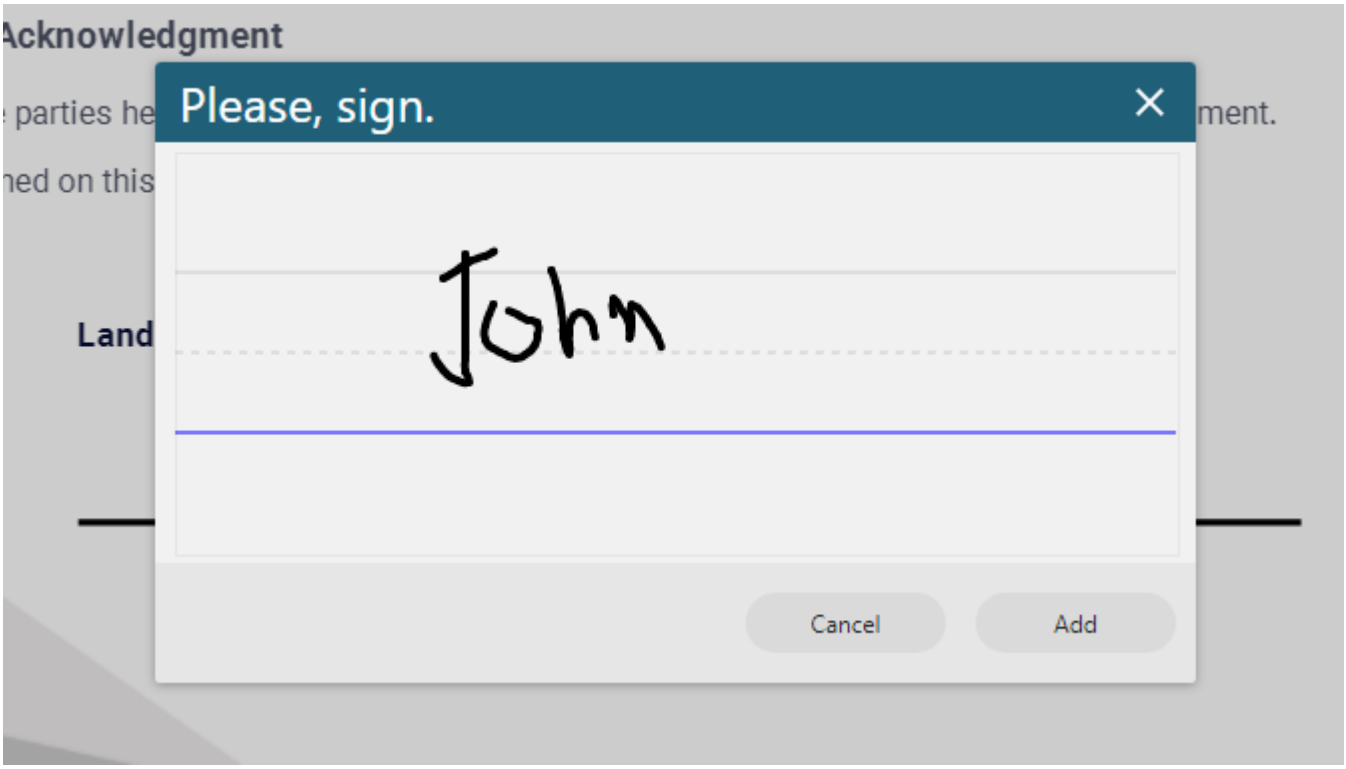
Note: The saved signature data is owned by the active user and can be set by using the **currentUserName** property or **userName** option. Therefore, if these properties are changed, saved signature data will also change.

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 DsPdfViewer demos.

Add Signature from Server

You can also load and add a graphical signature to a PDF document from the server using **addStamp** method of **DsPdfViewer** class. The addStamp method is asynchronous, so it is possible to perform the "viewer.addStamp" operation concurrently if you await the async code. Refer to the following example code for the same:

JavaScript

```
function addStampFromUrl(imageUrl, viewer){
  // Define graphical signature parameters.
  fetch(imageUrl)
  .then(response => response.blob())
  .then(blob => blob.arrayBuffer())
  .then(arrayBuffer => {
        const fileId = new Date().getTime() + ".png";
        const fileName = fileId;
        const pageIndex = 0;
        const imageData = new Uint8Array(arrayBuffer);
        const rect = [0, 0, 200, 200];

        viewer.storage.setItem(fileId, imageData);

        // Add graphical signature to the PDF document.
        viewer.addStamp(
                    imageData,
```

```
                    {
                        fileId,
                        fileName,
                        pageIndex,
                        rect,
                        select: false,
                        subject: "",
                        rotate: 0,
                        convertToContent: false

                    });
  });
}
addStampFromUrl("https://i.imgur.com/signature.png", viewer, 0);
```

Limitations

addStamp method does not support the SVG image format. You need to convert the image to another supported format.

Digital Signature

A digital signature is an electronic, encrypted, stamp of authentication on PDF documents to secure the authenticity of the content. The signature confirms that the information was created by the signer and has not been tampered with. DsPdfViewer enables a user to digitally sign a PDF document and save it to the server using an optional save settings parameter **sign** of **SignatureInfo** type in the client-side **save** method. This method is used to specify

signature settings to digitally sign a PDF document. The signing is implemented on the server side.

The client-side save method sends a request to the server to save the document with a signature. The server side signing is implemented using the following two events: **VerifyAuthToken** and **OnSign**. The VerifyAuthToken validates the authentication token, while the onSign event adds the signature to the PDF document after verifying it with the certificate.

Refer to the example code below, which implements server side signing by binding the events and defining the event handlers in the Startup.cs file:

```
public class Startup {

    const int MAX_MESSAGE_SIZE = 268435456/*256MB*/;

    static Startup() {
        Licensing.AddLicenseKeys();
    }

    // Configure the DsPdfViewer.
    public void Configuration(IAppBuilder app) {
        app.UseCors(CorsOptions.AllowAll);
        GcPdfViewerHub.Configure(app);
        GcPdfViewerController.Settings.VerifyToken += VerifyAuthToken;
        GcPdfViewerController.Settings.Sign += OnSign;

    }

    // Validate the authentication token provided during DsPdfViewer initialization.
    private void VerifyAuthToken(object sender, VerifyTokenEventArgs e) {
            string token = e.Token;
        if (string.IsNullOrEmpty(token) || !token.StartsWith("support-api-demo")) {
            e.Reject = true;
        }
    }

    // Implement PDF document server side signing.
    private void OnSign(object sender, SignEventArgs e) {
        var signatureProperties = e.SignatureProperties;

            string projectRootPath = HttpContext.Current.Server.MapPath("~");
            string certificatePath = Path.Combine(projectRootPath,
"assets/DsPdfTest.pfx");
        byte[] certificateBytes = File.ReadAllBytes(certificatePath);
        if (certificateBytes == null) {
            e.Reject = true;
            return;
        }
            X509Certificate2 certificate = new X509Certificate2(
            certificateBytes, "qq", X509KeyStorageFlags.MachineKeySet |
X509KeyStorageFlags.PersistKeySet | X509KeyStorageFlags.Exportable);
        signatureProperties.SignatureBuilder = new
GrapeCity.Documents.Pdf.Pkcs7SignatureBuilder()
        {
```

```
            CertificateChain = new X509Certificate2[] { certificate },
            HashAlgorithm =
GrapeCity.Documents.Pdf.Security.OID.HashAlgorithms.SHA512,
                Format =
GrapeCity.Documents.Pdf.Pkcs7SignatureBuilder.SignatureFormat.adbe_pkcs7_detached
        };

            string fontPath = Path.Combine(projectRootPath, "assets/arialuni.ttf");
        signatureProperties.SignatureAppearance = new SignatureAppearance
        {
            TextFormat = new TextFormat
            {
                Font = Font.FromFile(fontPath),
                    FontSize = 5.5f
            }
        };
    }

}
```

Refer to the following example code to digitally sign a PDF document:

```
// Specify the signature settings.
viewer.save(undefined, { sign: { signatureField: "Employee", signerName: "Employee
Name", location: locationName }, reload: true });
```

DsPdfViewer's save method also provides an optional parameter **saveMode** which is used to save a PDF document using incremental update or as linearized.

Refer to the following example code to save a PDF document using incremental update:

```
viewer.save("IncrementalUpdate.pdf", { saveMode: "IncrementalUpdate" });
```

For more information on how to add a digital signature, see Sign document using digital signature.

> **Note:** SupportApi should be configured in order to digitally sign a PDF document, and it is only available with the Professional License of DsPdfViewer. For more information, see Configure PDF Editor.

Get Signature Information

DsPdfViewer enables a user to check if a PDF document is digitally signed. DsPdfViewer provides a **getSignatureInfo** client-side method to acquire all the information about the signatures available in the PDF document. The following example codes depict the various scenarios in which this method can be used.

Refer to the following example code to acquire an array of all signature fields available in the PDF document, irrespective of whether they are signed or not:

```
const signatureInfo = await viewer.getSignatureInfo();
```

Refer to the following example code to check if a PDF document is signed:

```
const signatureInfo = await DsPdfViewer.findControl("#viewer").getSignatureInfo();
console.log(signatureInfo.signed ?
    "The document is signed with digital signature." : "The document is not signed");
```

Refer to the following example code to iterate through all the signature fields available in the PDF document:

```
const info = await DsPdfViewer.findControl("#viewer").getSignatureInfo();
for (const field of info.signatureFields) {
    console.log("Field '" + field.fieldName + "', has signature value: " +
!!field.signatureValue);
}
```

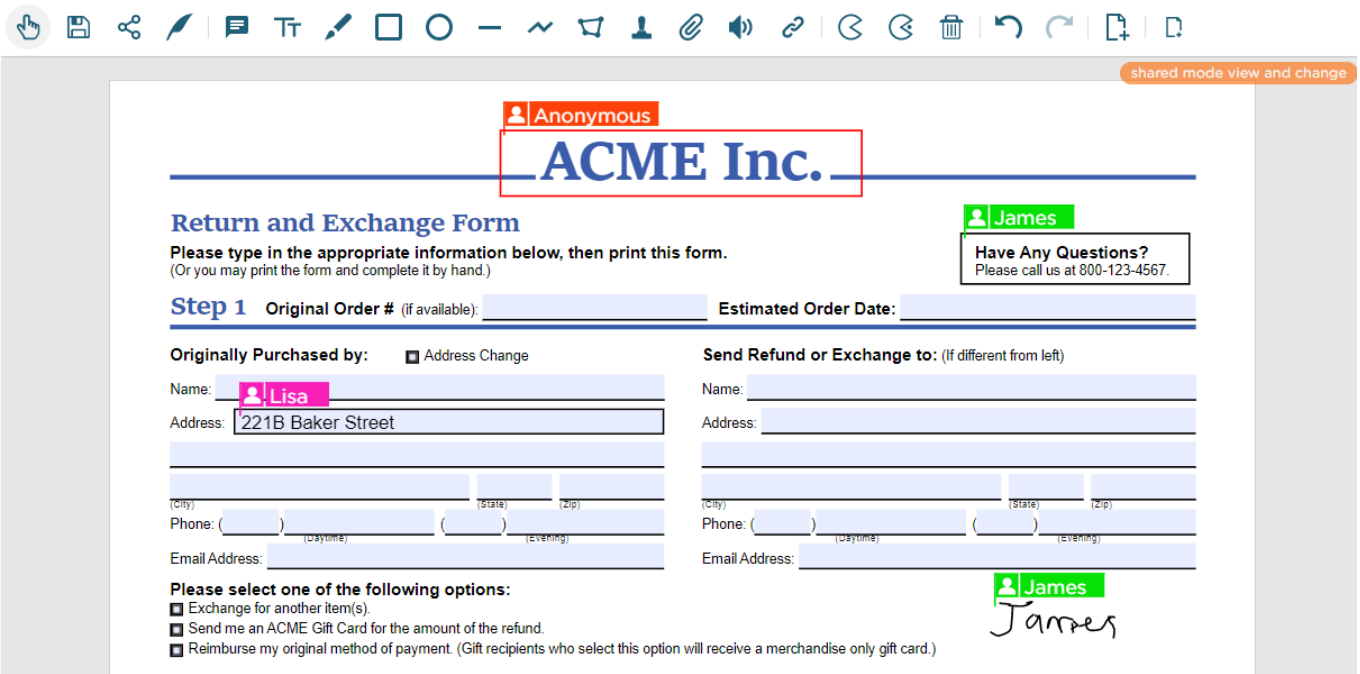
Refer to the following example code to show information about all signatures used in the PDF document:

```
var viewer = DsPdfViewer.findControl("#root");
const signatureInfo = await viewer.getSignatureInfo();
if (signatureInfo.signed) {
    let s = "";
    for (let i = 0; i < signatureInfo.signedByFields.length; i++) {
        const signedField = signatureInfo.signedByFields[i], signatureValue =
signedField.signatureValue;
        const signerName = signatureValue.name, location = signatureValue.location,
            signDate =
viewer.pdfStringToDate(signatureValue.modificationDate).toDateString();
        s += `${i + 1}. Signed by field: ${signedField.fieldName}, signer name:
${signerName}, location: ${location}, sign date: ${signDate}\n`;
    }
    alert("The document was signed using digital signature:\n" + s);
} else {
    alert("The document is not signed.");
}
```

Share and Collaborate

DsPdfViewer 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.



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. The steps listed below describe how to set up collaboration mode in ASP.NET Core Web Application and ASP.NET WebForms Application:

Web Application in .NET 6/ .NET 7

To support persistent client connections for collaboration mode, configure your web application by editing Program.cs file following the steps mentioned below:

1. Add SignalR service to the services collection by calling "SupportApi.Connection.GcPdfViewerHub.ConfigureServices(services);".
   GcPdfViewerHub.ConfigureServices(builder.Services);
2. Map SignalR hub.
   // Configure signalr hub for collaboration mode.

   IApplicationBuilder applicationBuilder = app.UseEndpoints(endpoints =>

   {

   endpoints.MapControllers();

```
        endpoints.MapHub("/signalr", opts => {
            opts.TransportMaxBufferSize = 268435456L;
            opts.ApplicationMaxBufferSize = 268435456L;
        });
    });
```

Web Application in .NET Core 3.1/ .NET 5

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

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");
   });
}
```

WebForms Application

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

```
[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:

.NET 6/ .NET 7

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

Program.cs

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

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

Program.cs

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

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

Program.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);
                    }
                }
            });
        }
}
```

.NET Core 3.1/ .NET 5

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

startup.cs

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

2. 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);
```

3. 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

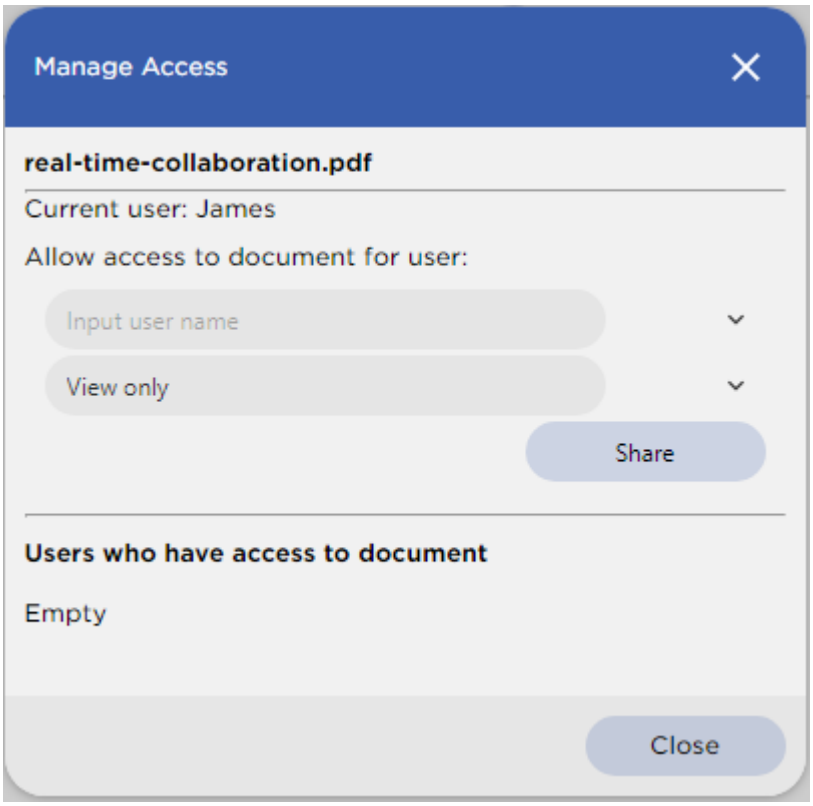
You can share a PDF document by clicking on the 'share' button in DsPdfViewer's toolbar.

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

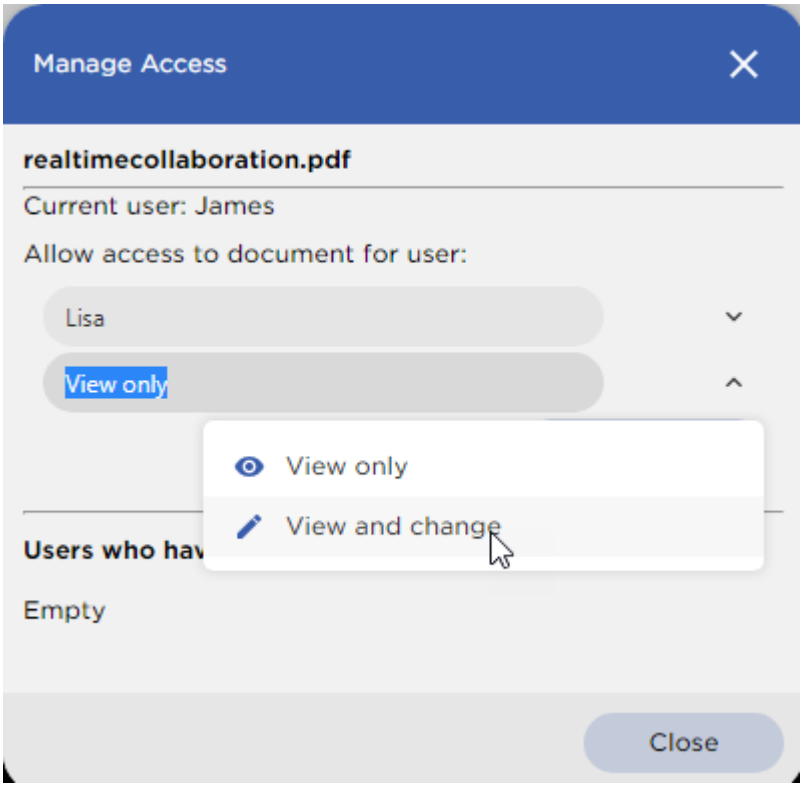
Index.cshtml

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

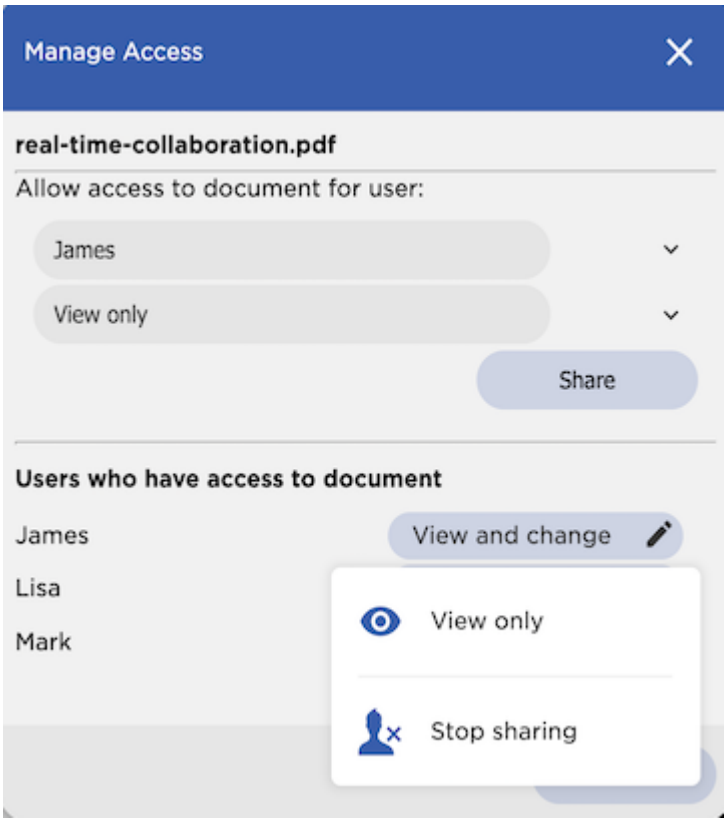
When you click on 'share' button, the 'Manage Access' dialog box opens as shown below:



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 DsPdfViewer("#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:

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 DsPdfViewer("#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 DsPdfViewer("#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 DsPdfViewer("#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 DsPdfViewer("#root", {
    sharing: {
        presenceMode: 'others'
    },
    supportApi: "api/pdf-viewer"
});
```

UI Customizations

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

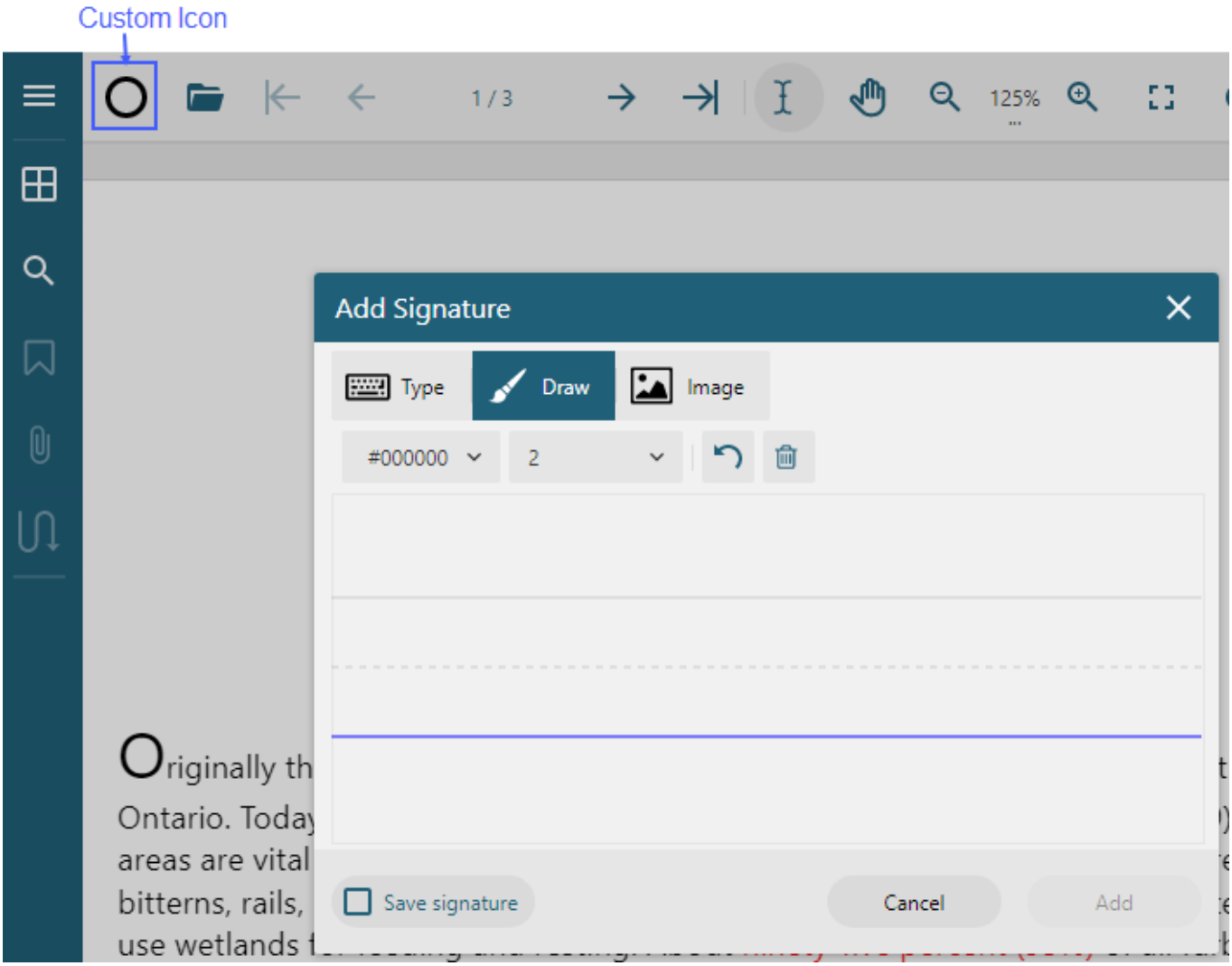
Add a Custom Toolbar Item using SVG Icon

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
12s5.373 12 12 12 12-5.373 12-12-5.373-12-12-12zM12 21c-4.971 0-9-4.029-9-9s4.029-9
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 DsPdfViewer 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']);
```

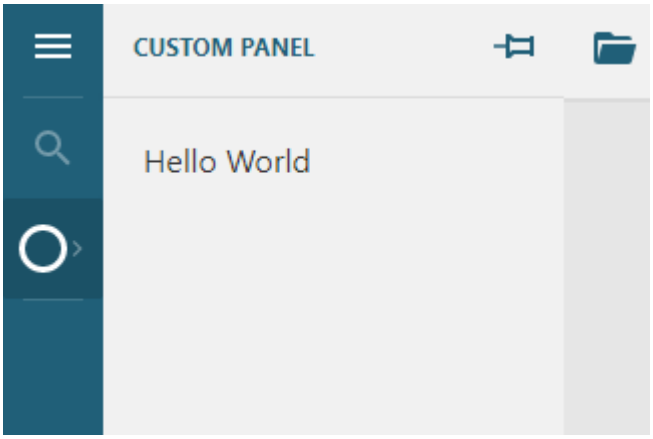
Custom Sidebar Panel

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) {
        //DsPdfViewer.LicenseKey = 'your_license_key';
        var viewer = new DsPdfViewer(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",
```

```
            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 12-12-5.373-12-12zM12 21c-4.971 0-9-4.029-9-9s4.029-9 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 DsPdfViewer, you can add the document list panel to the left sidebar, which helps the user view the list of PDF documents and open any of them by clicking on it. **addDocumentListPanel** method of DsPdfViewer class adds the document list panel to the sidebar, and the **documentListUrl** property of **ViewerOptions** class defines the list of available PDF documents to display in the document list panel.

documentListUrl property either accepts a URL to the document list service, which returns a JSON string with an array of available documents, or a Data URI. It also allows the user to set a predefined list of documents defined using the **name**, **path**, and **title** properties of **DocumentListItem** type. The following example codes depict each of these scenarios.

Bind to API Service

To create an API service that returns a list of available PDF documents that you want to add to the document list panel, add the below sample code in the Controller class:

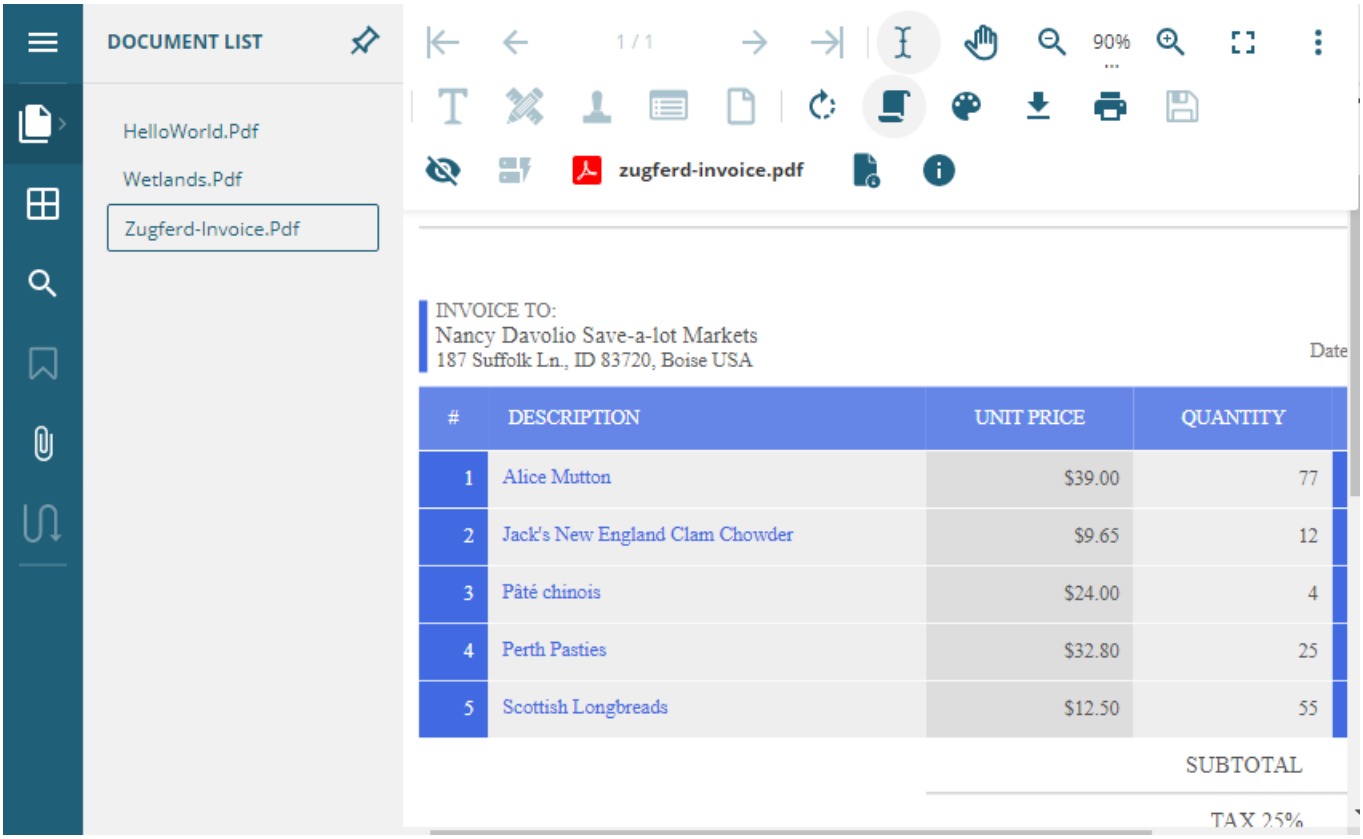
SupportApiController.cs

```
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)
    {
        string filePath = Path.Combine("Resources", "PDFs", name);
        Response.Headers["Content-Disposition"] = $"inline; filename=\"{name}\"";
        return new FileStreamResult(System.IO.File.OpenRead(filePath),
"application/pdf");
    }

    //This method is used by the Document List Panel sample.
    [Route("DocumentList")]
    public IActionResult DocumentList()
    {
        string dirPath = Path.Combine("Resources", "PDFs");
        var directoryInfo = new DirectoryInfo(dirPath);
        var allPdfs = directoryInfo.GetFiles("*.pdf");
        return new JsonResult(allPdfs.Select(
            f_ => new {
                path = $"api/pdf-viewer/get-pdf-from-list?name={f_.Name}",
                name = f_.Name,
                title = $"Open {f_.Name}",
            }));
    }
}
```

To add the document list panel to 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 viewer = new DsPdfViewer("#root", { documentListUrl: "api/pdf-
viewer/DocumentList" });
    viewer.addDefaultPanels();
    viewer.addDocumentListPanel();
}
```



Bind to External Service

The following example code loads the document list from an external service and adds the document list panel to the viewer:

JavaScript

```
// Load document list from external service.
var viewer = new DsPdfViewer("#root", { documentListUrl:
"http://localhost:5000/documents.json" } );
viewer.addDocumentListPanel();
```

Bind to Data URI

The following example code loads the document list from a data URI and adds the document list panel to the viewer:

JavaScript

```
// Load document list from data URI.
var viewer = new DsPdfViewer("#root", { documentListUrl: 'data:,[{"path":
"doc1.pdf"}, {"path": "doc2.pdf", "name": "Hello doc 2", "title": "title 2"}]' } );
viewer.addDocumentListPanel();
```

Bind to Pre-defined List

The following example code loads the document list with pre-defined documents and adds the document list panel to the sidebar:

JavaScript

```
var options = { };

// Define document list.
options.documentListUrl = [
  {
     path: "/sample1.pdf",
     name: "Open sample1.pdf",
     title: "Sample 1"
  },
  {
     path: "/sample2.pdf",
     name: "Open sample2.pdf",
     title: "Sample 2"
  },
  {
     path: "/sample3.pdf",
     name: "Open sample3.pdf",
     title: "Sample 3"
  }
];

// Initialize DsPdfViewer and add document list panel.
var viewer = new DsPdfViewer("#root", options);
viewer.addDocumentListPanel();
```

Customize Document List Appearance

You can also enhance the appearance of the Document List using HTML markup, which you can set through the **previewContent** property of DocumentListItem type. This property specifies custom HTML markup to represent the document list.

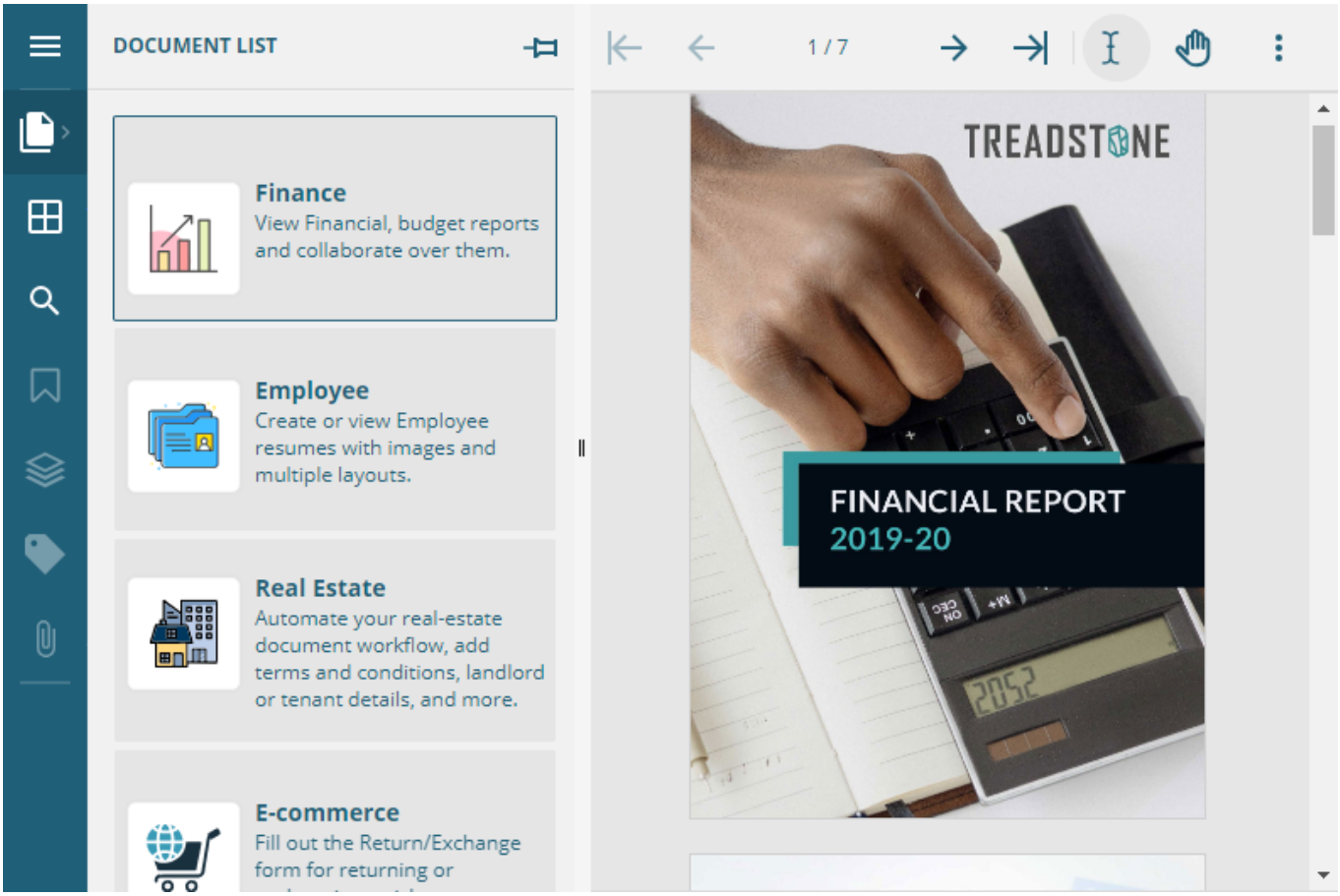
Refer to the following example code to load a pre-defined document list with custom preview content:

JavaScript

```
// Set onload function.
window.onload = function(){

    // Set base path for the PDF documents.
    const baseAssetsPath = "/documents-api-pdfviewer/demos/product-bundles/assets/";

    // Define the document list.
    options.documentListUrl = [

        // Set appearance of first PDF document.
        {
            path: baseAssetsPath + "pdf/documents-list/financial-report.pdf",
            title: "Finance",
            previewContent: renderPreviewCard("Finance",
                            "View Financial, budget reports and collaborate over
them.",
```

```
                            baseAssetsPath + "images/preview/svg/Finance.svg")
        },

        // Set appearance of second PDF document.
        {
            path: baseAssetsPath + "pdf/documents-list/resumeWithImage.pdf",
            title: "Employee",
            previewContent: renderPreviewCard("Employee",
                            "Create or view Employee resumes with images and multiple
layouts.",
                            baseAssetsPath + "images/preview/svg/Employee.svg")
        },

        // Set appearance of third PDF document.
        {
            path: baseAssetsPath + "pdf/documents-list/Real-Estate-Schedule-
Form.pdf",
            title: "Real Estate",
            previewContent: renderPreviewCard("Real Estate",
                            "Automate your real-estate document workflow, add terms
and conditions, landlord or tenant details, and more.",
                            baseAssetsPath + "images/preview/svg/Real-Estate.svg")
        },

        // Set appearance of fourth PDF document.
        {
            path: baseAssetsPath + "pdf/documents-list/Return_ExchangeForm.pdf",
            title: "E-commerce",
            previewContent: renderPreviewCard("E-commerce",
                            "Fill out the Return/Exchange form for returning or
exchanging articles at a store.",
                            baseAssetsPath + "images/preview/svg/E-commerce.svg")
        },

        // Set appearance of fifth PDF document.
        {
            path: baseAssetsPath + "pdf/documents-list/news.pdf",
            title: "Media",
            previewContent: renderPreviewCard("Media",
                            "View newsletters, media articles in multiple languages
and columnar formats.",
                            baseAssetsPath + "images/preview/svg/Media.svg")
        },

        // Set appearance of sixth PDF document.
        {
            path: baseAssetsPath + "pdf/documents-
list/patientmedicalhistoryform.pdf",
            title: "Healthcare",
            previewContent: renderPreviewCard("Healthcare",
                            "Fill out this confidential medical history form for a
```

```
patient.",
                            baseAssetsPath + "images/preview/svg/Healthcare.svg")
        },

        // Set appearance of seventh PDF document.
        {
            path: baseAssetsPath + "pdf/documents-list/house-plan-layers.pdf",
            title: "Architecture",
            previewContent: renderPreviewCard("Architecture",
                            "View detailed architecture of house, data, HVAC,
lighting plans and more.",
                            baseAssetsPath + "images/preview/svg/Architecture.svg")
        }
    ];

    options.friendlyFileName = '';

    // Initialize DsPdfViewer and add document list panel.
    const viewer = new DsPdfViewer("#viewer", options);
    viewer.addDefaultPanels();
    const viewerDefault = viewer.toolbarLayout.viewer.default;
    viewerDefault.splice(viewerDefault.indexOf("open"), 1);
    const documentListPanelHandle = viewer.addDocumentListPanel();
    viewer.onAfterOpen.register(function() {
        viewer.leftSidebar.menu.panels.open(documentListPanelHandle.id);
    });

    // Open default PDF.
    viewer.open("/documents-api-pdfviewer/demos/product-bundles/assets/pdf/viewer-
pdf-document-list.pdf");
}

// Set HTML markup for the PDFs.
function renderPreviewCard(displayName, description, imgSrc) {
    return `< button class="preview-card gc-btn">
    <div class="col1" style="background-image: url(${imgSrc});"></div>
    <div class="col2"><h2>${displayName}</ h2 >< p >${ description}</ p ></ div >
</ button >`;
}
```



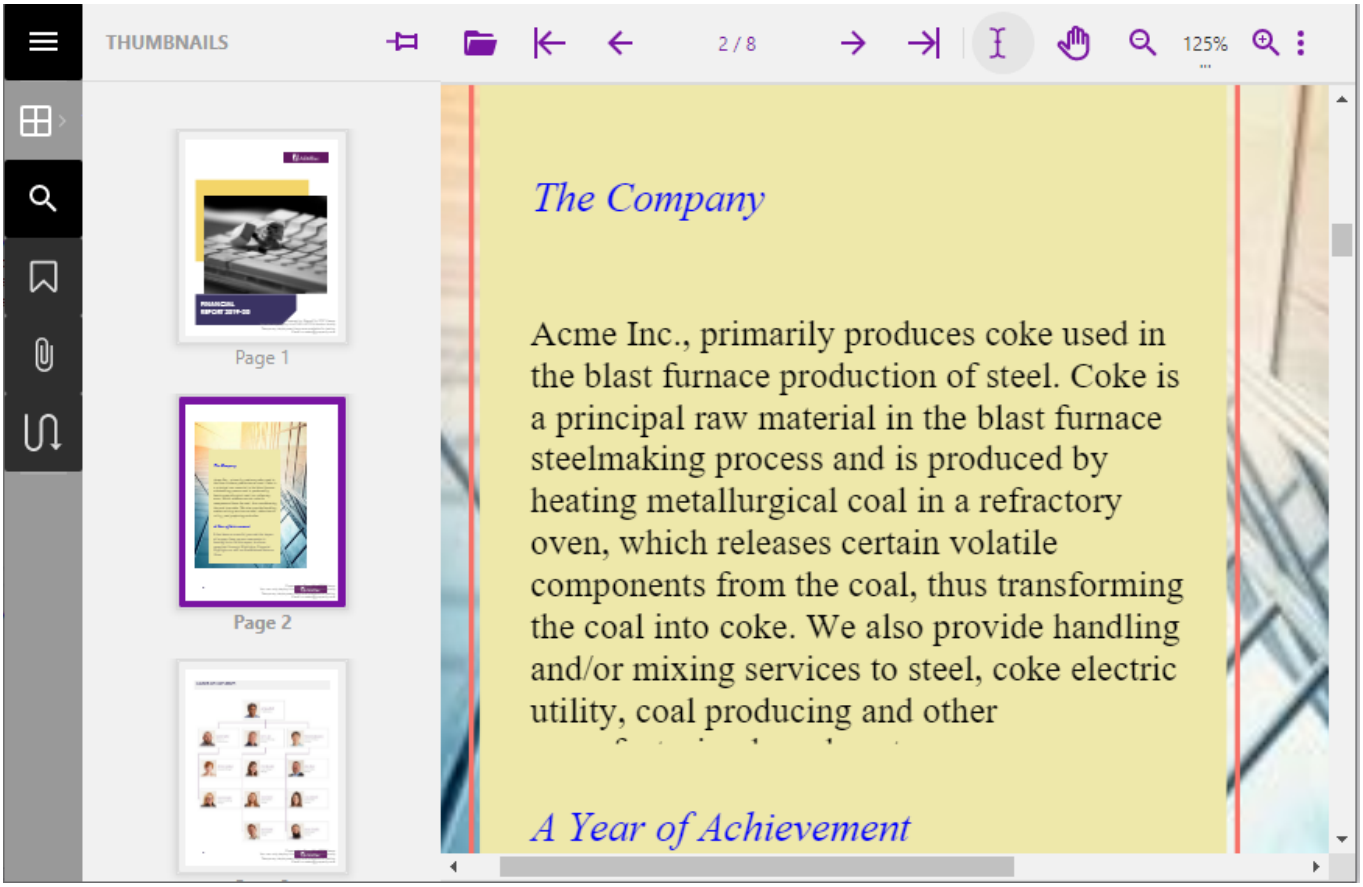
Customize Theme using CSS Styles

You can change the theme of DsPdfViewer 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>
```



Customize Context Menu Activation on Text Selection

You can customize the context menu display of DsPdfViewer using the value of **showContextMenuOnSelection** option. This option has the following three values:

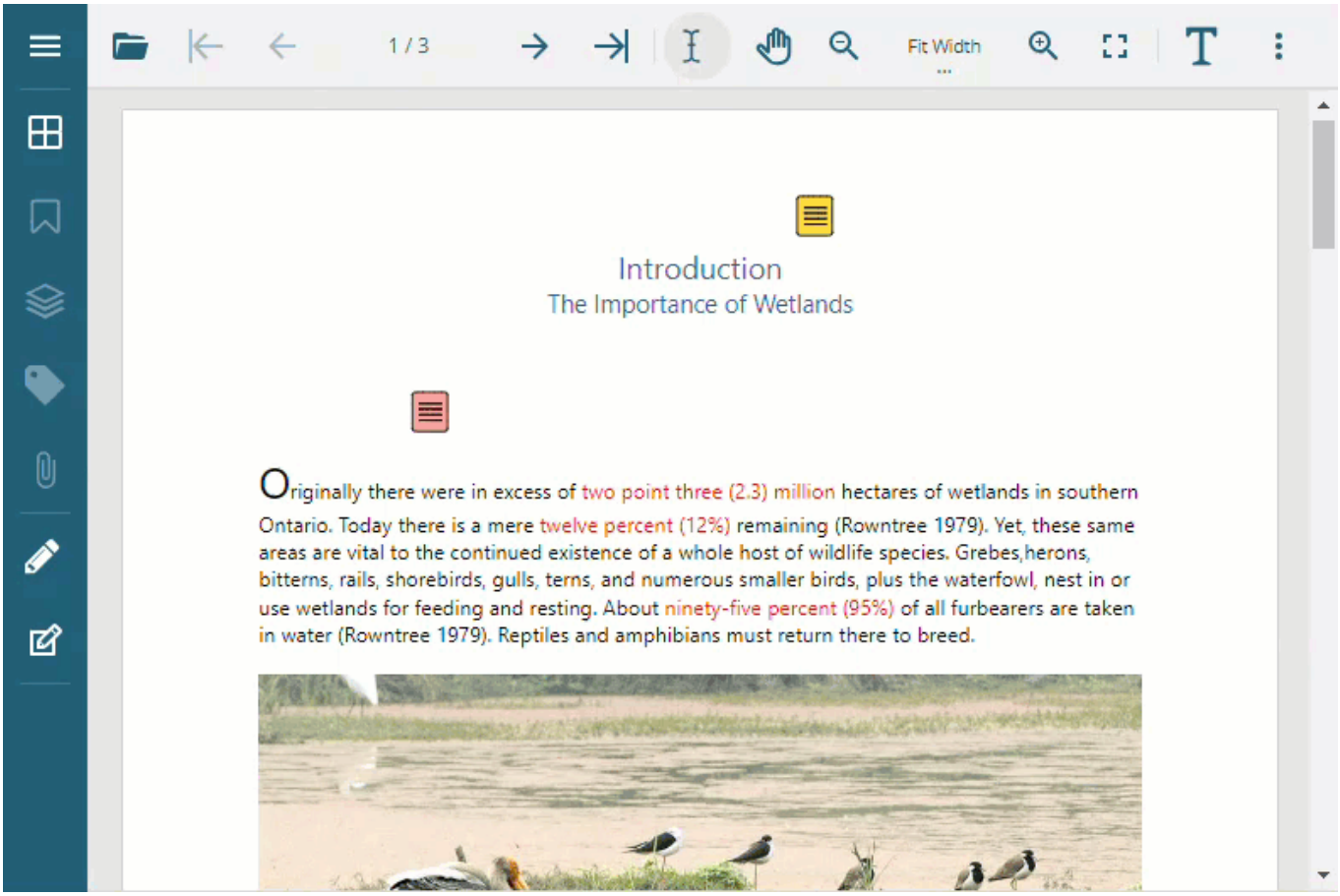
Values	Description
Auto	Automatically determines whether to show the context menu on text selection based on the device type. Auto sets the value to "Off" on devices with a mouse and "On" on devices without a mouse.
On	Shows the context menu as soon as the text is selected, irrespective of the device type.
Off	Never shows the context menu when text is selected; rather, the user should right-click on the selected text to show the context menu.

By default, the value of showContextMenuOnSelection option is set to Auto, which makes the DsPdfViewer display the context menu by right-clicking on the selected text when working on devices with a mouse, while the menu is displayed as soon as the text is selected on devices without a mouse.

Refer to the following example code to always show the context menu when the text is selected:

JavaScript

```
// Set context menu to always show.
var viewer = new DsPdfViewer("#host", { supportApi: 'api/pdf-viewer',
showContextMenuOnSelection: "On" });
```



Localization

You can localize the DsPdfViewer 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', 'gotonext': '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. Bearbeitungswerkzeuge sind deaktiviert.', 'supportApiNotAvailable': 'Support API-Server nicht verfügbar. Bearbeitungswerkzeuge 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', 'deeppink': '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' },
```

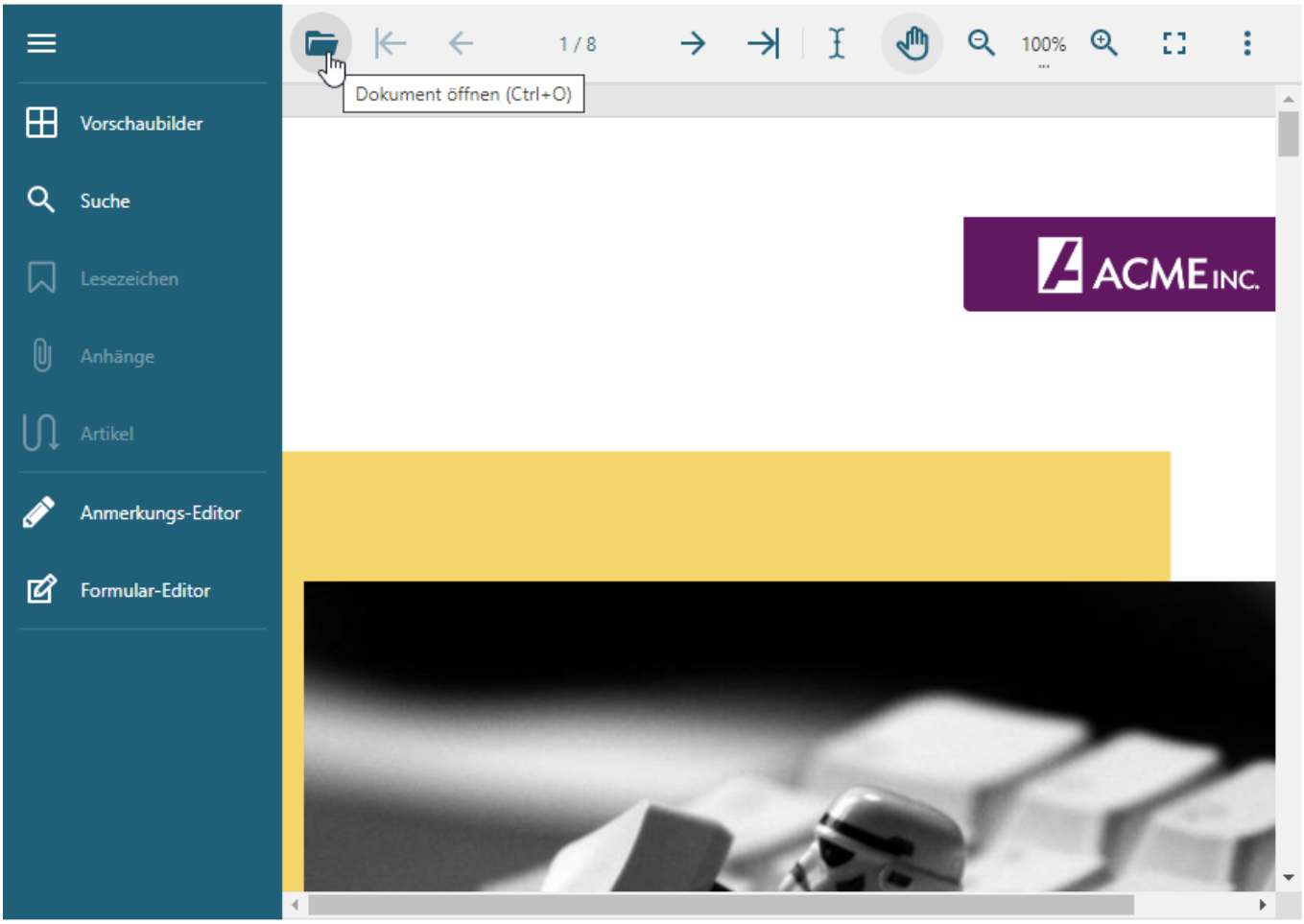
'no-file': { 'label': 'Keine Datei' } }, 'image-file-editor': { 'no-image': { 'label': 'Kein Bild' }, 'select-image': { 'title': 'Bilddatei auswählen' }, 'remove-image': { 'title': 'Bild entfernen' }, 'download-image': { 'title': 'Image herunterladen' }, 'reset-aspect-ratio': { 'title': 'Bild-Seitenverhältnis 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': 'Lesezeichen', 'title': 'Lesezeichen' }, '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': 'Streichholzschachtel', '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}}',
'didn-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: {{currentUserName}}', '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': { 'userNameValidation': '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': 'PDF Viewer Version {{version}} ({{supportApiVersion}})', 'line1': 'PDF Viewer Version {{version}}', 'line2': 'PDF Viewer is only licensed for commercial use when

```
purchased with {{anchorStart}}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@mescius.com to purchase a license.' } }, 'nolicensekey': { 'message': {
'line1': 'License Not Found', 'line2': '', 'line3': 'You need a valid license key to
run 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@mescius.com if you need assistance' }, 'watermark': {
'line1': 'Powered by PDF Viewer.', 'line2': 'You can only deploy this EVALUATION
version locally.', 'line3': 'Temporary deployment keys are available for testing.',
'line4': 'Email us.sales@mescius.com.' } }, 'evallicense': { 'watermark': { 'line1':
'Powered by PDF Viewer.', 'line2': 'Your temporary deployment key expires in
{{expiresInDays}} day(s).' } }, 'evalexpiredlicense': { 'message': { 'line1':
'Powered by PDF Viewer.', 'line2': '', 'line3': 'Your temporary deployment key has
expired.', 'line4': 'Email us.sales@mescius.com for help.' } }, 'localhostonly': {
'message': { 'line1': 'License Not Found', 'line2': '', 'line3': 'You need a valid
license key to run 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@mescius.com if you need assistance.' }
}, 'keyforanotherproduct': { 'message': { 'line1': 'This license key is for a
different product.', 'line2': '', 'line3': 'Contact us.sales@mescius.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@mescius.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:
DsPdfViewer.i18n.init({
    resources: { 'DE': { viewer: translation } },
    defaultNS: 'viewer'
});

var viewer = new DsPdfViewer("#host",
```

```
    { supportApi: 'api/pdf-viewer', language: 'DE' }

);
viewer.addDefaultPanels();
viewer.addAnnotationEditorPanel();
viewer.addFormEditorPanel();
```

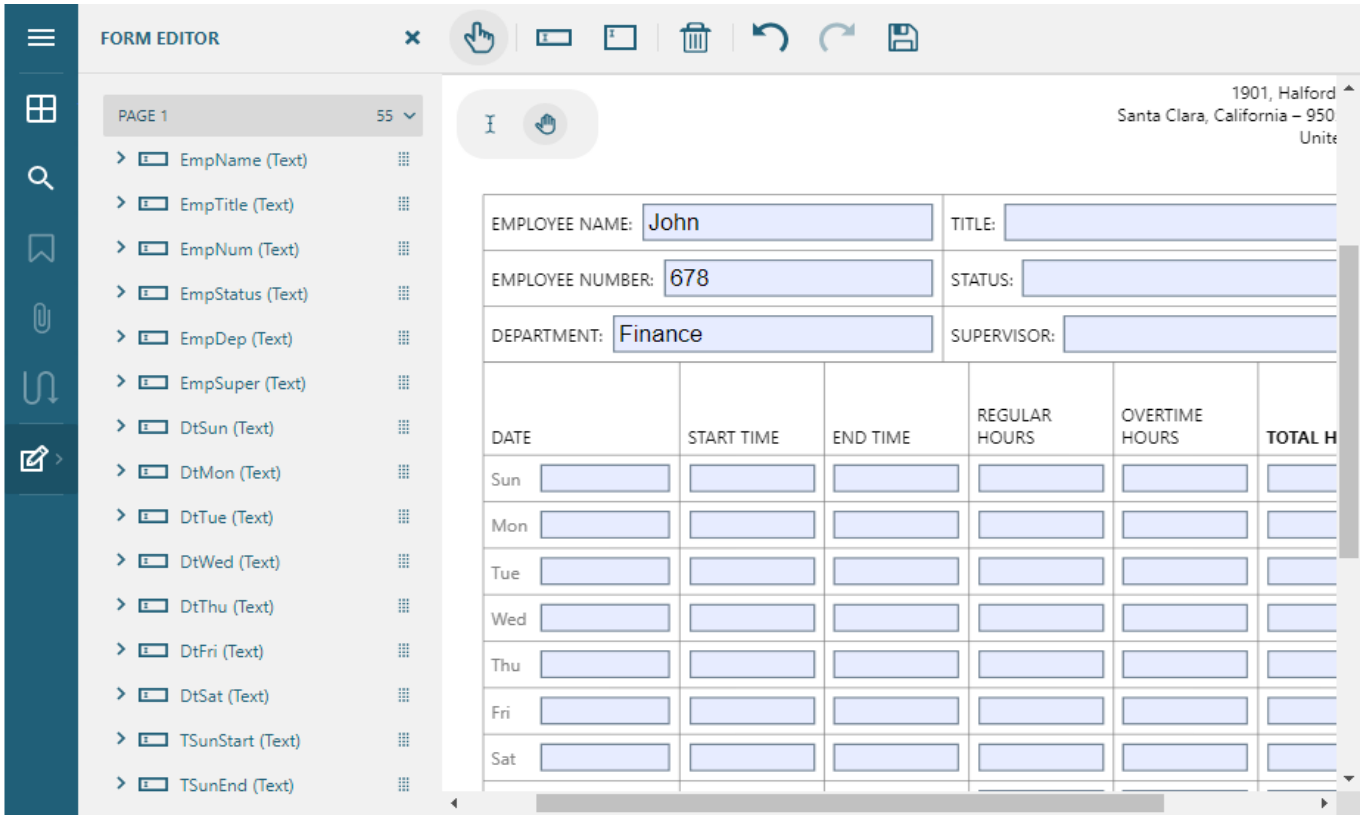


Customizations while Editing PDF Documents

When DsPdfViewer 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 DsPdfViewer("#host", { supportApi: 'SupportApi/DsPdfViewer' });
    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 DsPdfViewer("#host", { supportApi: 'SupportApi/DsPdfViewer' });
    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 DsPdfViewer("#host", { supportApi: 'SupportApi/DsPdfViewer' });
    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 DsPdfViewer("#host", { supportApi: 'SupportApi/DsPdfViewer' });
    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>
```

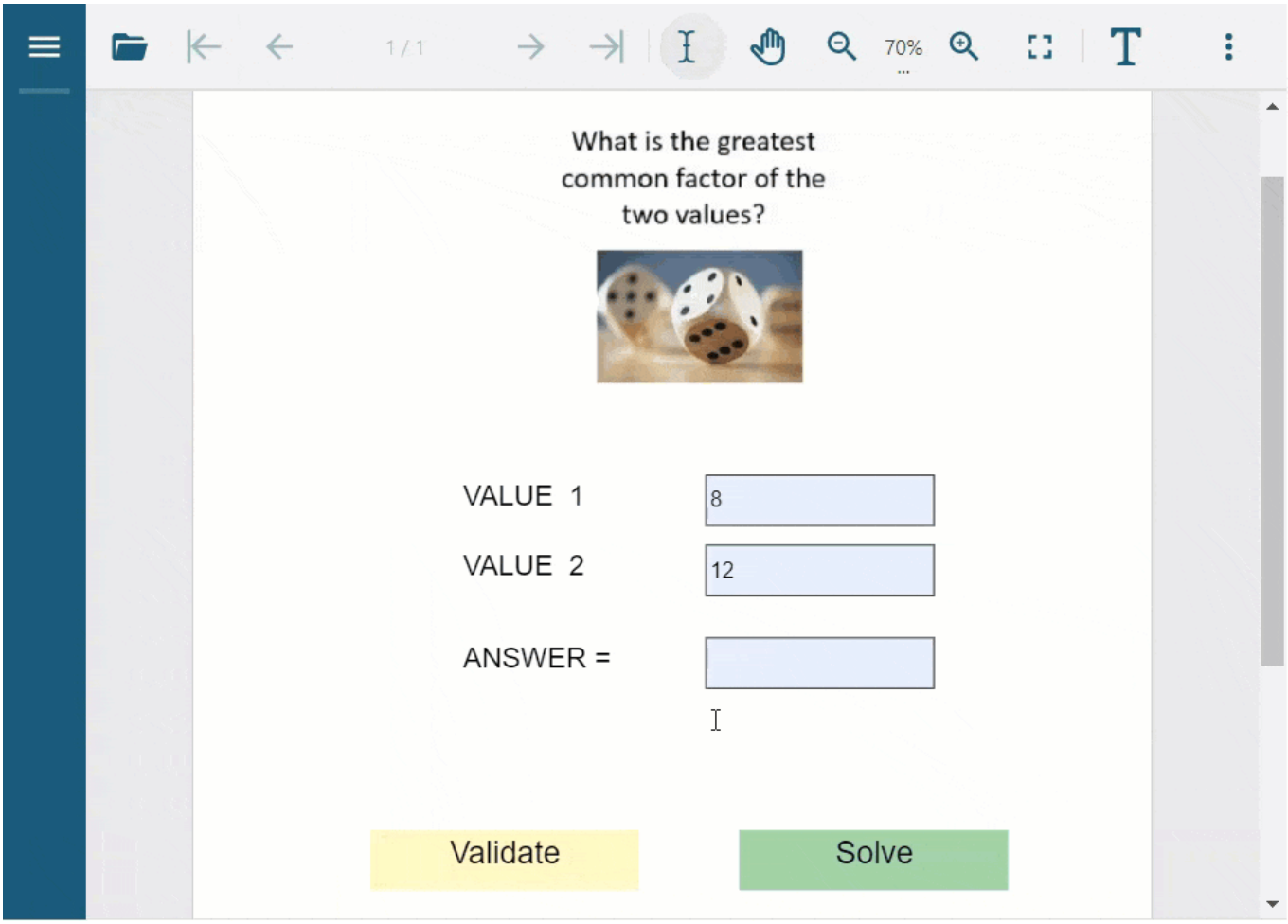
Custom Validation of Form Fields

You can use the **validateForm** method to validate an active form in DsPdfViewer. 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 DsPdfViewer(selector, options);
    return viewer;
}
createPdfViewer("#host");
viewer.open("viewer-custom-validation.pdf?ts=1");
```



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.

Limitations

DsPdfViewer prohibits the usage of script and iframe tags in custom HTML content due to security considerations.

Form Filler

Form Filler feature, in DsPdfViewer, 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 DsPdfViewer
- 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.

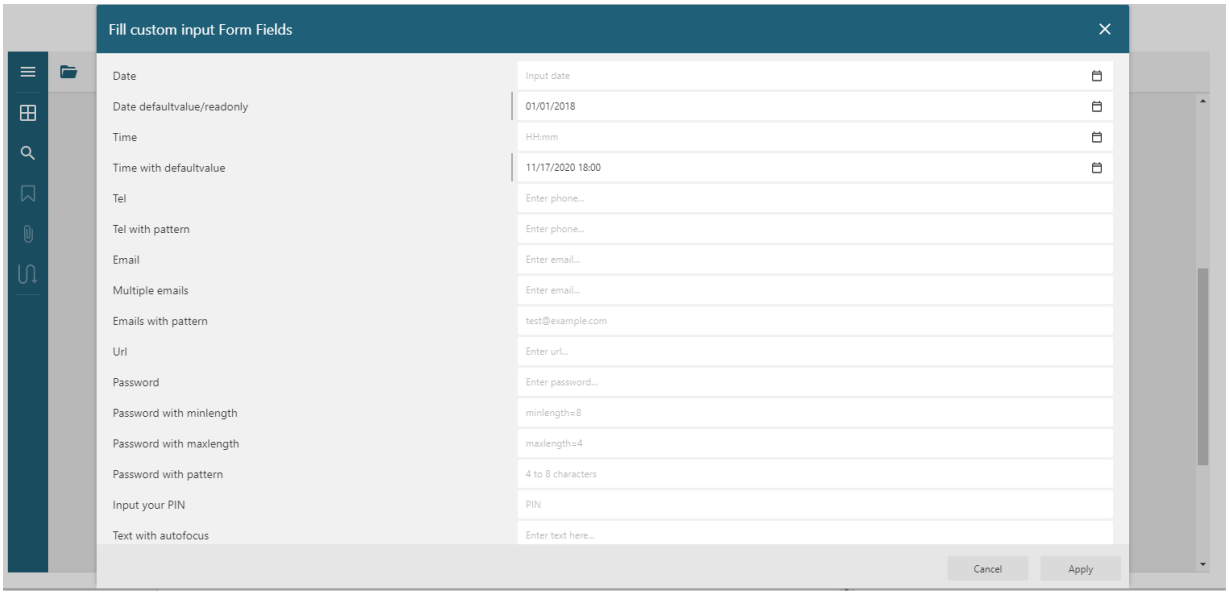
Note: SupportAPI is only needed if you want to save the filled form on client. Otherwise, load a PDF Form with custom form input types (explained below in this topic), enable the Form Filler dialog, fill the PDF Form within the Form Filler dialog, or even

> print a filled form on the client without using SupportApi.

Fill Forms using Form Filler Dialog

The 'Form Filler' button is provided in DsPdfViewer 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 DsPdfViewer project as covered in View PDF.

The DsPdfViewer'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 DsPdfViewer(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;
}
```

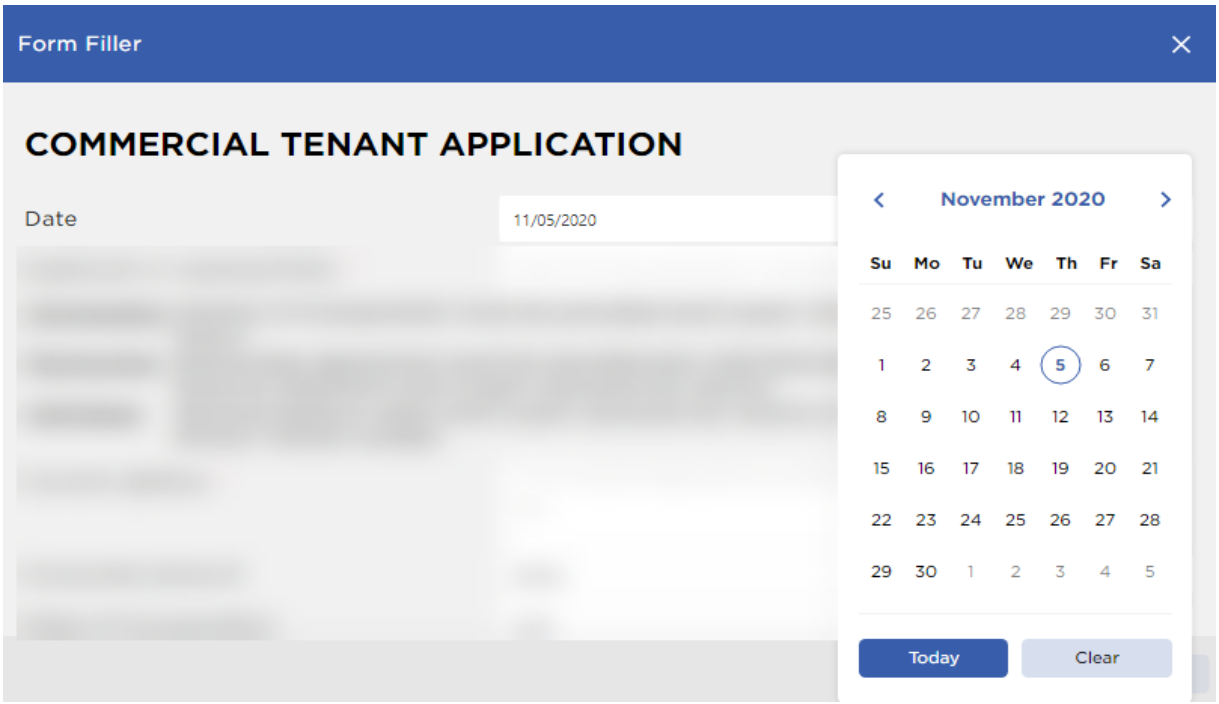
The sample PDF form used in this example is a tenant application form which seeks the details of a tenant for house rental purposes. You can also download the form here.

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

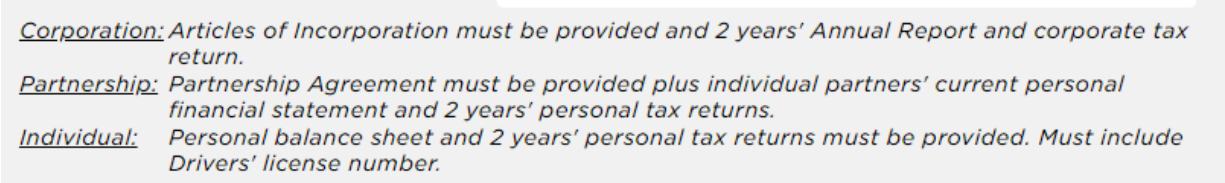
```
'Entity': {
        title: 'Name of individual, partnership, or corporation',
        displayname: 'Applicant or Leasing Entity',
        placeholder: 'Name of individual, partnership, or corporation',
}
```

- 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>
```

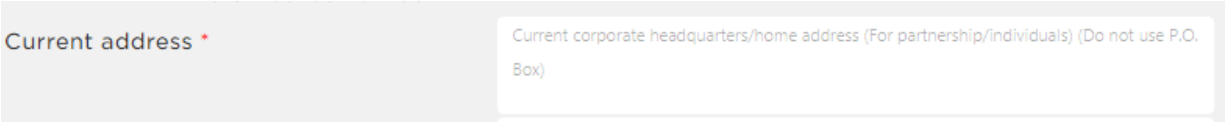
```
                    </table>`
            },
```



- 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,
    }
```

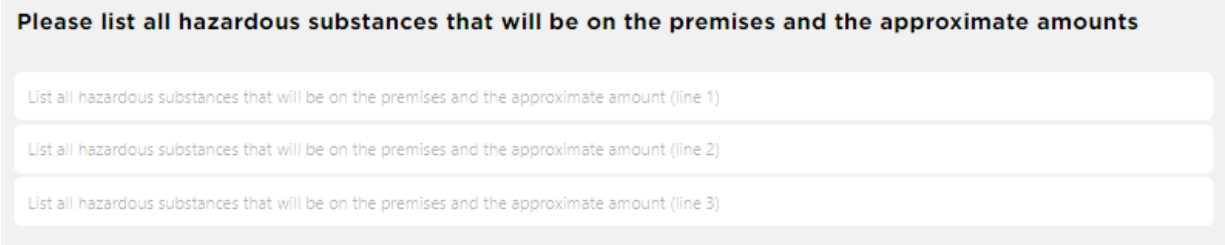


- A full-width field is displayed without a label by using the **nolabel** property.

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
    }
```



Add Validation Properties for Form Input Types

- A required input type validation is added to 'Entity' field by using the **required** property.

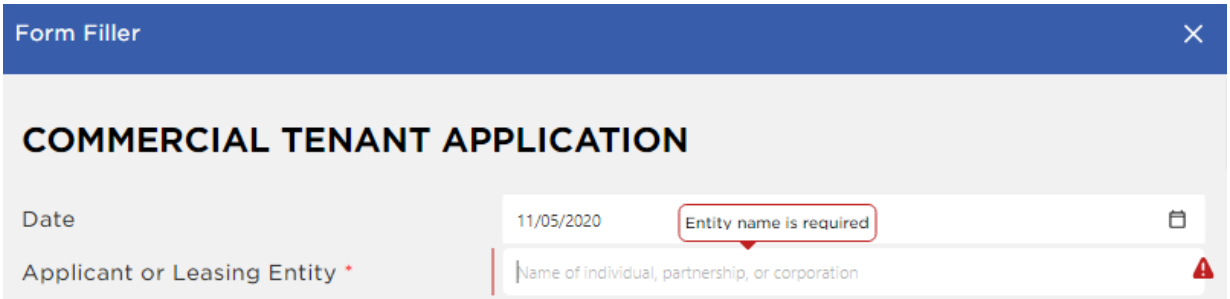
Index.cshtml

```
'Entity': {
                title: 'Name of individual, partnership, or corporation',
                displayname: 'Applicant or Leasing Entity',
                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
            },
```

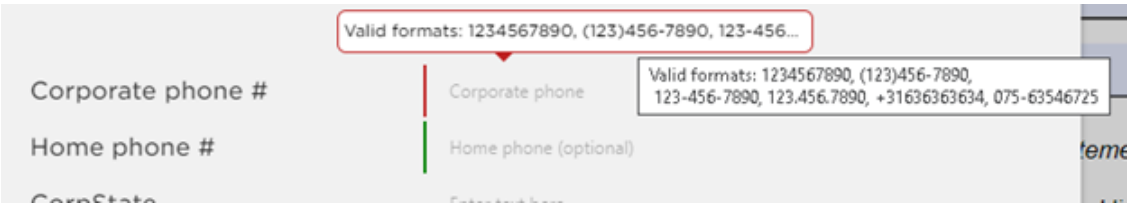


- A pattern can be set to validate 'Corporate phone number' input type using regular expression by using the **pattern** property.

Index.cshtml

```
 'CorpPhone': {
```

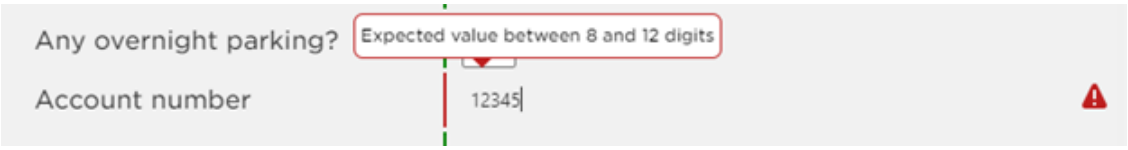
```
                title: 'Corporate phone number',
                displayname: 'Corporate phone #',
                placeholder: 'Corporate phone',
                type: 'tel',
                pattern: '^[\+]?[()?[0-9]{3}[)]?[-\s\.]?[0-9]{3}[-\s\.]?[0-9]{4,6}$',
                validationmessage: 'Valid formats: 1234567890, (123)456-7890,\n 123-456-7890, 123.456.7890, +31636363634, 075-63546725',
                validateoninput: true
            },
```



- 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
    }
```



- 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;
                }
            },
```

- A common **validator** function is called for each input type in the list.

Index.cshtml

```
var viewer = new DsPdfViewer(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, DsPdfViewer 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 DsExcel Templates or by using GcProps property in DsPdf. The form can then be filled in DsPdfViewer 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" "First line\nSecond line"	Uses the title property to specify text that most 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 DsPdf.

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.DefaultAppearance.Font = tf.Font;
field.Widget.DefaultAppearance.FontSize = tf.FontSize;
field.Name = "dateField1";
field.GcProps[new PdfName("type")] = new PdfString("date");
field.GcProps[new PdfName("required")] = new PdfBoolObject((PdfBool)true);
field.GcProps[new PdfName("validationmessage")] = new PdfString("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 cannot be empty."	Represents a localized message that describes the validation constraints that the control 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 DsPdf.

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.DefaultAppearance.Font = tf.Font;
field.Widget.DefaultAppearance.FontSize = tf.FontSize;
field.Name = "timeField1";
field.GcProps[new PdfName("type")] = new PdfString("time");
field.GcProps[new PdfName("required")] = new PdfBoolObject((PdfBool)true);
field.GcProps[new PdfName("validateoninput")] = new PdfBoolObject((PdfBool)true);
field.GcProps[new PdfName("validationmessage")] = new PdfString("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 DsPdf.

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.DefaultAppearance.Font = tf.Font;
field.Widget.DefaultAppearance.FontSize = tf.FontSize;
field.Name = "telField1";
field.GcProps[new PdfName("type")] = new PdfString("tel");
field.GcProps[new PdfName("title")] = new PdfString("Text area code");
field.GcProps[new PdfName("placeholder")] = new PdfString("###");
field.GcProps[new PdfName("pattern")] = new PdfString(@"^[0-9]{3}$");
field.GcProps[new PdfName("validateoninput")] = new PdfBoolObject((PdfBool)true);
field.GcProps[new PdfName("validationmessage")] = new PdfString(@"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 DsPdf.

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.DefaultAppearance.Font = tf.Font;
field.Widget.DefaultAppearance.FontSize = tf.FontSize;
field.Name = "numberField1";
field.GcProps[new PdfName("type")] = new PdfString("number");
field.GcProps[new PdfName("title")] = new PdfString("Enter number between 1 and 100");
field.GcProps[new PdfName("placeholder")] = new PdfString("[1-100]");
field.GcProps[new PdfName("required")] = new PdfBoolObject((PdfBool)true);
field.GcProps[new PdfName("min")] = new PdfNumber(1);
field.GcProps[new PdfName("max")] = new PdfNumber(100);
field.GcProps[new PdfName("validateoninput")] = new PdfBoolObject((PdfBool)true);
field.GcProps[new PdfName("validationmessage")] = new PdfString("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 ths 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.
validateonmessage	"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 DsPdf.

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.DefaultAppearance.Font = tf.Font;
field.Widget.DefaultAppearance.FontSize = tf.FontSize;
field.Name = "emailField1";
field.GcProps[new PdfName("type")] = new PdfString("email");
field.GcProps[new PdfName("title")] = new PdfString("Input your e-mail address");
field.GcProps[new PdfName("placeholder")] = new PdfString("Input your email");
field.GcProps[new PdfName("pattern")] = new PdfString(@".+@globex.com");
field.GcProps[new PdfName("validateoninput")] = new PdfBoolObject((PdfBool)true);
field.GcProps[new PdfName("validationmessage")] = new PdfString(@"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" "current-password" "new-password" "one-time-code"	Allows the user's password manager to automatically enter the password.
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-z0-9._%+-]+@[a-z0-9.-]+\.[a-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	Represents a localized message that describes the validation constraints that the control does not satisfy (if any).

	be 3 digits."	

The 'password' input type and its properties can also be added to a PDF form by using the following example code in DsPdf.

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.DefaultAppearance.Font = tf.Font;
field.Widget.DefaultAppearance.FontSize = tf.FontSize;
field.Name = "pinField1";
field.GcProps[new PdfName("type")] = new PdfString("password");
field.GcProps[new PdfName("title")] = new PdfString("Input your PIN");
field.GcProps[new PdfName("placeholder")] = new PdfString("PIN");
field.GcProps[new PdfName("pattern")] = new PdfString(@"^[0-9]{3,3}$");
field.GcProps[new PdfName("maxlength")] = new PdfNumber(3);
field.GcProps[new PdfName("validateoninput")] = new PdfBoolObject((PdfBool)true);
field.GcProps[new PdfName("autocomplete")] = new PdfString("one-time-code");
field.GcProps[new PdfName("validationmessage")] = new PdfString(@"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"	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.

	"organization" "street-address" "address-line1" "address-line2" "address-line3" "address-level4" "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	Set this property to true to use the text area as a user input element.

	"false"	
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 "false"	The spellcheck property is an enumerated attribute that defines whether the element may 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 DsPdf.

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.DefaultAppearance.Font = tf.Font;
field.Widget.DefaultAppearance.FontSize = tf.FontSize;
field.Name = "textField1";
field.GcProps[new PdfName("type")] = new PdfString("text");
field.GcProps[new PdfName("title")] = new PdfString("Input text");
field.GcProps[new PdfName("placeholder")] = new PdfString("Input here");
field.GcProps[new PdfName("maxlength")] = new PdfNumber(10);
field.GcProps[new PdfName("validateoninput")] = new PdfBoolObject((PdfBool)true);
field.GcProps[new PdfName("validationmessage")] = new PdfString("The value is incorrect,
maximum length is 10.");
doc.AcroForm.Fields.Add(field);
```

DsPdf 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 DsPdf.

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.DefaultAppearance.Font = tf.Font;
field.Widget.DefaultAppearance.FontSize = tf.FontSize;
field.Name = "monthField1";
field.GcProps[new PdfName("type")] = new PdfString("month");
field.GcProps[new PdfName("title")] = new PdfString("Enter month.");
field.GcProps[new PdfName("placeholder")] = new PdfString("month");
field.GcProps[new PdfName("required")] = new PdfBoolObject((PdfBool)true);
field.GcProps[new PdfName("validationmessage")] = new PdfString("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 DsPdf.

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.DefaultAppearance.Font = tf.Font;
field.Widget.DefaultAppearance.FontSize = tf.FontSize;
field.Name = "rangeField1";
field.GcProps[new PdfName("type")] = new PdfString("range");
field.GcProps[new PdfName("title")] = new PdfString(" Please select a number between 1 and
1000.");
field.GcProps[new PdfName("min")] = new PdfNumber(1);
field.GcProps[new PdfName("max")] = new PdfNumber(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 DsPdf, refer Fill Custom Form Input Types.

Fill Custom Form Input Types

DsPdfViewer 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 DsExcel Templates or DsPdf. The forms with custom form input types can only be opened and viewed in DsPdfViewer (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 DsPdf'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", "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)}
            }

        };
```

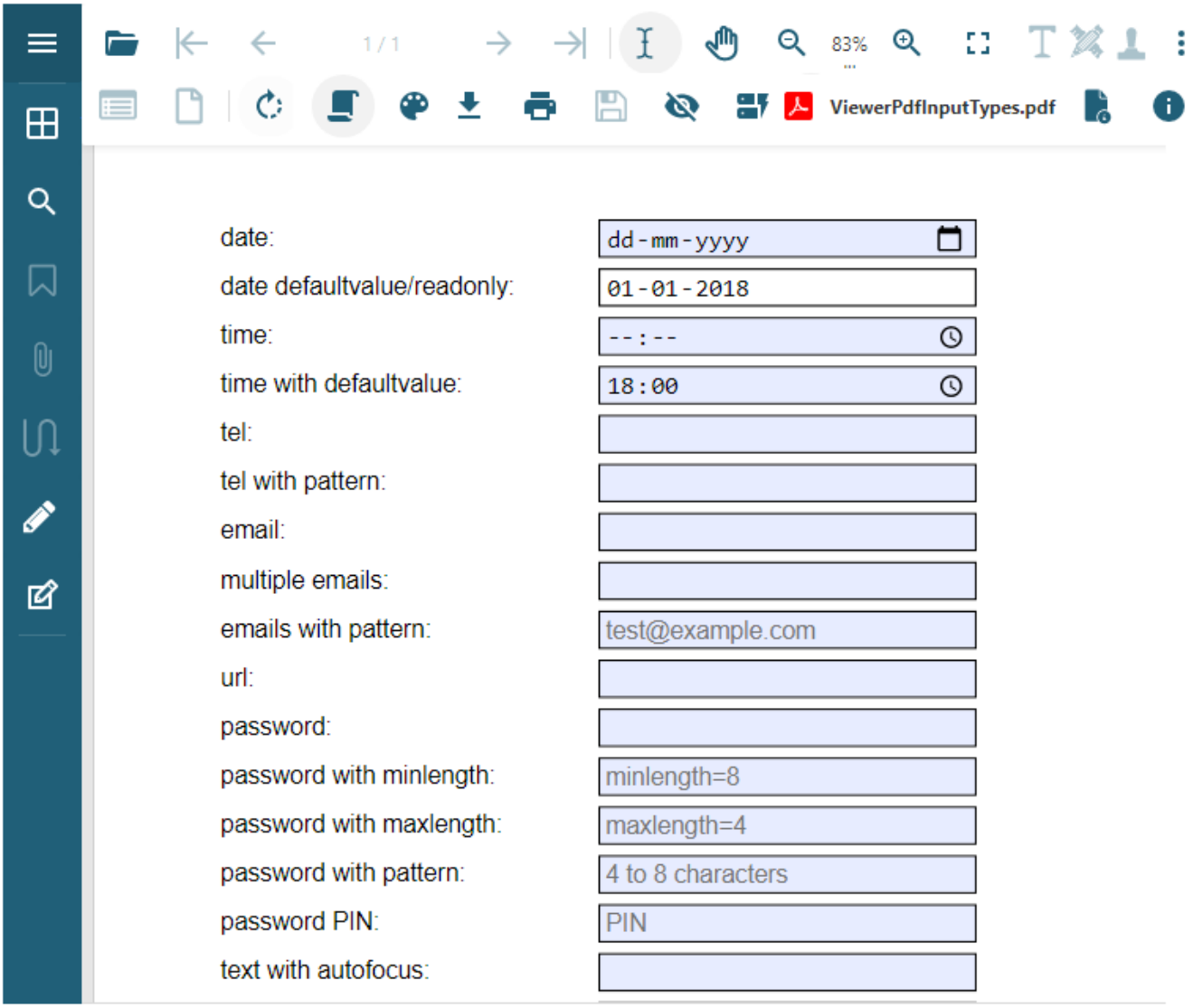
The below mentioned code adds the custom form input types defined above to a PDF form.

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;

            // サンプルフィールドを追加します
           int orderindex = 1;
            foreach(var data in SampleGcPropTextFields)
            {
                string fieldName = data.Key;

                g.DrawString($"{fieldName}:", tf, ip);

                TextField field = new TextField();
                // GcPropsディクショナリーを記入します
               KeyValuePair<string, object>[] gcProps = data.Value;
                foreach(var gcProp in gcProps)
                {
                    field.GcProps[new PdfName(gcProp.Key)] =
IPdfObjectExt.PdfNameFromString(gcProp.Value.ToString());
                }
                field.GcProps[new PdfName("orderindex")] = new
PdfNumber(orderindex);
                field.Name = fieldName;
                field.Widget.Page = page;
                field.Widget.Rect = new RectangleF(ip.X + fldOffset, ip.Y, 72 * 3,
fldHeight);
                field.Widget.DefaultAppearance.Font = tf.Font;
                field.Widget.DefaultAppearance.FontSize = tf.FontSize;
                doc.AcroForm.Fields.Add(field);
                ip.Y += dY;
                orderindex++;
            }

            // 保存して完了します。
           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 DsExcel. For more information, refer Custom Form Input Types.

Keyboard Shortcuts

The following table lists the keyboard shortcuts supported in DsPdfViewer that will help in performing a variety of supported operations more efficiently and make the DsPdfViewer components more accessible:

Shortcut Key	Key Code	Action	Action Description
Ctrl + "Numpad +"	107	zoomin	Zoom in.
Ctrl + "+"	187	zoomin	Zoom in.
Ctrl + "Numpad -"	109	zoomout	Zoom out.
Ctrl + "-"	189	zoomout	Zoom out.

Ctrl + 0	48	zoom_pagesize	Zoom to 100%.
Ctrl + 9	57	zoom_clientsize	Zoom to page width.
Ctrl + A	65	select_all	Select all text.
R	82	rotate	Rotate clockwise.
Shift + R	82	rotate	Rotate counterclockwise.
H	72	pan	Enable the pan tool.
S	83	selection	Enable the text selection tool.
Spacebar	32	holdToPan	Press and hold the Spacebar to temporarily enable the pan.
Ctrl + O	79	open	Open file from the local system.
Ctrl + F	70	search	Open the search panel.
Ctrl + P	80	print	Print document.
Left Arrow	37	move_caret_left	Move the caret (text cursor) left.
Up Arrow	38	move_caret_up	Move the caret up.
Down Arrow	40	move_caret_down	Move the caret down.
Right Arrow	39	move_caret_right	Move the caret right.
Home	36	move_caret_sequence_start	Move the caret to the start of the sequence.
Ctrl + Home	36	move_caret_document_start	Move the caret to the start of the document.
End	35	move_caret_sequence_end	Move the caret to the end of the sequence.
Ctrl + End	35	move_caret_document_end	Move the caret to the end of the document.
Ctrl + Enter	13	confirm-ok	Confirm changes.
ESC	27	confirm-cancel	Cancel changes
Page Up	33	scrollUp	Scroll up.
Page Down	34	scrollDown	Scroll down.

Customize Keyboard Shortcuts

DsPdfViewer also supports redefining, disabling, overriding, and removing the default keyboard shortcuts, as well as binding the default keyboard shortcuts to other keys and creating custom keyboard shortcuts via API using **shortcuts** option of ViewerOptions class.

Bind Keyboard Shortcuts

DsPdfViewer supports binding any tool to any keyboard shortcut key. The following are the available built-in tools accessible through keyboard shortcut keys:

Keyboard Shortcut Tools	
zoomin	move_caret_left
zoomout	move_caret_up

zoom_pagesize	move_caret_down
zoom_clientsize	move_caret_right
select_all	move_caret_sequence_start
rotate	move_caret_document_start
pan	move_caret_sequence_end
selection	move_caret_document_end
holdToPan	confirm-ok
open	confirm-cancel
search	scrollUp
print	scrollDown

Refer to the following example code to bind the holdToPan action to “P” key:

```
// Bind the "P" shortcut to the holdToPan action and leave the Ctrl+P shortcut for
the "print" action.
viewer.options.shortcuts["P"] = [{ ctrl: true, tool: "print" }, { tool: "holdToPan"
}];
```

Disable Keyboard Shortcuts

Refer to the following example code to disable holdToPan keyboard shortcut:

```
/* Initialize DsPdfViewer and
 set the function(event) for both keys to null. */
var viewer = new DsPdfViewer("#viewer", {

    shortcuts: {

        "Z": {
ctrl: true,
            tool: function(event) { }
},
        "Y": {
ctrl: true,
            tool: function(event) { }
}
    }
});

// Disable holdToPan shortcut.
viewer.options.shortcuts["32"] = () => { };
```

Remove Default Keyboard Shortcuts

Refer to the following example code to remove all default keyboard shortcuts:

```
// Initialize DsPdfViewer and set the shortcuts to null.
var viewer = new DsPdfViewer("#root");
viewer.options.shortcuts = { };
```

Override Keyboard Shortcuts

Refer to the following example code to override Ctrl + “Numpad +“ keyboard shortcut:

```
/* Initialize DsPdfViewer and
 set the function(event) to zoom the PDF page to 10%. */
var viewer = new DsPdfViewer("#root", {
  shortcuts: {
    107: {
ctrl: true,
      tool: function(event) {
        DsPdfViewer.findControl("#root").zoomValue += 10;
          event.preventDefault();
    }
}
  }
});
```

Create Custom Keyboard Shortcuts

Refer to the following example code to create a custom keyboard shortcut:

```
/* Create a custom keyboard shortcut to
 alert the user that Ctrl+Alt+E is pressed. */
viewer.options.shortcuts["E"] = {
ctrl: true,
  alt: true,
  tool: function(e) { alert("Ctrl+Alt+E pressed."); }
};
```

Client API Reference

Please refer DsPdfViewer API for all the assemblies required to create applications using DsPdfViewer.

Samples

All the DsPdf 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.

Walkthrough

Follow the walkthrough, which is a step-by-step tutorial, to understand more complex tasks that can be accomplished using DsPdf.

- 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

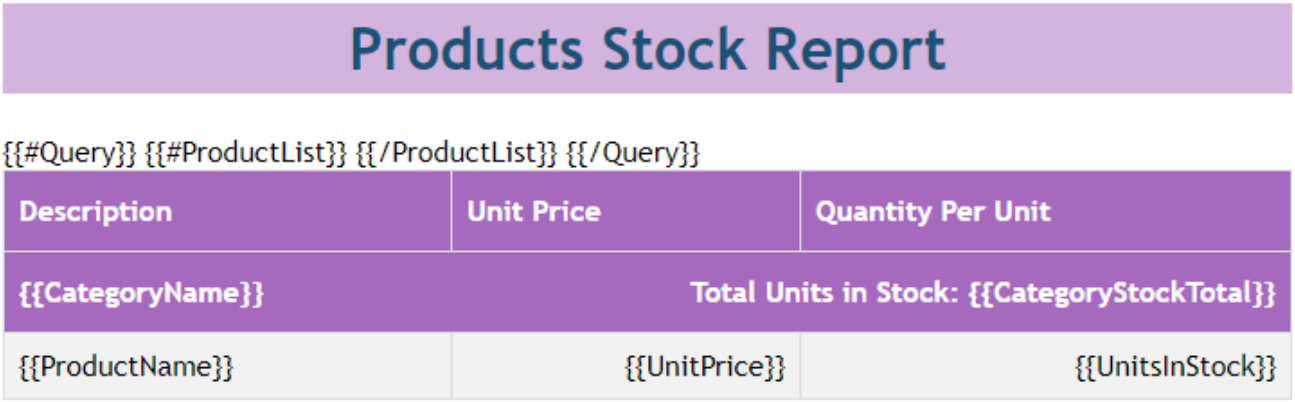
ProductListTemplate.html

```
<!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 align='right'>{{UnitsInStock}}</td>

    </tr>
        {{/ProductList}}
        {{/Query}}
    </table>
</body>
</html>
```

The below image shows the output of HTML template file:



Create a console application

1. Create a .NET Core console application in Visual Studio and install 'DS.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;
using GcPdfWeb.Samples.Common;
```

```
    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 **GcHtmlBrowser** class that is used to render HTML.
2. To render the bound HTML, pass the data bound template as a parameter to the **NewPage** method of the browser.
3. Define PDF options such as header and footer, for rendering HTML to PDF using the **PdfOptions** class.
4. Render the generated HTML to a temporary file by using **GetTempFileName** method of the **Path** class.
5. Create a PDF file from the rendered temporary file by using the **SaveAsPdf** method.

C#

```
// Create an instance of GcHtmlBrowser that is used to render HTML:
using var browser = Util.NewHtmlBrowser();
    // Render the bound HTML:
    using var htmlPage = browser.NewPage(boundTemplate);
    // PdfOptions specifies options for HTML to PDF conversion:
    var pdfOptions = new PdfOptions()
    {
        Margins = new PdfMargins(0.2f, 1, 0.2f, 1),
        PreferCSSPageSize = false,
        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) MESCIUS, 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:
    var tmp = Path.GetTempFileName();
    htmlPage.SaveAsPdf(tmp, pdfOptions);

    // 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 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:

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

 Generated on 14/2/2020

API Reference

This section contains documentation for all the assemblies required to create applications using DsPdf.

Assembly	Description
DS.Documents.Barcode	Cross-platform library that provides an object model for creating barcodes.
DS.Documents.Imaging	Cross-platform library for working with raster images.
DS.Documents.Imaging.Windows	Platform-specific library that allows other DS.Documents packages to work with optimized system APIs on Windows.
DS.Documents.Html	Cross-platform library that provides HTML processing and rendering features.
DS.Documents.Pdf	Cross-platform library that allows you to create, analyze, and modify PDF documents.

Release Notes

Release notes are available for both DsPdf and DsPdfViewer.

- Breaking Changes
- DsPdf Release Notes
- DsPdfViewer Release Notes

Breaking Changes

Refer to the DsPdf breaking changes for specific versions:

- Version 7.0.0
- Version 6.2.0
- Version 6.1.0
- 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 DsPdfViewer breaking changes here:

- Version 3.1.2

DsPdf Release Notes

Refer to the following release notes for the major releases of DsPdf.

- Release Notes for Version 7.1.0
- Release Notes for Version 7.0.0
- Release Notes for Version 6.2.0
- Release Notes for Version 6.1.0
- Release Notes for Version 6.0.0
- Release Notes for Version 5.2.0.800
- 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 7.1.0

New Features and Improvements

- Added RichMediaAnnotation class: Represents a rich media annotation.
- Added RichMediaAnnotationActivation class: Specifies the activation conditions of a RichMediaAnnotation.
- Added RichMediaAnnotationDeactivation class: Specifies the deactivation conditions of a RichMediaAnnotation.
- Added RichMediaAnnotationPresentationStyle class: Defines possible values for RichMediaAnnotation.PresentationStyle.
- Added RichMediaAnnotationContentType enumeration: Defines possible types of RichMediaAnnotation content.
- Added GcPdfDocument.RemoveDuplicateImages method: Removes images with identical content from the current document.
- Added MergeDocumentOptions.RemoveDuplicateImages property: Gets or sets a value indicating whether to remove duplicate images after merging.
- Added GcGraphics.DrawRotatedText() method: Draws rotated text inside an unrotated rectangle (similar to how MS Excel draws rotated text in borderless cells).
- Added GcGraphics.DrawSlantedText() method: Draws rotated text inside a slanted rectangle (similar to how MS Excel draws rotated text in cells with borders).
- Added GcGraphics.MeasureRotatedText() method: Calculates the bounds of rotated text inside an unrotated rectangle.
- Added GcPdfDocument.GetImages method overload that allows including unreferenced images (by default, they are not included).
- Fixed null reference exception occurs when merging a certain PDF with other PDFs.
- Fixed exception that occurs when saving a certain PDF as an image.

Version 7.0.0

Important Note

This is the initial release of the DS.Documents.Pdf package. This package replaces GrapeCity.Documents.Pdf, and provides the same functionality, ensures future enhancements, and is backwards compatible with GrapeCity.Documents.Pdf. Existing subscriptions will continue to apply to this new package.

Breaking Changes from GrapeCity.Documents.Pdf 6.2.5 version

- Method IPdfObject.IsContentEqual(IPdfObject) is removed. Static method GrapeCity.Documents.Pdf.Spec.IPdfObjectExt.IsContentEqual(IPdfObject v1, IPdfObject v2) can be used instead.

New Features and Improvements

- GrapeCity.Documents.Pdf.RawImage class is marked as obsolete. GrapeCity.Documents.Drawing.Image can be used instead.
- Reference to package Portable.BouncyCastle 1.8.2 has been replaced with BouncyCastle.Cryptography 2.2.1 for improved security.
- Added support for PDF caret annotations; see GrapeCity.Documents.Pdf.Annotations.CaretAnnotation for details.

Version 6.2.0

Breaking Changes

- The stream that is passed to one of GcPdfDocument.Load() method overloads must be kept open while the loaded GcPdfDocument is in use.
- If using GcPdfDocument.MergeWithDocument(), the passed GcPdfDocument is effectively in use while the current document is in use. If it was loaded from a stream, that stream must be kept open too.
- Several existing collection properties can no longer be modified in place (see below). Instead, create a new instance of the collection and assign it to the property.
- WidgetAnnotation.TextFormat property is removed (previously it was obsolete).
- GrapeCity.Documents.Pdf.Layers.OptionalContentOrderCollection class is now derived from PdfCollection<IPdfObject> (previously it was derived from List<object>).
- GrapeCity.Documents.Pdf.AcroForms.SignatureLockedFields.FieldNames property type changed from ObservableCollection<string> to IEnumerable<string>. To change the value, create a new instance of the collection and assign it to the property.
- GrapeCity.Documents.Pdf.SoundObject.EncodingFormat property type changed from string to IPdfName.
- GrapeCity.Documents.Pdf.Text.CMap.CMap class is renamed to CMapTable, and CMapBase class is removed.
- GrapeCity.Documents.Pdf.Actions.ActionHide.LinkedObjects property type changed from List<object> to LinkedObjectCollection. To change the value, create a new instance of the collection and assign it to the property.
- GrapeCity.Documents.Pdf.Actions.ActionFieldsBase.FieldNames property changed from List<string> FieldNames { get; } to IList<IFieldDef> Fields { get; }.
- Properties ViewerPreferences.PageMode and ViewerPreferences.PageLayout marked as obsolete, corresponding properties are now available directly on GcPdfDocument: GcPdfDocument.PageMode and GcPdfDocument.PageLayout.
- Property WidgetAnnotation.TriggerEvents.Activate marked as obsolete; the corresponding property is now available directly on WidgetAnnotation: WidgetAnnotation.Activate.
- GrapeCity.Documents.Pdf.Annotations.PolygonAnnotationBase.Points property type changed from IList<PointF> to IReadOnlyList<PointF>. To change the value, create a new instance of the collection and assign it to the property.
- GrapeCity.Documents.Pdf.Annotations.ImageScale class is now derived from GrapeCity.Documents.Pdf.Wrappers.PdfDictWrapper (previously it was derived from NotifyObject).
- GrapeCity.Documents.Pdf.Annotations.ButtonAppearance.Image property type changed from object to IXObject. IXObject is a common interface for FormXObject and PdfImageObjectBase. An IImage can be assigned to this property using PdfImageHandler: doc.ImageHandlers.GetImageHandler(someBitmap).
- GrapeCity.Documents.Pdf.Annotations.InkAnnotation.Paths property type changed from ObservableCollection<PointF[]> to IReadOnlyList<IReadOnlyLust<PointF>>. To change the value, create a new instance of the collection and assign it to the property.
- GrapeCity.Documents.Pdf.Annotations.TextMarkupAnnotation.Area property type changed from ObservableCollection<Quadrilateral> to IReadOnlyList<Quadrilateral>. To change the value, create a new instance of the collection and assign it to the property.
- GrapeCity.Documents.Pdf.Annotations.RedactAnnotation.Area property type changed from ObservableCollection<Quadrilateral> to IReadOnlyList<Quadrilateral>. To change the value, create a new instance of the collection and assign it to the property.
- Now, even if an annotation belongs to several pages, its vertical coordinates are converted to the DsPdf top/left coordinate system using the media box of the first page in the annotation's Pages collection. The AnnotationBase.PdfRect property provides access to native PDF coordinates.
- GrapeCity.Documents.Pdf.Structure.StructElement.ID property type changed from IList<byte> to IPdfString.
- PdfIndirectSerializableObject class was removed, the GrapeCity.Documents.Pdf.Wrappers.PdfWrapperBase and its descendants can be used instead.
- PdfDict class is now derived from Dictionary<PdfName, IPdfObject> (previously it was derived from Dictionary<string, object>). Instead of using a string directly, use new PdfName(string) instead.
- GcPdfDocument.NamedDestinations property type changed from IDictionary<string, Destination> to IDictionary<string, IDestination>. The IDestination interface is common to the Destination and ActionGoTo classes.
- PdfLang type is removed, and all properties that previously had that type are now of the type string.
- ICCProfile.IsSame() method was removed.

- GrapeCity.Documents.Pdf.Annotations.UnknownAnnotation.Subtype property type changed from string to PdfName.
- PdfDictWrapperObject class was removed; use PdfDictWrapper instead.
- CheckBoxField.ValueObj property behavior changed; now it returns a Boolean true if all widgets of the CheckBoxField use the same name for the appearance stream used to display a checked state. Previously, it returned "Off" if the field was unchecked or the name of the checked appearance stream. Use PdfValue property to get the PdfName used to hold the value.

New Features and Improvements

- GcPdfDocument.Load() method now implements lazy loading of PDF objects. This provides several advantages, including improved compatibility with PDFs produced by various vendors, the preservation of custom content unknown to DsPdf as is, and improved average load speed.
- Added the ability to open or modify (with limitations) a password-protected PDF without specifying the password; see GcPdfDocument.Load(stream, DecryptionOptions) overload.
- Added helper methods to TextBoxField, ComboBoxField, and CombTextField: SetPercentFormat(), SetPercentValue(), SetNumberFormat(), SetNumberValue(), SetDateFormat(), SetDateValue(), SetTimeFormat(), SetTimeValue(), SetSpecialFormat(), and SetSpecialFormatValue().
- Added the ability to generate custom time-stamp tokens. The new ITimeStampGenerator interface can be implemented by the user and assigned to SignatureProperties.TimeStamp and TimeStampProperties.TimeStamp.
- Added following helper classes in GrapeCity.Documents.Layout.Composition namespace to simplify drawing layouts on a GcGraphics:
  - Surface Class: Represents a surface that can draw its views on a GcGraphics.
  - Layer Class: Represents a drawing layer with visuals, optional clipping, and nested layers.
  - View Class (derived from Layer Class): Represents a Layer with an associated LayoutView object and transformation.
  - Space Class: Represents a space on a Layer with an associated LayoutRect.
  - Visual Class (derived from Space Class): Represents a figure, text, or image on a Layer.

Resolved Issues

- Fixed several issues related to errors when opening existing PDFs.

Version 6.1.0

Breaking Changes

- Type of property GrapeCity.Documents.Pdf.Recognition.Structure.Element.Children changed from IList<Element> to IReadOnlyList<Element>.
- Type of property GrapeCity.Documents.Pdf.Recognition.Structure.Element.ContentItems changed from IList<Element> to IReadOnlyList<Element>.

New Features and Improvements

- Added InterpolationMode property in SaveAsImageOptions class that specifies the sampling mode to use when drawing images with resizing.
- Added ClonePage() method in PageCollection class that clones the page at a specified index and inserts it at a specified position.
- Added GrapeCity.Documents.Pdf.Recognition.Structure.Element.HasItems property that gets a value indicating whether the current element has any children.
- Added GrapeCity.Documents.Pdf.Recognition.Structure.Element.Items property that gets the list of the current element's child elements.
- Added GrapeCity.Documents.Pdf.Recognition.Structure.LogicalStructureItem class, which is the base abstract

class for items in a LogicalStructure.
- Added GrapeCity.Documents.Pdf.AcroForms.CheckBoxField.Opt property that specifies export values for the widget annotations' constituent check boxes.
- Added following methods to GrapeCity.Documents.Pdf.AcroForms.RadioButtonField class:
  - GetCheckedAppearanceStreamNames(): Gets the names of the appearance streams of the widget annotations associated with this field that are used to display the widgets in checked state.
  - GetCheckedAppearanceStreamName(): Gets the name of a widget annotation's appearance stream that is used to display the widget in the checked state.
  - SetCheckedAppearanceStreamName(): Sets the name of a widget annotation's appearance stream that will be used to display the widget in the checked state.
  - SetCheckedAppearanceStreamNames(): Sets the name of an appearance stream that will be used by the widget annotations associated with this field to display the widgets in checked state.
- Added new classes to GrapeCity.Documents.Imaging package (classes in GrapeCity.Documents.Layout namespace and the TableRenderer class) can be used on GcPdfGraphics. See the GrapeCity.Documents.Imaging package for details.
- If the viewState argument is null in a GcPdfDocument.SaveAs*() call, a new default ViewState will be created and used.

Resolved Issues

- Miscellaneous minor bug fixes.

Version 6.0.0

New Features and Improvements

- Added BuildRichTextAppearanceStreams property in GcPdfDcoument class to generate appearance stream for free text and line annotations with rich text content.
- Added AnnotationBase.RemoveAppearance method to remove all appearance streams assiciated with the current annotation.
- Added support for new class GcHTMLBrowser for converting HTML markup to PDF.
- Added other supporting classes such as BrowserFetcher, HtmlPage, LaunchOptions, PageOptions, TimeoutOptions.
- Added another class, GcPdfGraphicsExt is added to provide extension methods for rendering HTML to GcPdfGraphics.
- Added appearance streams for all the annotation types except for rich texts in free text and line annotations.
- Replace custom-built Chromium with Chrome or Edge which is installed in OS, or you can now download chromium from a public website as well.

Version 5.2.0.800

New Features and Improvements

- Added WidgetAnnotation.RotationAngle property that specifies the angle (in degrees) of rotation for widget annotation.
- Added ability to find and replace, and find and delete text in PDF.
- Added GcPdfDocument.ReplaceText() method that replaces a specified text on all pages of the current document.
- Added Page.ReplaceText() method that replaces a specified text on the current page.
- Added ITextMap.ReplaceText() method that replaces a specified text fragment with another text.
- Added DeleteTextMode enum that specifies how deleting text would affect the surrounding content.
- Added GcPdfDocument.DeleteText() method that deletes a specified text from all pages of the current

document.
- Added Page.DeleteText() method that deletes a specified text from the current page.
- Added ITextMap.DeleteText() method that deletes a specified text fragment.
- Added StandardSecurityHandler.PasswordMatches property which indicates how the password specified when loading the PDF matches the document's User and Owner passwords.

Changes From the Previous Version

- StandardSecurityHandler.HasOwnerPassword and HasUserPassword properties marked as obsolete. Use the new property PasswordMatches instead.

Resolved Issues

- Resolved an issue where a standard font was incorrectly embedded and could not be used in a specific scenario.
- Resolved an issue where redact incorrectly erased additional characters in some cases.

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.CreatRegex().

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 occured 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.SetGcImagingLicenseKey() and instance GcHtmlRenderer.ApplyGcImagingLicenseKey() 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, DsPdf 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 DsPdf 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, DsPdf 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! DsPdf 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 PdfAConformaceLevel Metadata.PdfA and GcPdfDocument.PdfAConformaceLevel 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.

DsPdfViewer Release Notes

Refer to the following release notes for the major releases of DsPdfViewer.

For details about latest hot fixes, see the nuget page.

Version 7.1.0

New Features and Improvements

The following features have been added to this version of the product:

- Fixed the issue of not submitting a comment by pressing the Enter key on the 'Done' button.
- Fixed the issues with loading large files.
- Fixed the issue of not deactivating Edit mode when switching the viewer layout from the annotation editor while the second toolbar is open.
- Fixed the issue of highlighting the space before the found text when searching for a text.
- Enhanced the behavior and appearance of the context menu and added the ability to add a note to selected text.
- Removed the text markup menu that automatically popped up on text selection. The main context menu includes that menu as a submenu.
- Added the `showContextMenuOnSelection` option to control the context menu behavior when text is selected. The option has the following values:
  - "Auto": Automatically determines whether to show the context menu based on the device type.
  - "On": Always shows the context menu when text is selected.
  - "Off": Never shows the context menu when text is selected.
    The default value is "Auto." On systems with mice, "Auto" behaves like "Off." On small devices (phones, tablets) without mice, "Auto" behaves like "On.".
- Updated context menu texts in the Reply Tool. (DOC-6014)
- Added support for rich media annotations (play embedded audio or video, modify rich media annotations).
- Reply Tool Improvements:
  - Added the ability to delete a comment item using the Delete key. Enabled navigation through comment items using the TAB and Arrow keys.
  - Added the ability to resize the right sidebar element.
  - The Reply Tool now activates automatically when a markup annotation or text comment is added via the context menu, focusing on the new comment in the list.

- The color for ReplyTool icons has been removed.
- Split ISupportApi into base and multi-user parts and added the ability to specify a custom implementation for SupportApi that conforms to the ISupportApiBase interface.

Version 7.0.0

Important Note

This is the initial release of the @mescius/dspdfviewer package. It replaces @grapecity/gcpdfviewer, and provides the same functionality, ensures future enhancements, and is backwards compatible with @grapecity/gcpdfviewer. Existing subscriptions will continue to work with DsPdfViewer.

New Features and Improvements

The following features have been added to this version of the product:

- Added DsPdfViewer class and @mescius/dspdfviewer package. Functionally DsPdfViewer is identical to GcPdfViewer.
- Added floating text search bar.
- Added localization resources and a localization example to the DsPdfViewer build.
- Added new option useFloatingSearchBar: enable a floating search bar instead of the sidebar search panel. The default value is true.
- Updated some tooltips.

Version 4.2.0

New Features and Improvements

The following features have been added to this version of the product:

- Added the ability to use an async function as a "beforeFormSubmit" handler.
- Added the ability to specify HTML content for document list items.
- Added new methods: getSharedDocuments, openSharedDocumentByName, and openSharedDocumentByIndex.

Bug Fixes

The following issues have been resolved since the last release:

- Fixed the issue where content of some password-protected PDF documents is not displayed.
- The name of the document list item changed from lower case to upper case.
- Fixed the issue where PDF text cannot be selected when zooming in on the document.
- Miscellaneous UI improvements.

Version 4.1.0

New Features and Improvements

The following features have been added to this version of the product:

- Added PDF Organizer button that allows users to rearrange, duplicate or remove pages of a PDF, or merge PDFs.
- Added support for PDF's initial view settings (hide toolbars or menus, open with a specific page layout, etc.).
- Added ignoreInitialView option that ignores initial view settings specified in PDFs when it is set to true.
- Added support for zoom-dependent optional content (layers).
- Added the ability to specify a custom progress message during the save action.
- Added resolvePageIndex() method that resolves the page index using PDF page references.
- Added holdToPan action that temporarily enables the pan tool when you press and hold the spacebar.
- The "Layers" and "StructureTree" panels are added to the default sidebar layout, and the "Articles" panel is removed. The default set of sidebar panels is now as follows: "Thumbnails," "Search," "Outline," "Layers," "StructureTree," and "Attachments."
- The printable flag is now set to true for all new annotations.
- Returning model type for the viewer.viewerPreferences property changed: all property names are now camelCased. New properties are added: openAction, pageMode, and pageLayout.
- Reduced the heights of dialogs' title bars.
- The way text annotation (sticky note) properties display has been improved.
- The hideAnnotationPopups option now allows specifying the types of annotations that won't show popups. Possible values are: ['Text', 'Link', 'Line', 'Square', 'Circle', 'Polygon', 'PolyLine', 'Ink', 'Popup', 'FileAttachment', 'Sound', 'Redact', 'Stamp'] or true or All (true and All have the same behavior).
- The save() method now allows specifying a range of pages or ranges of pages to save, changing page order, or duplicating pages.
- The saveAsImages() method now allows specifying the zoom factor.

Bug Fixes

The following issues have been resolved since the last release:

- Fixed the issue where, in a PDF with different page sizes, the 'move to next page' option was not working correctly.
- Fixed the issue where a locked stamp annotation can still be edited in DsPdfViewer.
- Fixed the issue where you could not undo changes if the editor layout was not activated.
- Fixed the issue where a popup annotation may move unexpectedly.
- Fixed the issue where you could not navigate between pages using page up and page down keys in single page view.
- Fixed the issue where the list of layers is not shown correctly in some cases.

Version 4.0.0

New Features and Improvements

The following features have been added with this version of the product.

- Added toolbar items and context menu support for text markup annotations where highlight, underline, strikeout, and squiggly buttons are added.
- Added support to resize line annotations.
- Added ability to position and drag the signature between pages using the mouse.
- Added disableFeatures option used to disable certain features of the DsPdfViewer depending on company security policy.
- Added “Ear” icon for sound annotations.
- Added support to save the PDF current document as image ‘PNG’ by adding “Save current document as

images" button to the toolbar.
- Modified icons like text, sound, and file attachment to match them like Acrobat Reader. However, sound and file attachment icons are no longer resizable.
- Modified the layout of "Text" quick editing tools to add new text markup annotations.

Bug Fixes

The following issues have been resolved since the last release.

- Fixed the issue where appearance of Annotation borders is modified.
- Fixed the issue where visibility of line annotation is handled.
- Fixed the issue where annotation icons are corrected in case annotations were getting converted into content.
- Fixed the issue of image resize while stamp annotation is rotated.
- Fixed the issue where stamp annotation can now be added in rotated document.
- Fixed the issue where an error "operation is not valid due to the current state of the object" occurred when a stamp was attached to a PDF.

Version 3.2.0

New Features and Improvements

- Added the ability to change current user name from the UI.
- Added the ability to change the minimum/maximum zoom factor using the zoomOptions option.
- Updated tooltips for Save/download button.

Bug Fixes

- Fixed the issue where date fields did not work correctly in form filler dialog.
- [iOS]Fixed the issue where the print preview was empty.
- [Windows Touch]Fixed the issue where Reply tool button did not respond to touch.
- [Windows Touch]Fixed the issue where thumbnail pane does not work correctly.
- [iOS Desktop mode]Fixed the issue where custom input types display was incorrect.
- [iOS Desktop mode]Fixed the issue where read-only fields still showed the dropdown menu.
- Fixed the issue where "Draw polygon annotation" button could be pushed after canceling the draw operation, but did not work.

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 DsPdfViewer.
- Added the new gcPdfVersion property, which gets the version of DsPdf 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 DsPdfViewer 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 DsPdf 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 DsPdfViewer.
- 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.