1 Overview

ComPDFKit for Web is a robust online PDF viewer library for developers who need to develop applications on the Web, which offers powerful JavaScript APIs for quickly viewing and editing any PDF online. It is feature-rich and battle-tested, making PDF files process and manipulation easier and faster for your online project.

1.1 Key Features

Viewer component offers:

- Standard page display modes, including Scrolling, Double page, and Cover mode.
- Navigation with thumbnails, outlines, and layers.
- Text search & selection.
- Zoom in and out.
- Switch between different themes, including light mode and dark mode.

Annotations component offers:

- Create, edit, and remove kinds of annotations, including Notes, Free Text, Lines, Arrow, Squares, Circles, Ink, and Stamps.
- Support for annotation appearances.
- Import and export annotations to/from XFDF.

Forms component offers:

- Create, edit and, remove form fields, including Text Fields, Check Boxes, Radio Button, List Box, Combo Box, Push Button, and Signatures.
- Fill out PDF Forms.

Signatures component offers:

- Drawn signatures.
- Image signatures.
- Typed signatures.

Security component offers:

- Encrypt and decrypt PDFs, Password can be set to protected the PDF.

Document Comparison component offers:

- Overlay Comparison to superpose two files and show the differences by different colors.

Content Editor component offers:

- Programmatically add and remove text in PDFs and make it possible to edit PDFs like Word. Allow selecting text to copy, resize, change colors, text alignment, and the position of text boxes.

1.2 License

ComPDFKit for Web is a commercial SDK, which requires a license to grant developer permission to release their apps. Each license is only valid for one root domain. Other flexible licensing options are also supported, please contact our marketing team to know more. However, any documents, sample code, or source code distribution from the released package of ComPDFKit to any third party is prohibited.

2 Get Started

It is easy to embed ComPDFKit in your web app with a few lines of JavaScript code. Take just a few minutes and get started.

The following sections introduce the structure of the installation package, how to run a demo, and how to make a web app with ComPDFKit for Web.

2.1 Requirements

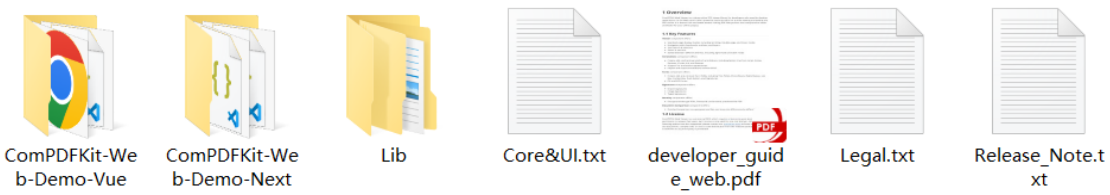
To integrate ComPDFKit for Web in your browser, you must have a development environment and a browser.

- The latest stable version of Node.js.
- ComPDFKit for Web supports most mainstream browsers, and it's better to use the latest version. IE browser is not supported currently.

2.2 Package Structure

The package of ComPDFKit for Web includes the following files.

- ***"ComPDFKit-Web-Demo"*** - A folder containing Web sample projects.
- ***"Lib"*** - Include the library of ComPDFKit for Web.
- ***"developer_guide_web.pdf"*** - Developer guide and API reference.
- ***"Core&UI.txt"*** - Third-party code usage agreement.
- ***"Legal.txt"*** - Legal and copyright information.
- ***"Release_Note.txt"*** - Release information.



2.3 How to run a demo

ComPDFKit for Web provides one demo for developers to learn how to call the SDK on the Web. You can find them in the ***"ComPDFKit-Web-Demo"*** folder. This guide will show you how to run it in **VSCode**.

1. Open the **Demo** project in **VSCode**.
2. Install all dependencies that are needed in the **Demo**.

```
npm install
```

3. Run ***"ComPDFKit-Web-Demo"*** in the development environment.

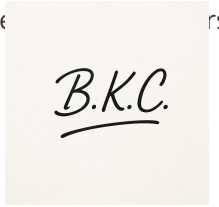
```
npm run dev
```

4. PDF will be opened and displayed.

2.4 How to Make a Web App in JavaScript With ComPDFKit for Web

This section will help you to quickly get started with ComPDFKit for Web to make a Web app with step-by-step instructions. Through the following steps, you will get a simple web application that can display the contents of a specified PDF file.

Before you start the following steps, create a ne rst.



2.4.1 Add ComPDFKit for Web

1. Add the ***""@compdfkit"*** folder in the **lib** directory to the **root** directory or **assets** directory of your project. This will serve as the entry point for the ComPDFKit for Web and will be imported into your project.
2. Add the ***"webviewer"*** folder that contains the required static resource files to run the ComPDFKit Web demo, to your project's static resource folder.

2.4.2 Display a PDF Document

1. Import the "***webviewer.js***" file in the ***"@compdfkit"*** folder into your project.
2. Initialize the ComPDFKit for Web in your project by calling `ComPDFKitViewer.init()`.
3. Pass the PDF address you want to display and your license key into the `init` function.

```
// Import the JS file of ComPDFKit Web Demo.
import ComPDFKitViewer from "/@compdfkit/webviewer";

const viewer = document.getElementById('webviewer');
ComPDFKitViewer.init({
  pdfUrl: 'Your PDF Url',
  license: 'Input your license here'
}, viewer)
.then((core) => {
  const docViewer = core.docViewer;
  docViewer.addEvent('documentloaded', () => {
    console.log('ComPDFKit Web Demo');
  })
})
```

Note: *You need to contact ComPDFKit team to get the **license**.*

4. Once your project is running, you will be able to see the PDF file you want to display.

2.4.3 Display a PDF Document with ComPDFKit Server

```
// Import the JS file of ComPDFKit Web Demo.
import ComPDFKitViewer from "/@compdfkit/webviewer";

const viewer = document.getElementById('webviewer');
ComPDFKitViewer.init({
  pdfUrl: 'Your PDF Url',
  license: 'Input your license here',
  webviewerServer: 'Server'
}, viewer)
.then((core) => {
  const docViewer = core.docViewer;
  docViewer.addEvent('documentloaded', () => {
    console.log('ComPDFKit Web Demo');
  })
})
```

2.5 How to Make a Next.js App With ComPDFKit for Web

2.5.1 Get Started with the Next.js Demo

1. Open the **ComPDFKit-Web-Demo-Next** project in **VSCode**.
2. Install all dependencies that are needed in the **Demo**.

   ```
   npm install
   ```

3. Run **ComPDFKit Web Demo** in the development environment.

   ```
   npm run dev
   ```

4. PDF will be opened and displayed.

2.5.2 Make your Next.js App With ComPDFKit for Web

1. Import the **@compdfkit** folder into root of your project.
2. Initialize the ComPDFKit for Web in your project by calling `ComPDFKitViewer.init()`.
3. Pass the PDF address you want to display and your license key into the `init` function.

```
import ComPDFKitViewer from "/@compdfkit/webviewer";

export default function Home() {
  const viewer = useRef(null);
  let docViewer = null

  useEffect(() => {
    ComPDFKitViewer.init({
```

```
      pdfUrl: 'Your PDF Url',
      license: 'Input your license here'
    }, viewer.current).then((core) => {
      docViewer = core.docViewer;
      docViewer.addEvent('documentloaded', () => {
        console.log('ComPDFKit Web Demo');
      })
    })
  });

  return (
    <div id="webviewer" ref={viewer}></div>
  )
}
```

4. PDF will be opened and displayed.

2.5.3 Save a document

```
// Import the JS file of ComPDFKit Web Demo.
import ComPDFKitViewer from "/@compdfkit/webviewer";

const viewer = document.getElementById('webviewer');
ComPDFKitViewer.init({
  pdfUrl: 'Your PDF Url',
  license: 'Input your license here'
}, viewer)
.then((core) => {
  const docViewer = core.docViewer;
  docViewer.addEvent('documentloaded',async () => {
    console.log('ComPDFKit Web Demo');

    const docStream = await docViewer.download()
    const docBlob = new Blob([docStream], { type: 'application/pdf' })
  })
})
```

2.6 UI Customization

2.6.1 Customize the Navigation Bar

One key part of customizing the UI is the customization of the navigation bar. There are several ways you may want to customize the navigation bar. To name a few:

- Add a custom save button.
- Remove existing tools from the navigation bar.
- Rearrange the feature area.
- Hide the text field tool in the sub-menus of the forms feature and move them to the right side of the navigation bar.

ComPDFKit for Web's WebViewer UI provides flexible APIs for easy customization of the navigation bar.

To understand the structure of the navigation bar and the different types of tools, you can read **The Composition of the Navigation Bar** and **Navigation Bar Tools**. You can also jump directly to the **UI Customization Samples** to see examples of customizing the navigation bar.

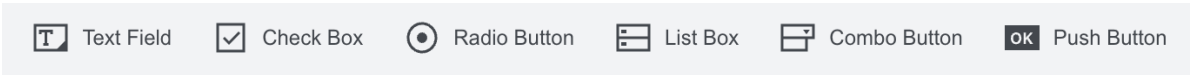
2.6.2 The Composition of the Navigation Bar

The navigation bar is mainly divided into three parts: Navigation Tools, Feature Area, and the sub-menus of corresponding features (including sub-toolbars).

- **Navigation Tools**:
  The icon buttons on the left and right sides of the image below (Navigation Bar) are the Navigation Tools.

- **Feature Area**:
  The part inside the feature box in the middle of the navigation bar is called the Feature Area (as shown in the image below). Each feature has a different **Sub-Menus**. Buttons in both the **Feature Area** and the **Sub-Menus** can be modified individually.

- **Sub-Menus**:
  Under different **Feature Areas**, there are corresponding **Sub-Menus** showing a group of tools for that feature. ComPDFKit for Web provides various APIs to customize the tools within this **Sub-Menus**. Below, the **Sub-Menus** for annotations and forms are shown:



2.6.3 Switching Feature Areas

You can call `setActiceToolMode` to switch between the different feature modes in the **Feature Area**.

```
ComPDFKitViewer.init(...)
  .then(instance => {
    const { UI } = instance;
    UI.setActiceToolMode('toolMenu-Annotation');
  })
```

2.6.4 Hide/Show Navigation Bar Elements

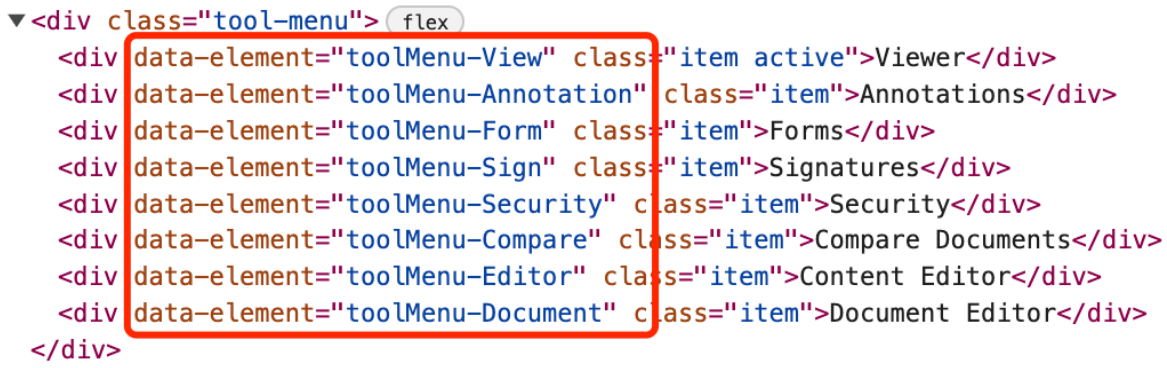
Find the Data Element Attribute of the Navigation Bar

To hide or show navigation bar elements, you first need to find the element's `data-element` attribute in the DOM inspector.

For Example:

We can see that its `data-element` value is `themeMode`. Now we can use this value to hide or show it.

The image below is the `data-element` attributes of the Toolbar of the **Feature Area**.



Hiding/Showing Navigation Bar Elements

- Hide/Show the annotation button in the **Feature Area**:

```
// Hide the annotation button in the feature area.
UI.disableElements('toolMenu-Annotation');
// Show the annotation button in the feature area.
UI.enableElements('toolMenu-Annotation');
```

- Hide/Show multiple navigation bar elements:

```
// Hide the left panel button and the right panel button.
UI.disableElements(['leftPanelButton', 'rightPanelButton']);
// Show the left panel button and the right panel button.
UI.enableElements(['leftPanelButton', 'rightPanelButton']);
```

2.6.5 Properties and Types of Navigation Tools

Navigation Tools are objects with certain properties. You can add, remove, or move Navigation Tool buttons by calling `setHeaderItems`. If you would like to add custom tools to the navigation bar, it is crucial to understand what types of custom tools you need to add and which attributes to use. Below are the properties and types for **Navigation Tools**.

1. Set the Header

Parameters for the `setHeaderItems` callback function: `header` object, can invoke `get`, `getItems`, `shift`, `unshift`, `push`, `pop`, `delete`, and `update` to perform corresponding operations on navigation tools and the toolbar.

```
UI.setHeaderItems(function(header) {
  // Get all feature area.
  const items = header.getHeader('tools').getItems();
  console.log(items);
});
```

2. Action Button

An action button can trigger an action. This kind of button has no active state. Its properties include:

- **name (string)** - Must be set to `customButton`.
- **type (string)** - Must be set to `actionButton`.

- **img (string)** - Path to an image or base64 data.
- **onClick (function)** - Function to be triggered on click.
- **title (string, optional)** - Tooltip of the button.
- **dataElement (string, optional)** - Option to set `data-element` value of the button element. It can be used to disable/enable the element.
- **dropItem (boolean, optional)** - Option set if the item is a dropdown.
- **text (string, optional)**- Effective when `dropItem` is true. Set the text displayed for the dropdown option.

```
const newActionButton = {
  name: 'customButton',
  type: 'actionButton',
  img: 'path/to/image',
  onClick: () => {
    alert('Hello world!');
  },
  dataElement: 'alertButton'
};
```

3. State Button

The state button is a customizable button. You can decide how many states it has, what state is active, and when to update the state. Its properties include:

- **name (string)** - Must be set to `customButton`.
- **type (string)** - Must be set to `statefulButton`.
- **initialState (string)** - String that is one of `states` object's keys.
- **states (object)** - The shape of the object looks like: { nameOfState1: state1, nameOfState2: state2, ... }
- The properties of **states** include:
  - **img (string, optional)**: Path to an image or base64 data.
  - **getContent (function, optional)**: Function to be called when you update the state. Define this property if you don't use the img property for this button **getContent** has a parameter `activeState`. `activeState` is an object that corresponds to the value of initialState in states.
  - **onClick (function)**: A function that is triggered when clicked. **onClick** has a parameter `activeState`. `activeState` is an object that corresponds to the value of **initialState** in states.
  - More properties you need.
- **mount (function)** - Function to be called after the button is mounted into DOM.
- **unmount (function, optional)** - Function to be called before the button is unmounted from DOM.
- **title (string, optional)** - Tooltip of the button.
- **dataElement (string, optional)** - String to set `data-element` value of the button element.
- **dropItem (boolean, optional)** - Option set if the item is a dropdown.
- **text (string, optional)** - Effective when `dropItem` is true. Set the text displayed for the dropdown option.

Examples:

A stateful button that shows the count. When you click it, it will increment the counter by 1.

```
const countButton = {
  name: 'customButton',
  type: 'statefulButton',
  initialState: 'Count',
  states: {
    Count: {
      number: 1,
      getContent: activeState => {
        return activeState.number;
      },
      onClick: activeState => {
        activeState.number += 1;
      }
    }
  },
  dataElement: 'countButton'
};
```

A state button showing the current page number. When you click it, the document will go to the next page. If you are on the last page, the document will turn to the first page.

```
const nextPageButton = {
  name: 'customButton',
  type: 'statefulButton',
  initialState: 'Page',
  states: {
    Page: {
      getContent: Core.getCurrentPage,
      onClick: activeState => {
        const currentPage = Core.getCurrentPage();
        const totalPages = Core.getPagesCount();
        const atLastPage = currentPage === totalPages;

        if (atLastPage) {
          Core.previousPage();
        } else {
          Core.nextPage();
        }
        activeState.getContent = Core.getCurrentPage();
      }
    }
  },
  mount: () => {
    // Execute after mounting.
    console.log('Mounted.');
  },
  unmount: () => {
    // Execute before destruction.
    console.log('Destroyed.');
  },
  dataElement: 'nextPageButton'
};
```

4. Toggle Element Button

The toggle element button is used to open/close a specified UI element. When the UI element is open, the button is in an active state.

Its properties include:

- **name (string)** - Must be set to `customButton`.
- **type (string)** - Must be set to `toggleButton`.
- **img (string)** - The path to an image or base64 data.
- **element (string)** - The property value of the `data-element` UI element to be opened/closed.
- **title (string, optional)** - Tooltip text for the button.
- **dataElement (string, optional)** - Set the option for the `data-element` button element value. It can be used for showing/hiding that element.

```
const newToggleButton = {
  name: 'customButton',
  type: 'toggleButton',
  img: `path/to/image`,
  element: 'pageModePanel',
  dataElement: 'pageModePanelButton'
};
```

5. Tool Button

The tool button is the button in the sub-menus under the feature module. For example, in the Form feature mode, you can create a button for a text field function by customizing a tool button and specifying its element value as `textfield`. You can place the tool buttons you need to customize anywhere. When the tool is activated, the button is in an active state.

Its properties include:

- **name (string)** - Must be set to `customButton`.
- **type (string)** - Must be set to `toolButton`.
- **toolName (string)** - The `data-element` button element value of the target tool.

```
const newToolButton = {
  name: 'customButton',
  type: 'toolButton',
  toolName: 'textfield'
};
```

6. Spacer

The spacer is just a `div` with a CSS attribute of `flex: 1`, used to occupy any remaining space in the tool. It is used to push buttons to each side of the default title.

Its properties include:

- **type (string)** - Must be set to `spacer`.

```
const newSpacer = {
  type: 'spacer'
};
```

7. Divider

Divider renders a vertical bar with some margin to separate item groups.

Its properties include:

- **type (string)** - Must be set to `divider`.

```
const newDivider = {
  type: 'divider'
};
```

2.6.6 Examples of Custom UI

Add a custom save button:

```
UI.setHeaderItems(function(header) {
  const mySaveButton = {
    name: 'customButton',
    type: 'actionButton',
    dataElement: 'mySaveButton',
    img: `<svg width="14" height="14" viewBox="0 0 14 14" fill="none"
xmlns="http://www.w3.org/2000/svg">
            <path d="M13 1L1 13" stroke="#BABABA" stroke-width="2" stroke-
linecap="round" stroke-linejoin="round"/>
            <path d="M1 1L13 13" stroke="#BABABA" stroke-width="2" stroke-
linecap="round" stroke-linejoin="round"/>
          </svg>`,
    onClick: function() {
      Core.download();
    }
  }
  header.push(mySaveButton);
});
```

Hide the text field tool in the form function area and put it on the right side of the navigation bar:

```
UI.setHeaderItems(header => {
  header.headers.toolItems.form.splice(0, 1);

  const myTextFieldButtom = {
    name: 'customButton',
    type: 'toolButton',
    toolName: 'textfield'
  }
  header.getHeader('rightHeaders').unshift(myTextFieldButtom);
});
```

Reorder the function area:

```
UI.setHeaderItems(header => {
  // Reverse the order of the function area.
  header.headers.tools.reverse();
  // Sort by the names of the function areas in alphabetical order.
  header.headers.tools.sort((a, b) => a.element.localeCompare(b.element));
});
```

Remove existing tools from the navigation bar:

```
UI.setHeaderItems(header => {
  // Remove tools on the left side of the navigation bar.
  header.getHeader('headers');
  header.update([]);
  // Remove function area.
  header.getHeader('tools').update([]);
});
```

3 Guides

If you're interested in the ComPDFKit features, please go through our guides to quickly add PDF viewing, annotating, editing, form filling, and signing to your application. The following sections list some examples to show you how to add document functionalities to your Web apps using our JavaScript APIs.

3.1 Viewer

3.1.1 Overview

ComPDFKit for Web includes a high-quality PDF viewer that's fast, precise, and feature-rich. It offers developers a way to quickly embed a highly configurable PDF viewer in any Web application.

Benefits of ComPDFKit PDF Viewer

- **Display Modes**: Freely switch between single-page or double-page view, page flipping or scrolling modes, adapting to different reading scenarios.
- **Multiple Themes:** Choose themes suitable for work, night reading, prolonged screen time, or creating custom themes.
- **PDF Navigation:** Navigate directly to specific locations, or use bookmarks and outlines.
- **Extensibility:** Easily add features such as annotations, forms, signatures, etc., to PDF viewer.
- **Fast UI Integration:** Achieve rapid integration and customization through extensible UI components.

Guides for Viewer

- Display Modes

  Choose between single-page, double-page, book, flip, or scroll modes, or set cropping and split-view modes.

- Page Navigation

  Efficiently navigate to different locations in the document through simple code.

- Text Search and Selection

  Locate the position of keywords in the document, and copy or mark the text of interest.

- Zooming

  Adjust the degree of zoom in or zoom out of a page in a PDF document to fit the user's visual preferences or device screen size.

- Set Language

  Set the language of UI.

3.1.2 Display Modes

ComPDFKit for Web provides viewing PDFs in different display modes. Let's see what the display modes are and call the following method to display in different display modes. The corresponding parameters are shown in the following table.

- Single Mode

  Show one page at a time for users to continuously scroll up and down to navigate through the pages.

- Double Page

  Display two pages side-by-side and continuously scroll up and down to navigate through the pages.

- Cover Mode

  Show the cover page on the first page during two-up.

  ```
  docViewer.webViewerSwitchSpreadMode(mode)
  ```

Name	Required	Type	Description
mode	yes	number	Display Modes Parameter, Single Mode: 0, Two-up Mode: 1, Cover Mode: 2

- Full Screen Mode

  It's a mode to view PDFs in full-screen, which fills the entire screen with the PDF content. The following method is given to start the full-screen presentation mode.

  ```
  docViewer.requestFullScreenMode()
  ```

3.1.3 PDF Navigation

Provide thumbnails, bookmarks, and layers to navigate PDF content.

- Page Navigation

  After loading a PDF document, you can programmatically interact with it, which allows you to view and transition between the pages like scrolling and jumping to specific pages.

  ```
  // Previous page.
  docViewer.previousPage()

  // Next page.
  docViewer.nextPage()

  // Jump to a page.
  docViewer.pageNumberChanged('3')
  ```

- Scroll

To scroll to specific distances in the horizontal and vertical directions, you can use this function.

```
// Top and left are optional.
docViewer.scrollTo({
  top: 200,
  left: 100
})
```

- Scroll Mode

  ComPDFKit for Web has two scrolling modes: Vertical and Horizontal. You can set the corresponding mode by using the following method.

  ```
  docViewer.webViewerSwitchScrollMode(mode)
  ```

  This is the parameters to apply different scroll modes.

Name	Required	Type	Description
mode	yes	number	Scroll Mode Parameter, Vertical: 0, Horizontal: 1

- Thumbnails

  Provide methods to render PDF pages as thumbnail images.

3.1.4 Text Search & Selection

ComPDFKit for Web offers developers an API for programmatic full-text search, as well as UI for searching and highlighting relevant matches. You can use the method below to search for content in PDFs with ComPDFKit for Web.

```
docViewer.search(value)
```

Name	Required	Type	Description
value	yes	string	The Searched Content

3.1.5 Zooming

ComPDFKit for Web provides super zoom out and in to unlock more zoom levels, and pinch-to-zoom or double tap on the specific area to perform a smart page content analysis, or you can programmatically interact with it by using the following method.

Use the following methods to zoom in/out by a certain zoom value or scale.

```
// Zoom in.
docViewer.zoomIn()

// Zoom out.
docViewer.zoomOut()

// Scale changed.
docViewer.webViewerScaleChanged(scale)
```

Name	Required	Type	Description
scale	yes	number	Zoom Scale, Zoom In: 10, Zoom Out: 0.5

3.1.6 Set Language

ComPDFKit for Web supports setting the language of UI.

```
// Import the JS file of ComPDFKit Demo.
import ComPDFKitViewer from "/@compdfkit/webviewer";

const viewer = document.getElementById('webviewer');
ComPDFKitViewer.init({
  pdfUrl: 'Your PDF Url',
  license: 'Input your license here'
}, viewer)
.then((core) => {
  const docViewer = core.docViewer;
  const { UI } = core;
  docViewer.addEvent('documentloaded', () => {
    console.log('ComPDFKit Web Demo loaded');

    const language = 'zh-CN';
    UI.setLanguage(language);
  })
})
```

Name	Required	Type	Accepted Values	Description
language	yes	string	zh-CN / en	languages of WebViewer

3.2 Annotations

3.2.1 Overview

Annotations allow users to highlight paragraphs, add comments, markup, sign, or stamp PDF documents without modifying the original author's content. The annotated content, along with the original text, can then be shared together.

Benefits of ComPDFKit PDF Annotation

- **Comprehensive Type Support:** Enables highlighting, text, freehand drawing, shapes, stamps, and more.
- **Create, Edit, Delete:** Perform creation, editing, and deletion operations either programmatically or directly through the UI.
- **Flattened Annotations:** Embed annotations permanently onto the document as images, ensuring document appearance stability and preventing further modifications.
- **Annotation Events:** Trigger specified workflows to achieve automation.
- **Annotation Import and Export:** Export annotations as XFDF templates and apply them to multiple documents.
- **Fast UI Integration:** Achieve rapid integration and customization through extensible UI components.

Guides for Annotations

- Create Annotations

  Generate annotations of various types.

3.2.2 Annotation Types

ComPDFKit for Web supports annotation types below. All annotations we supported are standard annotations (as defined in the PDF Reference) that can be read and written by many apps, such as Adobe Acrobat.

- Note
- Link
- Free text
- Shapes: Square, circle, line, and arrow
- Markup: Highlight, underline, strikeout, and squiggly
- Ink
- Stamp: Nearly 20 standard stamps and dynamic stamps

3.2.3 Create & Edit Annotations

ComPDFKit for Web includes a wide variety of standard annotations, and each of them is added to the project in a similar way. Before creating annotations, you need to initialize a pdf document in your project.

Note: *When adding an annotation, the coordinate origin is at the left top corner of the page.*

```
// Import the JS file of ComPDFKit Demo.
import ComPDFKitViewer from "/@compdfkit/webviewer";

const viewer = document.getElementById('webviewer');
ComPDFKitViewer.init({
  pdfUrl: 'Your PDF Url',
  license: 'Input your license here'
}, viewer)
.then((core) => {
  const docViewer = core.docViewer;
  docViewer.addEvent('documentloaded', () => {
    console.log('ComPDFKit Web Demo loaded');
  })
```

```
})
```

- Note

  Add a sticky note (text annotation) to a PDF Document page by using the following method.

```
docViewer.addAnnotations({
  type: 'text',
  pageIndex: 1,
  color: '#FF0000',
  fontSize: 16,
  fontName: 'Helvetica',
  opacity: 1,
  contents: 'test',
  rect: {
    left: 100,
    top: 30,
    right: 124,
    bottom: 54
  },
})
```

- Link

  To add a hyperlink or intra-document link annotation to a PDF Document page by using the following method.

```
// Add a hyperlink.
docViewer.addAnnotations({
  type: "link",
  pageIndex: 0,
  rect: {
    left: 92,
    top: 200,
    right: 167,
    bottom: 230
  },
  url: "https://example.com"
})

// Add a intra-document link.
docViewer.addAnnotations({
  type: "link",
  pageIndex: 0,
  rect: {
    left: 92,
    top: 200,
    right: 167,
    bottom: 230
  },
  destPage: 2
})
```

- Free Text

  Add a free text annotation to a PDF Document page by using the following method.

```
docViewer.addAnnotations({
  type: 'freetext',
  pageIndex: 0,
  color: '#000000',
  fontSize: 16,
  fontName: 'Helvetica',
  opacity: 1,
  textAlignment: 'left',
  contents: 'test',
  rect: {
    left: 100,
    top: 200,
    right: 160,
    bottom: 240
  }
})
```

- Shapes

  Add a shape annotation like a rectangle, circle, line, or arrow to a PDF document page by using the following method.

```
// Square.
docViewer.addAnnotations({
  type: 'square',
  pageIndex: 0,
  borderWidth: 2,
  borderColor: '#FF0000',
  rect: {
    left: 0,
    top: 50,
    right: 100,
    bottom: 100
  }
})

// Circle.
docViewer.addAnnotations({
  type: 'circle',
  pageIndex: 0,
  borderWidth: 2,
  borderColor: '#FF0000',
  rect: {
    left: 0,
    top: 50,
    right: 100,
    bottom: 100
  }
})

// Line.
docViewer.addAnnotations({
  type: 'line',
  pageIndex: 0,
  borderWidth: 2,
  borderColor: '#FF0000',
  rect: {
```

```
    left: 0,
    top: 50,
    right: 100,
    bottom: 100
  },
  linePoints: [0,50,100,100]
})
```

- Markup

  Add a highlight annotation to a PDF Document page by using the following method, and add other markup annotations in a similar way.

  Note: *The value of `quadPoints` are (left,top), (right,top), (left,bottom), and (right,bottom). The coordinate origin is at the bottom left corner of the page.*

```
docViewer.addAnnotations({
  type: 'highlight',  // Types include highlight, underline, strikeout,
squiggly.
  pageIndex: 0,
  opacity: 0.8,
  quadPoints: [
    { PointX: 116, PointY: 300 },
    { PointX: 360, PointY: 300 },
    { PointX: 116, PointY: 360 },
    { PointX: 360, PointY: 360 },
  ],
  rect: {
    left: 116,
    top: 300,
    right: 360,
    bottom: 360
  },
  color: "#FF0000",
  contents: "test",
})
```

- Ink

  Ink is the annotation to draw freely on PDFs with kinds of colors. Follow the method below to obtain our ink annotation.

```
docViewer.addAnnotations({
  type: 'ink',
  pageIndex: 0,
  borderWidth: 2,
  opacity: 0.8,
  borderColor: '#FF0000',
  inkPointes: [[
    { PointX: 9.20, PointY: 34 },
    { PointX: 9.20, PointY: 34 },
    { PointX: 9.20, PointY: 33 },
    { PointX: 9.20, PointY: 31 },
    { PointX: 10.20, PointY: 31 },
    { PointX: 11.20, PointY: 30 },
    { PointX: 12.20, PointY: 28 },
    { PointX: 13.2, PointY: 27 },
```

```
    { PointX: 13.2, PointY: 26 },
    { PointX: 15.2, PointY: 24 },
    { PointX: 17.2, PointY: 23 },
    { PointX: 22.2, PointY: 19 }
  ]],
  rect: {
    left: 9.2,
    top: 19,
    right: 22.2,
    bottom: 34
  },
})
```

- Stamp

  Add standard and text stamps to a PDF document page by using the following method. Provide more than 20 standard stamps and dynamic time stamps.

```
// Standard stamp.
docViewer.addAnnotations({
  type: "stamp",
  pageIndex: 0,
  rect: {
    left: 205,
    top: 379,
    right: 435,
    bottom: 431
  },
  stampType: "standard",
  contents: "NotApproved"
})

// Text stamp.
docViewer.addAnnotations({
  type: "stamp",
  pageIndex: 0,
  rect: {
    left: 220,
    top: 367,
    right: 320,
    bottom: 412
  },
  stampType: "text",
  contents: "REVISED",
  time: "28/07/2023 07:28:28",
  stampColor: 1,
  stampShape: 0
})

// Create an image stamp.
const res = await fetch('https://example.com//image.png');
const imageBlob = await res.blob();
const reader = new FileReader();
reader.onloadend = async () => {
  const imageBase64 = reader.result;

  docViewer.addAnnotations({
```

```
      type: "stamp",
      stampType: "image",
      pageIndex: 0,
      imageBase64,
      rect: {
        left: 220,
        top: 367,
        right: 320,
        bottom: 412
      }
    })
}
reader.readAsDataURL(imageBlob);
```

3.3 Forms

3.3.1 Overview

The Form (or AcroForm) feature allows users to create interactive form fields in a PDF document, enabling other users to provide information by filling out these fields. Essentially, PDF form fields are a type of PDF annotation known as Widget annotations. They are utilized to implement interactive form elements such as buttons, checkboxes, combo boxes, and more.

As PDF is an electronic format, it provides advantages that traditional paper forms do not have. For instance, users can edit information that has already been entered. Additionally, document creators can distribute PDF forms over the internet, restrict the content and format entered by users, as well as programmatically extract and categorize the information filled in by users.

Benefits of ComPDFKit Forms

- **Full Types Supported:** Supports all form field types, properties, and appearance settings.
- **Create, Edit, Delete Form Fields:** Perform creation, editing, and deletion operations programmatically or directly through the UI.
- **Fill Form Fields:** Seamlessly fill form fields using the `CPDFViewer` or automatically fill them programmatically.
- **Form Events:** Trigger specified workflows, enabling automation.
- **Form Flattening:** Permanently adds forms to the document as images, ensuring document appearance stability and preventing further modifications.
- **Fast UI Integration:** Achieve rapid integration and customization through extendable UI components.

Guides for Forms

- **Create & Edit Form Fields**

  Create various interactive form fields.

- **Fill Form Fields**

  Add content to form fields.

3.3.2 Supported Form Fields

ComPDFKit for Web now supports the form fields below by the PDF specification.

- Text Fields: It is a box or space for text fill-in data typically entered from a keyboard. The text may be set to a single line or multiple lines.
- Check Boxes: Check boxes could represent one or more checkboxes that can be checked or unchecked.
- Radio Button: Select one option from the predefined options.
- List Box: Select one or more options from the predefined options, similar to the Combo Box.
- Combo Box: Select one option from a drop-down list of available text options.
- Push Button: Create custom buttons on the PDF document that will perform an action when pressed.
- Signatures: Allow users to add electronic signatures to PDF documents.

More supported form fields are coming soon. Please contact us if you have other form field requirements.

3.3.3 Create & Edit Form Fields

Creating form fields works the same as adding any other annotation, as can be seen in the guides for programmatically creating annotations. But you need to change the annotation type to the corresponding form fields, such as `text-field` or `check-box`.

- Text Fields

  Text fields could be created, customized, named, filled, downloaded, hidden, and deleted. Except for the field, ComPDFKit for Web provides options to change the text color, background color, font, single/multiple lines, and alignment of the text in the text field. Here is the sample code below to set edit a text field.

```
docViewer.addAnnotations({
  type: "textfield",
  rect: {
    left: 102,
    top: 136,
    right: 161,
    bottom: 156
  },
  fieldName: "Text Field1",
  isHidden: 0,
  contents: 'test',
  backgroundColor: "#93B9FD",
  color: "#000000",
  fontName: "Helvetica",
  fontSize: 14,
  textAlignment: "left",
  isMultiLine: false,
  pageIndex: 0
})
```

- Check Boxes

Check boxes could be created, customized, named, filled, downloaded, hidden, and deleted. Except for the field, ComPDFKit for Web provides options to set the shape of the marker that appears inside the check box including check, circle, cross, diamond, square, or star. Here is the sample code below to set edit a check box.

```
docViewer.addAnnotations({
  type: "checkbox",
  rect: {
    left: 110,
    top: 190,
    right: 130,
    bottom: 210
  },
  fieldName: "Check Box1",
  isHidden: 0,
  borderColor: "#43474D",
  backgroundColor: "#93B9FD",
  borderWidth: 1,
  borderStyle: "solid",
  checkStyle: 0,
  isChecked: 0,
  pageIndex: 0
})
```

- Radio Button

  Radio buttons could be created, customized, named, filled, downloaded, hidden, and deleted. Except for the field, ComPDFKit for Web provides options to set the shape of the marker that appears inside the radio button including check, circle, cross, diamond, square, or star. Here is the sample code below to edit a radio button.

```
docViewer.addAnnotations({
  type: "radiobutton",
  rect: {
    left: 150,
    top: 190,
    right: 170,
    bottom: 210
  },
  fieldName: "Group1",
  isHidden: 0,
  borderColor: "#43474D",
  borderStyle: "solid",
  borderWidth: 1,
  backgroundColor: "#93B9FD",
  checkStyle: 1,
  isChecked: 0,
  pageIndex: 0
})
```

- List Box

  A list box could be created, customized, named, selected, downloaded, hidden, and deleted. Except for the field, ComPDFKit for Web provides options to change the text color, background color, and font in the list box. Here is the sample code below to edit a list box.

```
docViewer.addAnnotations({
  type: "listbox",
  rect: {
    left: 356,
    top: 176,
    right: 414,
    bottom: 203
  },
  borderColor: "#43474D",
  borderStyle: "solid",
  borderWidth: 1,
  fieldName: "List Box1",
  fontName: "Helvetica",
  fontSize: 14,
  isHidden: 0,
  items: [
    {
      Value: "Item1",
      String: "Item1"
    },
    {
      Value: "Item2",
      String: "Item2"
    }
  ],
  selected: 0,
  color: "#000000",
  backgroundColor: "#93B9FD",
  pageIndex: 0
})
```

- Combo Box

  Combo boxes could be created, customized, named, selected, downloaded, hidden, and deleted. Except for the field, ComPDFKit for Web provides options to change the text color, background color, and font in the combo box. Here is the sample code below to edit a combo box.

```
docViewer.addAnnotations({
  type: "combobox",
  rect: {
    left: 356,
    top: 176,
    right: 414,
    bottom: 203
  },
  borderColor: "#43474D",
  borderStyle: "solid",
  borderWidth: 1,
  fieldName: "Combo Box1",
  fontName: "Helvetica",
  fontSize: 14,
  isHidden: 0,
  items: [
    {
      Value: "Item1",
      String: "Item1"
```

```
    },
    {
      Value: "Item2",
      String: "Item2"
    }
  ],
  selected: 0,
  color: "#000000",
  backgroundColor: "#93B9FD",
  pageIndex: 0
})
```

- Push Button

  Push buttons could be created, customized, named, downloaded, hidden, and deleted. Except for the field, ComPDFKit for Web provides options to change the text color, background color, and font in the push button or set an action to go to the page or open a web link. Here is the sample code below to edit a push button.

```
// Go To Pages.
docViewer.addAnnotations({
  type: "pushbutton",
  rect: {
    left: 356,
    top: 176,
    right: 414,
    bottom: 203
  },
  backgroundColor: "#93B9FD",
  borderColor: "#43474D",
  borderStyle: "solid",
  borderWidth: 1,
  fieldName: "Button1",
  fontName: "Helvetica",
  fontSize: 14,
  actionType: 1,
  isHidden: 0,
  color: "#000000",
  title: "OK",
  pageIndex: 0,
  destPage: 2
})
// Open a Web Link.
docViewer.addAnnotations({
  type: "pushbutton",
  rect: {
    left: 356,
    top: 176,
    right: 414,
    bottom: 203
  },
  backgroundColor: "#93B9FD",
  borderColor: "#43474D",
  borderStyle: "solid",
  borderWidth: 1,
  fieldName: "Button1",
  fontName: "Helvetica",
```

```
  fontSize: 14,
  isHidden: 0,
  color: "#000000",
  title: "OK",
  pageIndex: 0,
  actionType: 6,
  url: "http://example.com"
})
```

3.3.4 Fill Form Fields

ComPDFKit for Web supports the AcroForm standard and provides a simple UI to fill out each form element. Click the corresponding form field and then fill in text or select options using either the onscreen keyboard or an attached hardware keyboard. Then click any blank area on the page to deselect the form element, which will commit the changes.

3.4 Signatures

3.4.1 Overview

It's a legal way to get consent or approval on electronic documents or forms. With Signatures, you can sign any PDF digital document conveniently and reliably without printing. Various types of signatures are supported.

3.4.2 Electronic Signatures

Electronic signatures could be seen as the evidence of the document signatory's agreement, and let you easily and securely sign documents with your signature. You can choose from three different ways to create your signature: draw it with a trackpad or mouse, type it with your preferred font style, or upload a photo of your handwritten signature. You can also customize the color and stroke width of your signature to make it look more authentic.

- Drawn

Drawn signatures let you create your signature with a trackpad or mouse. You can draw your signature as if you were using a pen or a pencil, and adjust the color and stroke width as you like. Drawn signatures are ideal for creating natural and personal signatures that match your handwriting style.

- Image

Image signatures let you upload a photo of your handwritten signature and use it to sign PDF documents. Image signatures are perfect for using your existing signature without having to redraw it every time.

- Typed

Typed signatures let you type your name and choose a font style that suits you. You can select from a variety of fonts that mimic handwriting, and change the color and stroke width as well. Typed signatures are convenient and fast for creating simple and elegant signatures that look professional.

3.5 Security

3.5.1 Overview

The security module provides features including password protection, permission settings, Bates coding, background, and page header and footer functionalities. Document security is guaranteed by managing document passwords and permissions, and by adding logos and copyright information.

Benefits of ComPDFKit Security

- **Access Control:** Restrict sensitive permissions such as access, copying, or printing by configuring security permissions associated with the document.
- **Password Management:** Create, modify, or remove document passwords and permissions.
- **Encryption Standards:** Support for standard PDF security procedures (40 and 128-bit RC4 encryption) as well as 128 and 256-bit AES (Advanced Encryption Standard) encryption.
- **Copyright Identification:** Display document source and copyright information through document background and header/footer, preventing unauthorized screen captures or misuse.
- **Dynamic Identification:** Automatically add Bates coding and header/footer associated with document content through specific expressions.

Guides for Security

- **PDF Permissions**

  By managing document passwords and permission settings, unauthorized access is prevented, and control over user operational permissions for the document is maintained.

3.5.2 PDF Permission

A PDF file can have two different passwords sets, a permissions or owner password and an open or user password.

A PDF user password is used to secure access to PDF documents and requires the correct password to view the content. It is commonly used to protect confidential reports and financial documents. The PDF reader also displays document rights such as copying and printing permissions when a user password is set.

A PDF permission password, also called an owner or master password, protects the permissions of a PDF document. It restricts actions such as making changes or comments and allows control over copying, printing, and modification. Permission passwords ensure integrity and allow management of advanced editing and security settings. They protect the user's copyright and differ from user passwords.

PDF files' password can be set/removed to encrypt/decrypt the PDF document. Encrypt and decrypt the password through the following function.

```
// Encrypt PDF Files.
docViewer.setPassword('test')

// Decrypt PDF Files.
docViewer.removePassword()
```

3.6 Document Comparison

3.6.1 Overview

The document comparison feature is utilized to compare the differences between two documents, highlighting modifications, additions, or deletions. This assists users in identifying changes in complex drawings, text, and image content.

Benefits of ComPDFKit Document Comparison

- **Overlay Comparison:** Highlight document differences using colored lines, suitable for comparing variances in complex drawings.
- **Content Comparison:** Retrieve differential content and locations, suitable for document comparison with substantial text and image differences.

Guides for Document Comparison

- **Overlay Comparison**

  Generate a comparative result document based on two documents for comparison, marking document differences with colored lines.

- **Content Comparison**

  Compare the content of two documents (including text and images) and present the differences in a list format.

3.6.2 Overlay Comparison

Overlay comparison is to superpose two files and show the differences by different colors. The superimposed part shows the mixed color. You can set the color, transparency, and Blend Mode of the two files. Here are the supported Blend Modes: Normal, Multiply, Screen, Overlay, Darken, Lighten, ColorDodge, ColorBurn, HardLight, SoftLight, Difference, Exclusion, Hue, Saturation, Color, Luminosity.

```
// Get the file stream of the compared files.
const documentA = await fetch('https://example.com/documentA.pdf')
const documentABlob = await documentA.blob()
const documentAFile = new File([documentABlob], 'documentA.pdf')
const documentB = await fetch('https://example.com/documentB.pdf')
const documentBBlob = await documentB.blob()
const documentBFile = new File([documentBBlob], 'documentB.pdf')

const data = {
  leftFile: documentAFile, // Old file.
  rightFile: documentBFile, // New file.
  type: 2, // The type of document comparison.
  inTransparency: '50', // The transparency of the old file.
  newTransparency: '50', // The transparency of of the new file.
  coverType: '0', // Blend Modes.
  inColor: '#FBBDBF', // The color of the old file.
  newColor: '#93B9FD' // The color of of the new file.
}

// Get object containing document blob object of the comparison result.
const res = await docViewer.compare(data)
```

3.6.3 Content Comparison

Content comparison is often used to quickly find the changes between different PDF versions, and is suitable for files with much text information. The content that can be compared includes text, images, etc.

```
// Get the file stream of the compared files.
const documentA = await fetch('https://example.com/documentA.pdf')
const documentABlob = await documentA.blob()
const documentAFile = new File([documentABlob], 'documentA.pdf')
const documentB = await fetch('https://example.com/documentB.pdf')
const documentBBlob = await documentB.blob()
const documentBFile = new File([documentBBlob], 'documentB.pdf')

const data = {
  leftFile: documentAFile, // Old file.
  rightFile: documentBFile, // New file.
  type: 1, // The type of document comparison.
  textCompare: true,
  imgCompare: true,
  replaceColor: '#93B9FD',
  insertColor: '#C0FFEC',
  deleteColor: '#FBBDBF'
}

// Get an array containing document blob object of the comparison result.
const res = await docViewer.compare(data)
```

3.7 Content Editor

3.7.1 Overview

Content editing provides the ability to edit text and images, allowing users to easily insert, adjust, and delete text or images. This makes PDF programs function similarly to a rich text editor, enabling direct composition and modification.

Benefits of ComPDFKit Content Editor

- **Text Editing:** Freely insert or delete text, or adjust text properties and positions.
- **Image Editing:** Freely insert or delete images, or adjust image properties and positions.
- **Search and Replace:** Search for a keyword and replace it with other content.
- **Undo and Redo:** Undo or redo any changes.
- **Custom Menus:** Customize content editing context menus.
- **Fast UI Integration:** Achieve quick integration and customization through expandable UI components.

Guides for Content Editor

- Initialize Editing Mode

  Set the content editing mode and choose the target type for editing.

- Create, Move, and Delete Text

Create, move, and delete text and images.

- **Edit Text Properties**

  Edit the existing text and image properties on the document.

- **End Content Editing and Save**

  Exit editing mode and save the document after completing the edits.

3.7.2 Initialize Editing Mode

Before editing, you should initialize editing mode. ComPDFKit provides methods to initialize editing mode. The following code shows you how to initialize the editing mode:

```
docViewer.startPDFContentEditMode()
```

3.7.3 Create, Move, and Delete Text

ComPDFKit provides methods to do various operations like creating text when you are in content editor mode.
You can use the mouse and keyboard to manipulate text areas on `WebViewer` as in Microsoft Word, you can add, copy, paste, cut, or delete text by keyboard.

```
// Insert Text.
docViewer.contentEditAnnotationsManager.addTextEditor({
  pageNumber: 1,
  rect: {
    left: 240,
    top: 32,
    right: 300,
    bottom: 48
  },
  fontData: {
    fontName: 'Helvetica',
    fontSize: 14,
    r: 0,
    g: 0,
    b: 0,
    opacity: 1,
    isBold: 0,
    italic: 0,
  },
  alignType: 2
})
```

3.7.4 Edit Text Properties

ComPDFKit provides multiple methods to modify text properties. You can modify text font size, name, color, alignment, italic, bold, transparency, etc when you are in content editor mode. The following code shows you how to set text to 22px, black, bold, font **Times-Roman**, and opacity 60.

ComPDFKit for Web supports Helvetica, Courier, Times Roman, and DroidSansFallbackFull font family. DroidSansFallbackFull font is a font family that supports Chinese, Japanese, Korean, and other fonts.

```
const editAnnotations = await
docViewer.contentEditAnnotationsManager.getEditAnnotation(1)
editAnnotation[0].setTextStyle({
  color: '#000000',
  opacity: 60,
  fontSize: 22,
  fontFamily: 'Times-Roman',
  fontStyle: 'bold'
})
```

3.7.5 End Content Editing and Save

```
docViewer.endPDFContentEditMode()
```

3.8 Document Editor

3.8.1 Overview

The document editing functionality offers a range of capabilities to manipulate pages, allowing users to control the document structure and adjust the layout and formatting, ensuring that the document content is presented accurately and in a well-organized manner.

Benefits of ComPDFKit Document Editor

- **Insertion or Deletion of Pages:** Insert or delete pages within the document to meet specific layout requirements.
- **Structural Adjustments:** Adjust the sequence or rotate the orientation of pages to meet specific display or printing needs.
- **Multi-Document Collaboration:** Extract pages from one document and insert them into another, facilitating collaboration and content integration.

Guides for Document Editor

- **Insert Pages**

  Insert blank pages or pages from another document into the target document.

- **Delete Pages**

  Remove pages from the document.

- **Rotate Pages**

  Rotate pages within a PDF document.

- **Replace Pages**

  Replace specified pages in the target document with pages from another document.

- **Extract Pages**

  Extract pages from the document.

- **Move Pages**

  Move pages from the document.

- **Copy Pages**

  Copy pages from the document.

3.8.2 Insert Pages

Insert a blank page or pages from other PDFs into the target document.

Insert Blank Pages

This example shows how to insert a blank page:

```
// Insert after the first page.
const pageIndex = 1;
const width = 612;
const height = 792;

docViewer.insertBlankPage(pageIndex, width, height)
```

Insert Pages from other PDFs

This example shows how to insert pages from other PDFs:

```
const file = {...} // File Object.
const pageIndexToInsert = 1
const page1 = 1, page2 = 2, page3To5 = '3-5'
const pagesToInsert = [page1, page2, page3To5] // Insert the range of PDF pages.

docViewer.insertPages(file, pageIndexToInsert, pagesToInsert)
```

3.8.3 Delete Pages

This example shows how to delete pages:

```
// Delete the first page of the document.
const pagesIndexToDelete = [0]
docViewer.removePages(pagesIndexToDelete)
```

3.8.4 Rotate Pages

This example shows how to rotate pages:

```
// Rotate the first page 90 degrees clockwise, with each unit of rotation
representing a 90-degree clockwise turn.
const pagesIndexToRotate = [0]
const rotation = 1;
docViewer.rotatePages(pagesIndexToRotate, rotation)
```

3.8.5 Replace Pages

The steps to replace pages are as follows:

1. Remove the pages in the target file that need to be replaced.
2. Insert the replacement pages into the location where the original document was deleted.

This example shows how to replace pages:

```
// Remove the first page from the document.
docViewer.removePages([0])

// Insert the first page of another document into the original document's first-page position to complete the replacement.
const file = {...} // File Object.
const pageIndexToInsert = 0
const pagesIndexToInsert = 'all' // Insert the range of PDF pages.

docViewer.insertPages(file, pageIndexToInsert, pagesIndexToInsert)
```

3.8.6 Extract Pages

This example shows how to extract pages:

```
// Extract the first, third, and fourth pages of the original document and save it in a new document.
const pagesIndexToExtract = [1, '3-4'] // Extract the range of the page.
const data = await docViewer.extractPages(pagesIndexToExtract)

//You can save the blob to a file or upload to a server
const blob = new Blob([data], { type: 'application/pdf' });
```

3.8.7 Move Pages

This example shows how to move pages:

```
// Move the first page of the document to the second page.
const pagesIndexToMove = [0]
const targetPageIndex = 2
docViewer.movePages(pagesIndexToMove, targetPageIndex)
```

3.8.8 Copy Pages

This example shows how to copy pages:

```
// Copy the first page of the document.
const pagesIndexToCopy = [0]
docViewer.copyPages(pagesIndexToCopy)
```

3.9 Coordinates

3.9.1 Overview

The (0, 0) point is located at the top left of the page in WebViewer. The x axis extends horizontally to the right and the y axis extends vertically downward.

Guides for Coordinates

- **Convert PDF coordinates to Window coordinates**

  PDF coordinates can be converted to Window coordinates.

- **Convert Window coordinates to PDF coordinates**

  Window coordinates can be converted to PDF coordinates.

- **Convert coordinates of clicked point to PDF coordinates**

  Get the coordinates of clicked point in PDF.

3.9.2 Convert PDF coordinates to Window coordinates

```
const pageNumber = 1;
const pagePoint = {
    x: 0,
    y: 0
};
const windowPoint = docViewer.pageToWindow(pagePoint, pageNumber);
// { x: 47, y: 64 }
```

3.9.3 Convert Window coordinates to PDF coordinates

```
const pageNumber = 1;
const windowPoint = {
    x: 47,
    y: 64
};
const pagePoint = docViewer.windowToPage(windowPoint, pageNumber);
// { pageNumber: 1, x: 0, y: 0 }
```

3.9.4 Convert coordinates of clicked point to PDF coordinates

```
const getMouseLocation = e => {
  const scrollElement = docViewer.getScrollViewElement();
  const scrollLeft = scrollElement.scrollLeft || 0;
  const scrollTop = scrollElement.scrollTop || 0;

  return {
    x: e.pageX + scrollLeft,
```

```
    y: e.pageY + scrollTop
  };
}

const handleClick = (e) => {
  let windowCoordinates = getMouseLocation(e);
  const page = docViewer.getSelectedPage(windowCoordinates, windowCoordinates);

  const clickPageNumber = (page.first !== null) ? page.first :
docViewer.getCurrentPage();
  const pagePoint = docViewer.windowToPage(windowCoordinates, clickPageNumber);
}

docViewer.addEvent('click', handleClick)
```

4 Support

4.1 Reporting Problems

Thank you for your interest in ComPDFKit for Web, the only easy-to-use but powerful development solution to integrate high quality PDF rendering and editing capabilities to your web app. If you encounter any technical questions or bug issues when using ComPDFKit for Web, please submit the problem report to the ComPDFKit team. More information as follows would help us to solve your problem:

- ComPDFKit for Web version.
- The Framework you used in your project.
- Detailed descriptions of the problem.
- Any other related information, such as an error screenshot.

4.2 Contact Information

Website:

- Home Page: https://www.compdf.com
- API Page: https://api.compdf.com/

Contact ComPDFKit:

- Contact Sales: https://api.compdf.com/contact-us
- Technical Issues Feedback: https://www.compdf.com/support
- Contact Email: support@compdf.com

Thanks,

The ComPDFKit Team