



DEVELOPER GUIDE

Foxit PDF SDK for Web

Microsoft® Partner
Gold Independent Software Vendor (ISV)

©Foxit Software Incorporated. All rights reserved.

TABLE OF CONTENTS

Foxit PDF SDK for Web Overview	1
Why Foxit PDF SDK for Web is your choice	1
Audience and Scope	2
Your Web Application.....	2
Evaluation	2
License.....	2
Getting Started	3
Understanding the Package Structure.....	3
Package Introduction.....	3
Package.json	4
The third-party libraries used in Foxit PDF SDK for Web.....	5
Quickly Run Examples.....	7
Integration	8
Preparations	8
Integrate the basic webViewer into your project	9
Integrate the complete webViewer into your project.....	11
Integration Modes.....	13
Example.....	14
Example Projects	14
UIExtension	14
PDFViewCtrl.....	16
HTTP Server Configuration Examples.....	18
Scaffold Demo	21
How to run this demo	21
Source code	21

Features	23
Digital Signature.....	23
Steps to sign and verify digital signature on PDF	23
Related digital signature APIs	24
Interact with the digital signature feature.....	25
About signature HTTP service.....	28
Import and Export	29
Annotation	29
Form.....	30
Stamp and customization	30
Default stamp list	30
Manage Stamp list.....	32
About the stamp category and name	33
Add a stamp onto page in Viewer	34
Add a custom stamp onto page by API.....	34
Set the default tool to a particular stamp in Viewer	35
Related APIs	36
Custom Speech Synthesizer	36
Speech Synthesizer APIs.....	37
Customize PDFTextToSpeechSynthesis.....	38
Integrate with 3rd Party TTS Service	41
Collaboration	45
Getting started with collaboration	45
Quickly run collaboration example	45
Working principle	45
The infinite loop problems.....	49
Best Practice	52

Website assets optimization	52
Gizp and Brotli compression.....	52
Cache	52
Foxit PDF SDK for Web configuration	53
Read only.....	53
Brotli compression	54
Preload webassembly artifacts.....	55
Tiling size	56
Tiling size and zoom.....	56
Rendering mode.....	57
Document loading.....	58
I18n Entries Resources Management.....	61
Explanation.....	61
Overview	61
SDK I18n Entries Resource Management.....	61
The directory structure and the role of the file	61
The directory of the custom entry file	62
Verify the configuration in developer environment.....	62
Add new languages	63
Rewrite some of the entries	64
Customize the entries of Addon	64
Troubleshooting	66
Thumbnail Loading Error.....	66
This component is unavailable until "thumbnail" addon is loaded	66
Solutions.....	66
Basics	68
Appearance	68

Custom Appearance Example	68
Device Adaptation	70
Built-in appearances	71
Modular	71
Create a new module.....	72
Get module object.....	72
The methods of the module object.....	72
The layout template	74
Example.....	74
Description of the format of layout template format.....	76
How to specify layout templates and implement device adaptation	76
Dynamically insert layout templates.....	76
Insert the layout template when initializing.....	78
UI fragments.....	78
simple example	78
The description of the Fragment configuration parameters	80
Note	82
Component selector	83
Syntax	83
Examples	84
Icon	86
l18n.....	86
Custom resources	86
Usage	86
Components.....	91
Basic Components.....	91
XButton component.....	91
File selector	99

Dropdown component	101
Tab component	115
Sidebar Component	128
Paddle component	138
Group component	144
Layer component	150
Number component	160
Contextmenu component	162
viewer component	165
Business Components	169
Pre-configured component	169
Directives	189
@controller	189
Usage	189
Example	189
@tooltip	190
Example	190
@draggable	196
Example	196
@device	200
Device Type	200
Example	201
@require-modules	203
Code example	203
Addons	208
Introduction to addons	208
Load Addons	209
The addon's structure	209

allInOne.js	210
Merge addons.....	211
Develop custom addons.....	211
Customization	212
Customize the UI.....	212
Customize the UI layout using template	212
Customize the UI using fragments.....	216
Modularization	219
Understanding the built-in components	221
Customize annotation right-click menu.....	228
Customize right-click menu for supported annotations	229
Customize right-click menu for unsupported annotations.....	231
Hiding the right-click menu or items.....	232
Showing a customized right-click menu	233
Customize page right-click menu	235
Page right-click menu items.....	235
Removing a menu item	235
Replacing a menu item.....	235
Inserting a new item	236
Hiding the right-click menu or items.....	236
Showing a customized right-click menu	237
Customize the Floating Text Selection Tooltip.....	239
A sample for creating a custom controller and modifying components by fragmentation	239
The logic processing methods used in floating tooltip.....	239
The component name of the floating tooltip.....	239
Customize Internationalization Resources	240
Assumption	240
Configuration	240

Verify the configuration in developer environment.....	241
Make the Web Viewer Adaptive to the Device	241
Framework Integration	242
Foxit PDF SDK for Web Example - Angular.js	242
Quickly run the out-of-the-box example for Angular	242
Integrate Foxit PDF SDK for Web into existing Angular project.....	242
Foxit PDF SDK for Web Example - React.js	247
Quickly run the out-of-the-box sample for react.js.....	247
Integrate Foxit PDF SDK for Web into existing Angular project.....	251
Foxit PDF SDK for Web Example - React.js created by "create-react-app"	252
Quickly run the out-of-the-box sample for create-react-app	252
Integrate Web SDK to react app created by "create-react-app"	254
Foxit PDF SDK for Web Example - Vue.js	256
Quickly run the out-of-the-box example for Vue.js.....	256
Integrate Foxit PDF SDK for Web into existing Vue.js project	257

Foxit PDF SDK for Web Overview

Foxit PDF SDK for Web is a lightweight powerful PDF library for web applications developed taking all advantages of Foxit's signature core rendering engine. Using the SDK, developers can deploy and customize a complete PDF viewer to display, annotate, fill forms and sign documents in a web browser. Integrating Foxit PDF SDK for Web into a zero-footprint web application allows end users to view PDF documents on any type of device without installing anything.

Why Foxit PDF SDK for Web is your choice

Foxit is a leading software provider of solutions for reading, editing, creating, organizing, and securing PDF documents. Foxit PDF SDK for Web is a cross-platform solution for PDF online viewing. Foxit PDF SDK for Web enterprise edition has been chosen by many of the world's leading firms for integration into their solutions. Customers choose this product for the following reasons:

Fully customizable

Developers can easily design a unique style for their Foxit PDF SDK for Web user interface, making it consistent to their own branding and other web applications.

Easy to integrate

Developers can easily integrate Foxit PDF SDK for Web by referring to the product's knowledge base and writing a small amount of code to display and edit PDF files. The web-based pdf viewer also provides a large amount of interfaces to connect users and user data.

Standard and consistent annotation data

The annotations in Foxit PDF SDK for Web are consistent when viewing and editing in other applications, as well as following industry-leading professional standards for quality and compliance.

Powered by Foxit's high fidelity rendering PDF engine

The core technology of Foxit PDF SDK for Web is based on Foxit's PDF engine, which is trusted by a large number of well-known companies. Foxit's powerful engine makes document viewing fast and consistent in all environments.

In addition, Foxit's products are offered with the full support of our dedicated support engineers if support and maintenance options are purchased. Updates are released on a regular basis. Foxit PDF

SDK for Web is the most cost-effective choice if you want to develop a cross-platform web PDF document viewer.

Audience and Scope

This document is primarily intended for developers who need to integrate the Foxit PDF SDK for Web into their web applications. It includes the direct reference examples as well as custom front-end APIs for customization.

Your Web Application

Foxit PDF SDK for Web provides a solution that enables a web viewer to interact with PDFs seamlessly without any plugins or local applications. Developers should prepare a PDF hosting server like Nginx, Apache or the HTTP server in Node.js platform and do the usual configuration before using Foxit PDF SDK for Web.

Evaluation

Foxit PDF SDK for Web allows users to download the trial version to evaluate the SDK. The trial version is the same as the standard version except for the 15-day limitation for free trial and the trial watermarks in the generated pages. After the evaluation period expires, customers should contact the Foxit sales team and purchase licenses to continue using Foxit PDF SDK for Web.

License

Developers are required to purchase licenses to use Foxit PDF SDK for Web in their solutions. Licenses grant users permission to release their applications based on Foxit PDF SDK for Web. However, users are prohibited to distribute any documents, sample codes, or source codes in the released packages of Foxit PDF SDK for Web to any third party without the permission from Foxit Software Incorporated.

Getting Started

Understanding the Package Structure

Package Introduction

Foxit PDF SDK for Web provides two packages as follows:

- Light package: FoxitPDFSDKForWeb_8_0.zip (excludes font resources)
- Full package: FoxitPDFSDKForWeb_8_0_Full.zip (includes font resources)

If you already have the font resources or only want to use online fonts, you can choose the light package. If you don't want to make any change on the font library and don't care the package size, the full package is the most suitable choice.

The package contains the following folder structure:

Folder/File	Description
docs:	Contains API reference documents and Foxit PDF SDK for Web's developer guide.
examples:	A series of demos and examples of how to take advantage of all Foxit PDF SDK for Web features.
external	Font resources (only for full package).
integrations	Integration samples for wrapping Foxit PDF SDK for Web into current popular JavaScript frameworks (AngularJS/React.js/Vue.js).
lib	Foxit PDF SDK for Web core libraries.
server	http-server and the Node.js scripts for a series of server-based utility applications to use in the viewer.
legal.txt	Legal and copyright information.
package.json	Project description file.

The "lib" folder's file structure is provided as follows:

jr-engine	Front-end rendering engine.
-----------	-----------------------------

locales	Internationalized entries data for using the viewer in different languages. Every language is placed in a different directory with its own label.
PDFViewCtrl	Plugins for the PDFViewCtrl library.
stamps	Stamps resources, image files and templates.
uix-addons	All plugins for the UIExtensions project.
adaptive.js	A responsive design script to adapt the viewer to mobile devices
PDFViewCtrl.css	CSS file for the PDFViewCtrl viewer UI style.
PDFViewCtrl.full.js	Complete script file for the PDFViewCtrl viewer library.
PDFViewCtrl.js	Script file for the PDFViewCtrl viewer library without third-party libraries.
PDFViewCtrl.polyfills.js	Browser-adapted polyfill script file for the PDFViewCtrl viewer library.
PDFViewCtrl.vendor.js	Third-party libraries script used by PDFViewCtrl (See the lists later).
preload-jr-worker.js	Worker script for loading resources of JS engine in parallel to the UI for improving the viewer loading speed.
UIExtension.css	The default CSS file of the UI.
UIExtension.vw.css	The CSS file using vmin unit.
UIExtension.full.js	Complete script file for the UIExtension full-featured viewer library.
UIExtension.js	Script file for the UIExtension viewer library without third-party libraries
UIExtension.polyfills.js	Browser-adapted polyfill script file for the UIExtensions viewer library.
UIExtension.vendor.js	Third-party libraries script used by UIExtension (See the lists later).
WebPDFJRWorker.js	Script files running in the Web Worker, which are used for calling the front-end rendering engine.
WebPDFSRWorker.js	Script files running in the Web Worker, which are used for calling the server rendering engine.

Package.json

Foxit PDF SDK for Web provides a package.json file to help developers quickly deploy and use the SDK, and make it easy to integrate into their project. The content is as follows:

```
{
  "name": "foxit-pdf-sdk-for-web",
  "version": "8.0",
  "description": "Foxit pdf sdk for web.",
  "author": "Foxit Software Inc.",
  "main": "./lib/PDFViewCtrl.full.js",
  "scripts": {
    "start": "concurrently --kill-others \"npm run start-http-server\" \"npm run start-snapshot-server\"",
    "start-snapshot-server": "node ./server/snapshot/src/index -p 3002",
    "start-http-server": "node ./server/index"
  },
  "devDependencies": {
    "boxen": "^4.1.0",
    "chalk": "^2.4.1",
    "concurrently": "^4.1.0",
    "http-proxy-middleware": "^0.19.1",
    "koa": "^2.7.0",
    "koa-body": "^4.0.4",
    "koa-body-parser": "^1.1.2",
    "koa-router": "^7.4.0",
    "koa2-connect": "^1.0.2",
    "lru-cache": "^4.1.3",
    "raw-body": "^2.3.3",
    "require-dir": "^1.0.0",
    "serve-handler": "^6.0.2"
  },
  "serve": {
    "port": 8080,
    "public": "/",
    "proxy": {
      "target": "http://127.0.0.1:3002",
      "changeOrigin": true
    }
  }
}
```

The third-party libraries used in Foxit PDF SDK for Web

Foxit PDF SDK for Web provides its script files in two versions: the full version script that includes the third-party libraries, and the regular script without any third-party libraries. If your project already uses the dependencies included in the SDK's third-party libraries, you don't need to re-install them.

The PDFViewCtrl.full.js script contains:

PDFViewCtrl.full.js	Complete script file for the PDFViewCtrl viewer library.
PDFViewCtrl.polyfills.js	Browser-adapted polyfill script file for the PDFViewCtrl viewer library.

PDFViewCtrl.vendor.js	Third-party libraries script used by PDFViewCtrl (See the list of vendors below this section).
PDFViewCtrl.js	Script file for the PDFViewCtrl viewer library without third-party libraries.

So, **PDFViewCtrl.polyfills.js** + **PDFViewCtrl.vendor.js** + **PDFViewCtrl.js** = **PDFViewCtrl.full.js**.

Essentially, the two scripts below are the same thing:

```
1) <script src="../../FoxitPDFSDKForWeb/lib/PDFViewCtrl.full.js"></script>
2) <script src="../../FoxitPDFSDKForWeb/lib/PDFViewCtrl.polyfills.js"></script>
   <script src="../../FoxitPDFSDKForWeb/lib/PDFViewCtrl.vendor.js"></script>
   <script src="../../FoxitPDFSDKForWeb/lib/PDFViewCtrl.js"></script>
```

The third-party libraries contained in **PDFViewCtrl.vendor.js** are outlined below:

```
jquery
i18next
i18next-chained-backend
i18next-localstorage-backend
i18next-xhr-backend
jquery-contextmenu
dialog-polyfill
hammerjs
eventemitter3
```

The **UIExtension.full.js** script contains:

UIExtension.full.js	Complete script file for the UIExtension viewer library.
UIExtension.polyfills.js	Browser-adapted polyfill script file for the UIExtension viewer library.
UIExtension.vendor.js	Third-party libraries script used by UIExtension (See the list of vendors below this section).
UIExtension.js	Script file for the UIExtension viewer library without third-party libraries.

So, **UIExtension.polyfills.js** + **UIExtension.vendor.js** + **UIExtension.js** = **UIExtension.full.js**.

Essentially, the two scripts below are the same thing:

```
1) <script src="../../FoxitPDFSDKForWeb/lib/UIExtension.full.js"></script>
```

```
2) <script src="../../FoxitPDFSDKForWeb/lib/UIExtension.polyfills.js"></script>
    <script src="../../FoxitPDFSDKForWeb/lib/UIExtension.vendor.js"></script>
    <script src="../../FoxitPDFSDKForWeb/lib/UIExtension.js"></script>
```

The third-party libraries contained in **UIExtension.vendor.js** are outlined below:

```
jquery
i18next
i18next-chained-backend
i18next-localstorage-backend
i18next-xhr-backend
dialog-polyfill
hammerjs
eventemitter3
spectrum-colorpicker
file-saver
```

Quickly Run Examples

Foxit PDF SDK for Web comes with a lot of example projects and files for building the viewer and/or implementing additional functionality. These examples are provided in the examples folder of Foxit PDF SDK for Web. To run them, initialize your (local) web server, open your browser and add the localhost (https://localhost:port) or corresponding IP number URL. The directory list of files will be displayed and you can choose which sample to use.

To quickly get a web server running on your local system, you can use node.js [http-server](#):

```
http-server
```

Additionally, you can append the '-o' command to open directly in your browser window:

```
http-server -o
```

You can also use Python's [SimpleHTTPServer](#) module:

```
python -m http.server 8000
```

You may want to refer to [Set up local server](#) for more information.

See also

- [Start Http Server using Nginx](#)

- [Start Http Server using Nodejs](#)

Integration

This section will help you to quickly get started with using Foxit PDF SDK for Web to build a simple web PDF viewer and a full-featured PDF viewer with step-by-step instructions provided.

Preparations

Create a new web project

- 1) Create a new directory as a project folder, such as "D:/test_web".
- 2) Copy the "**lib**", "**server**", and "**external**" (if you need to use the font resources) folders, as well as the "**package.json**" file from Foxit PDF SDK for Web package to "D:/test_web".
- 3) Copy a PDF file (for example, the demo guide in the "docs" folder) to "D:/test_web".
- 4) Create a html file (index.html) in the "D:/test_web" folder. Then the directory structure is:

```
test_web
+-- lib           (copy from the Foxit PDF SDK for Web package)
+-- server        (copy from the Foxit PDF SDK for Web package)
+-- package.json  (copy from the Foxit PDF SDK for Web package)
+-- index.html
```

The whole content of the index.html is:

```
<html>
<head>
  <meta charset="utf-8">
  <style>
    .fv__ui-tab-nav li span {
      color: #636363;
    }
    .flex-row {
      display: flex;
      flex-direction: row;
    }
  </style>
  <!-- ignore other unimportant code -->
</head>
<body>
</body>
</html>
```


Integrate the basic webViewer into your project

This section will describe how to integrate the basic webViewer sample using PDFViewCtrl based on the above created project. Just follow the steps below:

- 1) Add styles (/lib/PDFViewCtrl.css) to the <head> tag of the HTML page:

```
<link rel="stylesheet" type="text/css" href="/lib/PDFViewCtrl.css">
```

- 2) Import the "PDFViewCtrl.full.js" library found in the "lib" folder:

```
<script src="/lib/PDFViewCtrl.full.js"></script>
```

- 3) In the HTML <body> tag, add the <div> elements as the web viewer container:

```
<div id="pdf-viewer"></div>
```

- 4) Initialize PDFViewCtrl:

```
<script>
  var licenseSN = "Your license SN";
  var licenseKey = "Your license Key";
</script>
<script>
  var PDFViewer = PDFViewCtrl.PDFViewer;
  var pdfViewer = new PDFViewer({
    libPath: './lib', // the library path of Web SDK.
    jr: {
      licenseSN: licenseSN,
      licenseKey: licenseKey,
    }
  });
  pdfViewer.init('#pdf-viewer'); // the div (id="pdf-viewer")
</script>
```

Note: The trial values of **licenseSN** and **licenseKey** can be found in the `examples/license-key.js` file of Foxit PDF SDK for Web package.

- 5) Open a PDF document:

```
// modify the file path as your need.
fetch('/FoxitPDFSDKforWeb_DemoGuide.pdf').then(function(response) {
  response.arrayBuffer().then(function(buffer) {
    pdfViewer.openPDFByFile(buffer);
  })
})
```

The above steps are the key points of integrating the simple demo to your created project using PDFViewCtrl. After finishing it, refresh your browser (`<index.html>`).

Now, in this simple web PDF viewer, you can zoom in/out the PDF document by right-clicking anywhere on the page to select the zoom in or zoom out options.

The whole content of the **index.html** is:

```
<html>
<head>
  <meta charset="utf-8">
  <link rel="stylesheet" type="text/css" href="/lib/PDFViewCtrl.css">

  <!-- You can delete the following style because it doesn't work in this project -->
<style>
  .fv__ui-tab-nav li span {
    color: #636363;
  }
  .flex-row {
    display: flex;
    flex-direction: row;
  }
</style>
  <!-- ignore other unimportant code -->
</head>
<body>
  <div id="pdf-viewer"></div>
  <script src="/lib/PDFViewCtrl.full.js"></script>
  <script>
    var licenseSN = "Your license SN";
    var licenseKey = "Your license Key";
  </script>
  <script>
    var PDFViewer = PDFViewCtrl.PDFViewer;
    var pdfViewer = new PDFViewer({
      libPath: './lib', // the library path of Web SDK.
      jr: {
        licenseSN: licenseSN,
        licenseKey: licenseKey,
      }
    });
    pdfViewer.init('#pdf-viewer'); // the div (id="pdf-viewer")

    // modify the file path as your need.
    fetch('/FoxitPDFSDKforWeb_DemoGuide.pdf').then(function (response) {
      response.arrayBuffer().then(function (buffer) {
        pdfViewer.openPDFByFile(buffer);
      })
    })

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

Integrate the complete webViewer into your project

The previous section introduces how to integrate the basic webViewer sample using PDFViewCtrl, which is just a simple web PDF viewer. In this section, we will show you how to integrate the advanced webViewer using UIExtension based on the [Preparations](#). Follow the steps below:

- 1) Add styles (/lib/UIExtension.css) to the <head> tag of the HTML page.

```
<link rel="stylesheet" type="text/css" href="/lib/UIExtension.css">
```

- 2) Import the "UIExtension.full.js" library found in the "lib" folder:

```
<script src="/lib/UIExtension.full.js"></script>
```

- 3) In the HTML <body> tag, add the <div> elements as the webViewer container:

```
<div id="pdf-ui"></div>
```

- 4) Initialize UIExtension:

```
<script>
  var licenseSN = "Your license SN";
  var licenseKey = "Your license Key";
</script>
<script>
  var pdfui = new UIExtension.PDFUI({
    viewerOptions: {
      libPath: './lib', // the library path of web sdk.
      jr: {
        licenseSN: licenseSN,
        licenseKey: licenseKey
      }
    },
    renderTo: '#pdf-ui' // the div (id="pdf-ui").
  });
</script>
```

Note: The trial values of **licenseSN** and **licenseKey** can be found in the `examples/license-key.js` file of Foxit PDF SDK for Web package.

- 5) Open a PDF document:

```
// modify the file path as your need.
fetch('/FoxitPDFSDKforWeb_DemoGuide.pdf').then(function(response) {
  response.arrayBuffer().then(function(buffer) {
    pdfui.openPDFByFile(buffer);
  })
})
```

The above steps are the key points of integrating the advanced webViewer to your created project using UIExtension. After finishing it, refresh your browser ([<index.html>](#)).

Now, it is a full-featured web PDF viewer, you can view/edit/comment/protect the PDF document as desired.

The whole content of the [index.html](#) is:

```
<html>
<head>
  <meta charset="utf-8">
  <link rel="stylesheet" type="text/css" href="./lib/UIExtension.css">
  <style>
    .fv__ui-tab-nav li span {
      color: #636363;
    }
    .flex-row {
      display: flex;
      flex-direction: row;
    }
  </style>
  <!-- ignore other unimportant code -->
</head>
<body>
  <div id="pdf-ui"></div>
  <script src="./lib/UIExtension.full.js"></script>
  <script>
    var licenseSN = "Your license SN";
    var licenseKey = "Your license Key";
  </script>
  <script>
    var pdfui = new UIExtension.PDFUI({
      viewerOptions: {
        libPath: './lib', // the library path of web sdk.
        jr: {
          licenseSN: licenseSN,
          licenseKey: licenseKey
        }
      },
      renderTo: '#pdf-ui' // the div (id="pdf-ui").
    });

    // modify the file path as your need.
    fetch('/FoxitPDFSDKforWeb_DemoGuide.pdf').then(function (response) {
      response.arrayBuffer().then(function (buffer) {
        pdfui.openPDFByFile(buffer);
      })
    })

  </script>
```

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

Integration Modes

Integrate as a Global Variable

You can integrate the Foxit PDF SDK for Web to your project as a global variable:

```
<script src="./lib/PDFViewCtrl.full.js"></script>  
var PDFViewer = PDFViewCtrl.PDFViewer;  
var pdfViewer = new PDFViewer(...)
```

For a working example, check out the complete_WebViewer demo in the "examples/UIExtension" folder."

Integrate as module

For more integration modes, you may want to check our working examples in the "examples/UIExtension/integrate-as-module/" directory.

Example

Example Projects

UIExtension

Complete webViewer

It is a ready-to-go application that you can run directly or integrate into your project with full features provided by Foxit PDF SDK for Web. This application uses the full-featured package "UIExtension.full.js" for the PDF view and document parsing.

Source folder: /examples/UIExtension/complete_webViewer.

Integration Examples

These examples walk you through integrating Foxit PDF SDK for Web as an es-module, amd or commonJS module. For a global variable integration sample, refer to the code on /examples/UIExtension/complete_webViewer/index.html

Source folder: /examples/UIExtension/integrate-as-module.

Customize Global Annotation Properties

Provide an example to show how to set default annotation properties by using either of the constructor option or the function.

Source folder: /examples/UIExtension/default_annot_config.

Customize Tooltips Example

Provide an example to show how to customize tooltips on sidebar and toolbar.

Source folder: /examples/UIExtension/tooltip.

Asynchronous/Synchronous annotation loading

This demo shows how the annotations in a PDF can be automatically loading in both options, async and synchronous by using the 'lazy' property on the '<commentlist-sidebar-panel>' tag to true or false.

Source folder: /examples/UIExtension/commentlist-loadmode.

Customize Text Selection

This demo provides an example to show how to create a custom controller for text selection.

Source folder: /examples/UIExtension/custom-text-selection-tool.

Customize Annotations Pop-up

The default behavior of double clicking an annotation in webViewer is to trigger the comment panel. This demo guides you how to change the default event by adding the pop-up layer and overwriting the onDoubleTap event.

Source folder: /examples/UIExtension/custom_annotations_popup.

Customize User Interface

Provide code examples to show how to customize user interface. One introduces a non-adaptive sample for PC browser, the other guides how to set up adaptivity for across browsers. webViewer detects the 'navigator.userAgent' in browser when initializing and determines the UI layout accordingly – PC or mobile.

Source folder: /examples/UIExtension/custom_appearance.

Customize Layout by Templates

Provide examples to show the built-in templates in UIExtension and the reference methods. This example is suitable for users who need to fine-tune the template.

Source folder: /examples/UIExtension/layout_templates.

Customize Components by Fragments

Provide examples to show how to modify components and set up components configuration by using fragments.

Source folder: /examples/UIExtension/fragment_usage.

Annotation Data Migration Example

Provide an example to show how to migrate annotations JSON data from v6 to v7 to avoid data lost.

Source folder: /examples/UIExtension/migrateAnnotData.

PWA Example

Provide an example to show how to implement a progressive web app

Source folder: /examples/UIExtension/pwa.

UI Widgets Examples

These are examples referenced by UIExtension.components.widget in API Reference. Each sample shows the usage of a component (including how to pass parameters, event binding, and so on).

Source folder: /examples/UIExtension/tutorials/widgets.

Addon Usage Examples

With this example, you will learn how to merge addons and reference the merged-add.js in your code.

Source folder: /examples/UIExtension/use-merged-addon.

Webpack Scaffold Project

This project provides an open-source code of UI addon for customization.

[Go to Project Page](#)

PDFViewCtrl

Basic webViewer

It is a basic webViewer that demonstrates how to call Foxit PDF SDK for Web API to load a PDF document, and zoom in/out the document. This demo uses the "PDFViewCtrl.full.js" package in the "lib" folder.

Source folder: /examples/PDFViewCtrl/basic_webViewer.

Overwrite PDFPageRendering Example

Provide an example to show how to add a custom UI to the nodes of each PDF page by overwriting the PDFPageRendering class, such as a UI of adding a loading dynamic figure or a similar progress bar.

Source folder: /examples/PDFViewCtrl/override-rendering.

Preload Worker Example

Provide an example to show how to load the worker scripts of the JR engine in advance, to get performance benefit of reducing initializing time.

Source folder: /examples/PDFViewCtrl/preload-worker.

Asynchronous Loading Example

Provide an example to show how to async opening files from URL.

Source folder: /examples/PDFViewCtrl/url.

Offline Example

This example demonstrates how to register the "service-worker.js" found in the "examples/PDFViewCtrl/service-worker" folder to better cache the core dependency files "gsdk.js" and font files in a browser supported by the service worker, in order to speed up the reloading time or use the offline mode.

Source folder: /examples/PDFViewCtrl/service-worker.

Inline DIV Example

This example renders the simple UI of Foxit PDF SDK for Web to a div container with a specified size.

Source folder: /examples/PDFViewCtrl/div.

FileOpen Plugin Example

Provide an example for opening a fileOpen protection file.

Source folder: /examples/PDFViewCtrl/fileopen.

Page Layout Rewriting Example

This example shows how to create a single view page layout and navigate page by up and down arrow keys without scrolling feature. By this example, you will learn how to register and inherit IViewMode to implement your own layout and customize navigating page postures.

Source folder: /examples/PDFViewCtrl/view-mode.

Document Password Re-encryption Example

Provide an example to show how to open a document with password re-encryption. The password re-encryption node.js example can be found at ... \server\encrypt-password.

Source folder: /examples/PDFViewCtrl/encrypt-password.

Page Manipulation Example

Provide an example to show how to manipulate pages.

Source folder: /examples/PDFViewCtrl/ppo.

Form Widgets Adding Example

Provide an example for creating supported form widgets.

Source folder: /examples/PDFViewCtrl/add-form-fields.

Annotation Creating Example

Provide examples to show how to inherit a StateHandler class of link, screen and textMarkup annotation to implement the annotation creating class.

Source folder: /examples/PDFViewCtrl/create-annot.

License Validation Tool

Provide a tool for verifying license validation.

Source folder: /examples/PDFViewCtrl/check-license.

HTTP Server Configuration Examples

Start Http Server using Nginx

Using Windows as an example, assume that [Nginx](#) was installed on your system already. When you have Nginx server running, you can directly modify the 'nginx.conf' in the conf directory, here we directly modify the configuration file to make webViewer run. Please follows the steps below:

- 1) Download Foxit PDF SDK for Web package, unzip it to a folder.
- 2) Locate to the Nginx/conf folder, open the nginx.conf file, add the following listening information:

```
server {
    listen 8080;
    server_name 127.0.0.1;

    location / {
        alias "gotopath/FoxitPDFSDKForWeb/";
        charset utf8;
        index index.html;
    }
}
```

- 3) Restart Nginx server, now you can access the webViewer at

Complete webViewer address:

`http://localhost:8080/examples/UIExtension/complete_webViewer/`

Basic webViewer address:

`http://localhost:8080/examples/UIExtension/basic_webViewer/`

Note: You can run the webViewer according to the above configuration, but at that time the snapshot feature cannot work correctly. The snapshot cannot be cached to clipboard, so that you cannot paste it to the location as desired. Please follow the steps below to build the snapshot server:

- 1) Install node.js 9.0 or higher, if it is already installed, skip it.
- 2) In a command prompt, navigate to the root directory of Foxit PDF SDK for Web.
- 3) Type `npm install` to install the required dependencies.
- 4) Type `npm run start-snapshot-server` to start the snapshot server (the default port is 3002).

Note: If you want to specify the port for snapshot server as desired, you can change it in the "**server/snapshot/package.json**" file of Foxit PDF SDK for Web package. To find the default port 3002, and change it as you wish:

- 5) Configure Nginx reverse proxy in `nginx.conf`:

```
server {
    listen 8080;
    server_name 127.0.0.1;

    location / {
        alias "gotopath/FoxitPDFSDKForWeb/";
        charset utf8;
        index index.html;
    }

    location ~ ^/snapshot/(.+)$ {
        proxy_pass http://127.0.0.1:3002/snapshot/$1$is_args$args;
        proxy_redirect off;

        proxy_request_buffering on;

        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

- 6) Restart Nginx server, and refresh your browser, the snapshot feature should work correctly.

Start Http Server using Nodejs

Assume that [Node.js](#) 9.0 or higher is available on your system already. Then follow the steps below to run the webViewer:

- 1) Download Foxit PDF SDK for Web package, unzip it, navigate to the root folder and execute to install dependencies:

```
npm install
```

- 2) Run the web server with the command below:

```
npm start
```

- 3) The webViewer can be assessed with the following address:

Complete webViewer address:

http://localhost:8080/examples/UIExtension/complete_webViewer/

Basic webViewer address:

http://localhost:8080/examples/UIExtension/basic_webViewer/

Note: Using this method, you do not need to configure the proxy, the snapshot feature can be used normally. If you want to specify the ports for http-server and snapshot server as desired, you can change the two ports in the **package.json** file of Foxit PDF SDK for Web package.

To change the port for http-server and snapshot server, find the default port 8080 and 3002 as below, and change it as you wish:

```
"serve": {  
  "port": 8080,  
  "public": "/",  
  "proxy": {  
    "target": "http://127.0.0.1:3002",  
    "changeOrigin": true  
  }  
}
```

Scaffold Demo

This is a scaffold demo for UIExtension, including an open-source UI addon. It shows how to customize UI and how to use declaration file. The demo can be accessed at `./examples/UIExtension/scaffoldDemo`.

How to run this demo

Setup library

Execute command `npm run setup` in the demo root folder.

This setup would:

- Add the `lib` directory to the dependency list as a local npm repository..
- install all npm package that needed.

Run Demo

Execute command: `npm start`

Source code

The structure of `src` folder:

```
| addons.js          --- shows how to use foxit addons.
| index.js          --- entry.
|
└─addonExample      --- an addon example.
    | addon.info.json --- addon entry file, which specified all related file in an addon.
    | index.css       --- style sheet
    | index.js        --- addon script entry. DON'T modify it's file name.
    |
    └─locales         --- i18n files
        | en-US.json
        | zh-CN.json
        |
        └─stateHandlers --- State Handler Classes which extend IStateHandler
            | addTextField.js
            |
            └─templates
                | custom-dialog.art --- Art template for customized dialog.
                | tab-template.art  --- Art template for customized tool bar.
```

Entry file

The `src/addonExample/index.js` file is the script entry of addon. View [source file](#) for more details.

Features

Digital Signature

In this section, you will learn about the general steps to sign and verify signature, related signature APIs, ways to interact with the digital signature, and the test signature service routes we provided.

Steps to sign and verify digital signature on PDF

To sign and verify a digital signature on PDF, you should go over the following procedures:

- Sign Document
 1. Generate a file stream which contains signature's byteRange. You may refer to PDF Reference 1.7+ for details.
 2. Calculate the message digest of the content covered by signature's byteRange. This can be implemented by calling `PDFUI.registerSignHandler(signerInfo)` or `PDFDoc.sign(signInfo,digestSignHandler)`.
 3. Get signedData by signing the digest using certification. This can be implemented by calling `PDFUI.registerSignHandler(signerInfo)` or `PDFDoc.sign(signInfo,digestSignHandler)`.
 4. Write the signedData into the file stream. The signedData's position is specified in byteRange.
- Verify signature
 1. Get the original(unmodified) file content, the byteRange of signature, the signed data and signer.
 2. Calculate the message digest of the content covered by signature's byteRange. This can be implemented by calling `PDFUI.setVerifyHandler(verifyFunction)` or `PDFDoc.verifySignature(signatureField, verifyHandler)`.
 3. Verify the digest and signed data, and output the verified state result which includes information about document changes, issuer and timestamp status, ect.. This can be implemented by calling `PDFUI.setVerifyHandler(verifyFunction)` or `PDFDoc.verifySignature(signatureField, verifyHandler)`.

Related digital signature APIs

PDFUI.registerSignHandler(signerInfo)

This method is used to register signer data. Here is the example code:

```
pdfui.registerSignHandler({
  filter: "Adobe.PPKLite",
  subfilter: "adbe.pkcs7.sha1",
  flag: 0x100,
  distinguishName: "support@foxitsoftware.com",
  location: "FZ",
  reason: "Test",
  signer: "web sdk",
  sign: (signInfo, buffer) => {
    //sign handler which complete the signing action, return a Promise with signed data;
    //function getDigest() and sign() should be completed by user.
    let digest = getDigest(buffer);
    let signData = sign(digest);
    return Promise.resolve(signData);
  },
});
```

PDFUI.setVerifyHandler(verifyFunction)

This method is used to set verification handler which will be called when a signature is being verified. Verification handler returns a verifying result state called Signature_State. Here is the example code:

```
pdfui.setVerifyHandler((signatureField, plainBuffer, signedData) => {
  //function getDigest() and verify() should be completed by user.
  let digest = getDigest(plainBuffer);
  let verifiedStatus = verify(
    signatureField.getFilter(),
    signatureField.getSubfilter(),
    signatureField.getSigner(),
    digest,
    signedData
  );
  return Promise.resolve(verifiedStatus);
});
```

PDFDoc.sign(signInfo,digestSignHandler)

This method is used to sign the document. A message digest and sign function are required. Here is the example code:


```
/**
 * @returns {Blob} - File stream of signed document.
 */
const signResult= await pdfdoc.sign(signInfo,buffer) => {
  //function getSignData() should be completed by developer.
  return Promise.resolve(getSignData(signInfo,buffer))
});
```

PDFDoc.verifySignature(signatureField, verifyHandler)

This method is used to verify the signature. A callback function is required. Here is the example code:

```
/**
 * @returns {number} - Signature state.
 */
var result = await signedPDF.verifySignature(
  pdfform.getField("Signature_0"),
  function verify(signatureField, plainBuffer, signedData, hasDataOutOfScope) {
    //function verifySignData() should be completed by developer.
    let signInfo = {
      byteRange: signatureField.getBytesRange(),
      signer: signatureField.getSigner(),
      filter: signatureField.getFilter(),
      subfilter: signatureField.getSubfilter(),
    };
    return Promise.resolve(verifySignData(signInfo, buffer));
  }
);
```

PDFSignature Class

- PDFSignature.isSigned() - Check if the current signature is signed or not.
- PDFSignature.getBytesRange() - Get byte range which specifies scope of file stream of current signature.
- PDFSignature.getFilter() - Get the current signature filter.
- PDFSignature.getSubfilter() - Get the current signature subfilter.

Interact with the digital signature feature

You can try our signature workflow by the way of using API or UI. This workflow is based on the Node.js backend which can be accessed at `./server/pkcs7` in our package.

Method 1 Programmatically place a signature on the current document

- Run `https://webviewer-demo.foxitsoftware.com/` with starting a service.
- Run the following code on the console. A signature field will be automatically created and a digital signature will be placed on it.
- A signed document will be downloaded and reopened in your viewer. You can click on the signature field to verify it.

```
//this code example assumes you are running the signature service on a local host and using the default port 7777.
var pdfviewer = await pdfui.getPDFViewer();
var pdfdoc = await pdfviewer.getCurrentPDFDoc();
var signInfo = {
  filter: "Adobe.PPKLite",
  subfilter: "adbe.pkcs7.sha1",
  rect: { left: 10, bottom: 10, right: 300, top: 300 },
  pageIndex: 0,
  flag: 511,
  signer: "signer",
  reason: "reason",
  email: "email",
  distinguishName: "distinguishName",
  location: "loc",
  text: "text",
};
const signResult = await pdfdoc.sign(signInfo, (signInfo,buffer) => {
  return requestData(
    "post",
    "http://127.0.0.1:7777/digest_and_sign",
    "arraybuffer",
    { plain: new Blob([buffer]) }
  );
});
//open the signed PDF
const signedPDF = await pdfviewer.openPDFByFile(signResult);
var pdfform = await signedPDF.loadPDFForm();
var verify = (signatureField, plainBuffer, signedData, hasDataOutOfScope) => {
  return requestData("post", "http://127.0.0.1:7777/verify", "text", {
    filter: signatureField.getFilter(),
    subfilter: signatureField.getSubfilter(),
    signer: signatureField.getSigner(),
    plainContent: new Blob([plainBuffer]),
    signedData: new Blob([signedData]),
  });
};
var result = signedPDF.verifySignature(pdfform.getField("Signature_0"), verify);
```

Method 2 Place a signature from the UI

Let's use our online viewer <https://webviewer-demo.foxitsoftware.com/> to experience how it works.

- Preparation
 - Open <https://webviewer-demo.foxitsoftware.com/> on your browser.
- Add and sign a signature
 1. Click the signature button in the Form tab to switch to the addSignatureStateHandler.
 2. Click to draw a rectangle field on the page.
 3. Click Hand tool or press **Esc** key to switch to the handStateHandler.
 4. Set the sign information on the pop-up box and click Ok to sign it. The signed document will be downloaded and re-opened automatically.
- Verify signature
 - Click the signed signature field with the hand tool to verify it. A prompt box will be pop-up reporting the verifying result.

Note: To make this signature workflow work, we have referenced the following callback code in the *index.html* file of the *complete_webViewer*, and run a signature service on our backend.

```
//the variable `origin` refers to the service http address where your signature service is running.
//signature handlers
var requestData = (type, url, responseType, body) => {
  return new Promise((res, rej) => {
    var xmlHttp = new XMLHttpRequest();
    xmlHttp.open(type, url);

    xmlHttp.responseType = responseType || "arraybuffer";
    let formData = new FormData();
    if (body) {
      for (let key in body) {
        if (body[key] instanceof Blob) {
          formData.append(key, body[key], key);
        } else {
          formData.append(key, body[key]);
        }
      }
    }
  })
}

xmlHttp.onload = (e) => {
  let status = xmlHttp.status;
  if ((status >= 200 && status < 300) || status === 304) {
    res(xmlHttp.response);
  }
}
```

```
};
xmlHttp.send(body ? formData : null);
});
};
//set signature information and function. This function can be called to register different algorithm and
information for signing
//the api `/digest_and_sign` is used to calculate the digest and return the signed data
pdfui.registerSignHandler({
  filter: "Adobe.PPKLite",
  subfilter: "adbe.pkcs7.sha1",
  flag: 0x100,
  distinguishName: "e=zhiquan_ye@foxitsoftware.cn",
  location: "FZ",
  reason: "Test",
  signer: "web sdk",
  showTime: true,
  sign: (setting, buffer) => {
    return requestData("post", "origin", "arraybuffer", {
      plain: new Blob([buffer]),
    });
  },
});
//set signature verification function
//the api /verify is used to verify the state of signature
pdfui.setVerifyHandler((signatureField, plainBuffer, signedData) => {
  return requestData("post", "origin", "text", {
    filter: signatureField.getFilter(),
    subfilter: signatureField.getSubfilter(),
    signer: signatureField.getSigner(),
    plainContent: new Blob([plainBuffer]),
    signedData: new Blob([signedData]),
  });
});
```

About signature HTTP service

If you don't have backend signature service available, you can use the following HTTP service routes which we provide for the test purpose.

Server in US:

http://webviewer-demo.foxitsoftware.com/signature/digest_and_sign

<http://webviewer-demo.foxitsoftware.com/signature/verify>

https://webviewer-demo.foxitsoftware.com/signature/digest_and_sign

<https://webviewer-demo.foxitsoftware.com/signature/verify>

Server in China:

http://webviewer-demo.foxitsoftware.cn/signature/digest_and_sign

<http://webviewer-demo.foxitsoftware.cn/signature/verify>

Import and Export

Annotation

The Annotation supports three types of files to import/export data: XFDF, FDF and JSON. The following table lists what annotations currently don't support to import/export.

File Type	If all annots support	What not support
XFDF/FDF	Mostly	Link
JSON	Mostly	Screen Image, Link, Sound

API

The following table list APIs that Foxit PDF SDK for Web provides to import/export data file.

Method	XFDF/FDF	JSON	JSON
Import	PDFDoc.importAnnotsFromFDF()	PDFDoc.importAnnotsFromJSON(annotsjson)	PDFPage.addAnnot(annotjson)
Export	PDFDoc.exportAnnotsToFDF()	PDFDoc.exportAnnotsToJSON()	Annot.exportToJson()

It is recommended to use a corresponding method to import and export data. For example, If `PDFDoc.exportAnnotsToJSON()` is called to export data, then it would better the `PDFDoc.importAnnotsFromJSON(annotsjson)` is used to import.

Note: Adding exported JSON data to the document via the `PDFPage.addAnnot` method can lead to loss of binary data streams for some annotations, such as Stamp and fileAttachment. This is because the `PDFPage.addAnnot` method does not support JSON data that contains binary streams. Therefore, if the data exported by `PDFDoc.exportAnnotsToJSON` contains binary streams, then it cannot be passed to the `PDFPage.addAnnot` method.

```
var pdfViewer = await pdfui.getPDFViewer();
var test = {ExportDataFile:'http://pathToSourceFile.pdf',ImportDatafile:'http://pathToTargetFile.pdf'};
var resp = await fetch(test.ExportDataFile);
var file = await resp.blob();
```

```
var pdfdoc = await pdfViewer.openPDFByFile(file);
var annotJson = await pdfdoc.exportAnnotsToJSON();
var newResp = await fetch(test.ImportDatafile);
var newFile = await newResp.blob()

var newPdfdoc = await pdfViewer.openPDFByFile(newFile);
for(var i=0;i<annotJson.length;i++){
    var newPage = await newPdfdoc.getPageByIndex(annotJson[i].page);
    var newAnnot = await newPage.addAnnot(annotJson[i]);
}
```

Form

The Form supports three standard types of files to import/export data: XFDF, FDF and XML.

API

The following table list APIs that Foxit PDF SDK for Web provides to import/export data file.

- PDFDoc.exportFormToFile(fileType)
- PDFDoc.importFormFromFile(file, isXML)

Stamp and customization

Foxit PDF SDK for Web provides a wide range of stamp features that users can implement with the APIs and default icons. This section will walk you through how manage stamps and add a stamp into PDF.

Default stamp list

Foxit PDF SDK for web provides a default stamp list in Viewer as follows:

```
{
  "stamp": {
    "Static": {
      "Approved": {
        "url": "xxx://url.url",
        "fileType": "pdf"
      },
      "Completed": {
        "url": "xxx://url.url",
        "fileType": "pdf"
      },
      "Confidential": {
        "url": "xxx://url.url",
        "fileType": "pdf"
      }
    }
  }
}
```

```
    },
    "Draft": {
        "url": "xxx://url.url",
        "fileType": "pdf"
    },
    "Revised": {
        "url": "xxx://url.url",
        "fileType": "pdf"
    },
    "Emergency": {
        "url": "xxx://url.url",
        "fileType": "pdf"
    },
    "Expired": {
        "url": "xxx://url.url",
        "fileType": "pdf"
    },
    "Final": {
        "url": "xxx://url.url",
        "fileType": "pdf"
    },
    "Received": {
        "url": "xxx://url.url",
        "fileType": "pdf"
    },
    "Reviewed": {
        "url": "xxx://url.url",
        "fileType": "pdf"
    }
},
"SignHere": {
    "Accepted": {
        "url": "xxx://url.url",
        "fileType": "pdf"
    },
    "Initial": {
        "url": "xxx://url.url",
        "fileType": "pdf"
    },
    "Rejected": {
        "url": "xxx://url.url",
        "fileType": "pdf"
    },
    "SignHere": {
        "url": "xxx://url.url",
        "fileType": "pdf"
    },
    "Witness": {
        "url": "xxx://url.url",
        "fileType": "pdf"
    }
}
```

```
    },  
    "Dynamic": {  
      "Approved": {  
        "url": "xxx://url.url",  
        "fileType": "pdf"  
      },  
      "Confidential": {  
        "url": "xxx://url.url",  
        "fileType": "pdf"  
      },  
      "Received": {  
        "url": "xxx://url.url",  
        "fileType": "pdf"  
      },  
      "Reviewed": {  
        "url": "xxx://url.url",  
        "fileType": "pdf"  
      },  
      "Revised": {  
        "url": "xxx://url.url",  
        "fileType": "pdf"  
      }  
    }  
  }  
}
```

Manage Stamp list

The default stamp list doesn't allow changes. However, you can create your own stamps to replace the default ones, and then edit them. The first step is to make a PDF and corresponding svg file. You can refer to the examples in `lib\stamps\en-US\DynamicStamps` folder.

Create a custom stamp list

A custom stamp list can be predefined by calling the API `pdfViewer.initAnnotationIcons()` and loaded into the viewer. Once the following code runs, the default stamp list will be overwritten.

```
var initIcons = {  
  MyCategory1: {  
    StampName1: {  
      filetype: "jpg",  
      url: "http://stamp.jpg"  
    }  
  },  
  MyCategory2: {  
    StampName2: {  
      fileType: "png",  
      url: "stamp.png"  
    }  
  }  
},
```



```
...
};
var pdfViewer = await pdfui.getPDFViewer();
await pdfViewer.initAnnotationIcons({ stamp: initIcons });
```

Remove custom stamps

```
//remove a stamp with the category and name as 'MyCategory1' and 'StampName1' from you stamp list
var pdfViewer = await pdfui.getPDFViewer();
await pdfViewer.removeAnnotationIcon('stamp','MyCategory1','StampName1')
```

```
//clear the whole stamp list
var pdfViewer = await pdfui.getPDFViewer();
await pdfViewer.removeAnnotationIcon('stamp','', 'StampName1')
```

```
//clear all stampes under 'MyCategory1'
var pdfViewer = await pdfui.getPDFViewer();
await pdfViewer.removeAnnotationIcon('stamp','MyCategory1',")
```

Add a new custom stamp

```
var icons = {
  annotType: "stamp",
  fileType: "png",
  url: "http://stamp.png",
  // width:80,
  // height:30,
  category: "MyCategory",
  name: "MyStamp"
};
var pdfViewer = await pdfui.getPDFViewer();
await pdfViewer.addAnnotationIcon(icons);
```

About the stamp category and name

Stamps are organized by category and name. To find out what stamps already exist in your list, the easy way is to check the category and name information by calling `pdfui.getAnnotationIcons()`. Here are code samples.

Get the stamp category and name

```
//list all available stamps
await pdfui.getAnnotationIcons("stamp", false);
```

```
//list only custom stamps
await pdfui.getAnnotationIcons("stamp", true);
```

You also execute the following code to output the existing stamps.

```
var allIcons = pdfui.getAnnotationIcons("stamp", false);
```

```
var iconNames = [];  
for (var categoryKey in allIcons) {  
    var category = allIcons[categoryKey];  
    for (var name in category) {  
        iconNames.push({  
            category: categoryKey,  
            name  
        });  
    }  
}  
console.log(iconNames);
```

Add a stamp onto page in Viewer

The stamp list can be found by dropping down the stamp tool under Comment tab in Viewer. You can click a stamp icon and place it to a desired place on the page.

If you want to create a custom stamp and add it onto page, follow the steps below:

1. In Advanced Web Viewer, drop down the stamp tool under Comment tab, select **Custom Stamps**.
2. In the pop-up **Create Custom Stamp** dialog box, click **File -> Browse...** and choose an image file, or click **File -> Enter File URL**, input the URL address where the PDF and svg files are stored.
3. Fill in the category, name, width and height, as well as choose a type from drop-down menu to create a stamp. Then, the created stamp will appear in the Stamp list.
4. Click the created stamp icon and place it to a desired place on the page.

Add a custom stamp onto page by API

Before calling the `PDFPage.addAnnot` to add a custom stamp which doesn't exist in your stamp list, you should call `PDFViewer.addAnnotationIcon()` to add it into stamp list. If not doing this, the stamp appearance will display incorrectly on the page.

```
var icons = {  
    annotType: "stamp",  
    fileType: "png",  
    url: "http://stamp.png",  
    // width:80,  
    // height:30,  
    category: "MyCategory",  
    name: "MyStamp"  
};  
  
var stamp = {  
    type:'stamp',
```

```
rect:{left:0,bottom:0,right:200,top:100},
icon:'MyStamp',
iconCategory:'MyCategory'
});

var pdfViewer = await pdfui.getPDFViewer();
var pdfDoc = await pdfViewer.getCurrentPDFDoc();
var page = await pdfDoc.getPageByIndex(0);

await pdfViewer.addAnnotationIcon(icons);
await page.addAnnot(stamp)
```

If you only want to add a new stamp onto the page without adding the stamp icon in your stamp list of your viewer, you can run the following code:

```
pdfpage.addAnnot({
  type: PDFViewCtrl.PDF.annots.constant.Annot_Type.stamp,
  rect: { left: 0, right: 300, top: 450, bottom: 0 },
  iconInfo: {
    annotType: PDFViewCtrl.PDF.annots.constant.Annot_Type.stamp,
    category: "category",
    name: "name",
    fileType: "pdf",
    url: "http://path/file.pdf"
  }
});
```

Set the default tool to a particular stamp in Viewer

Use the PDFUI constructor option to define a stamp as the default tool handler:

```
pdfui = new UIExtension.PDFUI({
  customs: {
    defaultStateHandler: PDFViewCtrl.STATE_HANDLER_NAMES.STATE_HANDLER_CREATE_STAMP
    handlerParams: {
      category: 'SignHere',
      name: 'SignHere'
    }
  }
});
```

Use the API `StateHandlerManager.switchTo()` to set default tool:

```
pdfui.getStateHandlerManager().then(handlerManager =>
  handlerManager.switchTo(
    PDFViewCtrl.STATE_HANDLER_NAMES.STATE_HANDLER_CREATE_STAMP,
    {
      category: "SignHere",
      name: "SignHere"
      url: "http://xxx/xx.png", // or "blob:http://xxxxx"
      showUrl: "http://xxx/xx.png", // or "blob:http://xxxxx"
    }
  )
);
```

```

        fileType:'png',
        width: 80,
        height: 30,
    })
};

```

Related APIs

APIs	Description
PDFViewer.initAnnotationIcons(icons)	Initialize the icon of the annotation (after setting, the default icon will not be displayed)
PDFViewer.addAnnotationIcon(icon)	Add a single icon
PDFViewer.removeAnnotationIcon(type,category, name)	Remove a single icon
PDFUI.getAnnotationIcons(annotType,onlyCustomized)	Get custom icon
StateManager.switchTo(name,params)	Switch to addStampStateManager
PDFViewer.setFormatOfDynamicStamp(seperator,timeFormat)	Set the format of dynamic information
PDFPage.addAnnot(json)	Add stamp with specifying the existing icon as the style of stamp

Custom Speech Synthesizer

The speech synthesizer is a text-to-speech service that is used to convert text into sounds that approximate the sound of human speech. It can work with the Read Aloud feature to provide a powerful text-to-speech function which can read aloud page contents.

Depending on the speech technology, the sounds generated may be somewhat stilted and artificial sounding, or sound very much like the voice of a real person.

To better demonstrate how you could use different text-to-speech technology with our Foxit PDF SDK for Web, we take browser native [Web Speech API](#) as an example in the following section [Customize PDFTextToSpeechSynthesis](#), and use the Google cloud [text-to-speech API](#) in the section of [Integrating 3rd Party TTS Service](#).

Speech Synthesizer APIs

PDFTextToSpeechSynthesis Interface Specification

```
interface PDFTextToSpeechSynthesis {
  status: PDFTextToSpeechSynthesisStatus;
  supported(): boolean;
  pause(): void;
  resume(): void;
  stop(): void;
  play(utterances: IterableIterator<Promise<PDFTextToSpeechUtterance>>, options?: ReadAloudOptions):
  Promise<void>;
  updateOptions(options: Partial<ReadAloudOptions>): void;
}
```

1. status Properties

The `status` enumerates the current reading aloud state. It can be defined as below:

```
enum PDFTextToSpeechSynthesisStatus {
  playing, paused, stopped,
}
```

The default value is `stopped`.

2. supported():boolean Method

This method is used to detect if the `PDFTextToSpeechSynthesis` is supported in your current client environment. If there is a 3rd party speech service running in the background, you only need to check if `HTML<audio>` is supported on your client side.

Note: The client here could either be a browser, or something others such as Electron, Apache Cordova, etc.

Code Example:

```
class CustomPDFTextToSpeechSynthesis {
  supported(): boolean {
    return typeof window.HTMLAudioElement === 'function';
  }
  // .... other methods
}
```

3. pause(), resume and stop() Methods

These methods are used to control the state of reading aloud. Through these methods, the `PDFTextToSpeechSynthesis` can manage the voice media to pause, resume, stop, and specify the `status` property.

4. `updateOptions(options: Partial<ReadAloudOptions>)` Method

This method is used to update the `PDFTextToSpeechSynthesis` in the reading aloud state, such as change the voice volume.

5. `play(utterances: IterableIterator<Promise<PDFTextToSpeechUtterance>>, options?: ReadAloudOptions): Promise<void>` Method

Parameter Description:

1. `utterances`: This is an `IterableIterator` that contains the content of the text to be read as well as the page number and coordinate information, which can be used with `for...of` to iterate.
2. `options`: This is an optional parameter that contains the speed, pitch, volume of the playback and the 'external' parameter, where 'external' is the parameter object passed to the third party speech synthesizer service.

Customize PDFTextToSpeechSynthesis

Method 1: Implement PDFTextToSpeechSynthesis interface

Notice: This demo only supports in Chrome, Firefox, Chromium Edge.

```
<html>
</html>
<script>
  const PDFTextToSpeechSynthesisStatus =
UIExtension.PDFViewCtrl.readAloud.PDFTextToSpeechSynthesisStatus;
  class CustomPDFTextToSpeechSynthesis {
    constructor() {
      this.playingOptions = {};
      this.status = PDFTextToSpeechSynthesisStatus.stopped;
    }
    supported() {
      return typeof window.speechSynthesis !== 'undefined';
    }
    pause() {
      this.status = PDFTextToSpeechSynthesisStatus.paused;
      window.speechSynthesis.pause();
    }
    resume() {
      this.status = PDFTextToSpeechSynthesisStatus.playing;
      window.speechSynthesis.resume();
    }
    stop() {
      this.status = PDFTextToSpeechSynthesisStatus.stopped;
      window.speechSynthesis.cancel();
    }
  }
```

```
/**
 * @param {IterableIterator<Promise<PDFTextToSpeechUtterance>>}} utterances
 * @param {ReadAloudOptions} options
 *
 */
async play(utterances, options) {
  for await (const utterance of utterances) {
    const nativeSpeechUtterance = new window.SpeechSynthesisUtterance(utterance.text);
    const { pitch, rate, volume } = Object.assign(
      {}, this.playingOptions, options || {}
    );
    if(typeof pitch === 'number') {
      nativeSpeechUtterance.pitch = pitch;
    }
    if(typeof rate === 'number') {
      nativeSpeechUtterance.rate = rate;
    }
    if(typeof volume === 'number') {
      nativeSpeechUtterance.volume = volume;
    }
    await new Promise((resolve, reject) => {
      nativeSpeechUtterance.onend = resolve;
      nativeSpeechUtterance.onabort = resolve;
      nativeSpeechUtterance.onerror = reject;
      speechSynthesis.speak(nativeSpeechUtterance);
    });
  }
}

updateOptions(options) {
  Object.assign(this.playingOptions, options);
}

}

var libPath = window.top.location.origin + '/lib';
var pdfui = new UIExtension.PDFUI({
  viewerOptions: {
    libPath: libPath,
    jr: {
      licenseSN: licenseSN,
      licenseKey: licenseKey
    }
  },
  renderTo: document.body,
  appearance: UIExtension.appearances.ribbon,
  addons: [
    libPath + '/uix-addons/read-aloud'
  ]
});

pdfui.getReadAloudService().then(function(service) {
  service.setSpeechSynthesis(new CustomPDFTextToSpeechSynthesis());
});
```

```
</script>
```

Method 2: Use `AbstractPDFTextToSpeechSynthesis` to customize the speech synthesizer

```
<html>
</html>
<script>
  const PDFTextToSpeechSynthesisStatus =
UIExtension.PDFViewCtrl.readAloud.PDFTextToSpeechSynthesisStatus;
  const AbstractPDFTextToSpeechSynthesis =
UIExtension.PDFViewCtrl.readAloud.AbstractPDFTextToSpeechSynthesis;
  const CustomPDFTextToSpeechSynthesis = AbstractPDFTextToSpeechSynthesis.extend({
    init(){},
    supported() {
      return typeof window.speechSynthesis !== 'undefined';
    },
    doPause() {
      window.speechSynthesis.pause();
    },
    doResume() {
      window.speechSynthesis.resume();
    },
    doStop() {
      window.speechSynthesis.cancel();
    },
  },
  /**
   * @param {string} text
   * @param {ReadAloudOptions | undefined} options
   */
  async speakText(text, options) {
    const nativeSpeechUtterance = new window.SpeechSynthesisUtterance(utterance.text);
    const { pitch, rate, volume } = Object.assign(
      {}, this.playingOptions, options || {}
    );
    if(typeof pitch === 'number') {
      nativeSpeechUtterance.pitch = pitch;
    }
    if(typeof rate === 'number') {
      nativeSpeechUtterance.rate = rate;
    }
    if(typeof volume === 'number') {
      nativeSpeechUtterance.volume = volume;
    }
    await new Promise((resolve, reject) => {
      nativeSpeechUtterance.onend = resolve;
      nativeSpeechUtterance.onabort = resolve;
      nativeSpeechUtterance.onerror = reject;
      speechSynthesis.speak(nativeSpeechUtterance);
    });
  }
})
```



```
const libPath = window.top.location.origin + '/lib';
const pdfui = new UIExtension.PDFUI({
  viewerOptions: {
    libPath: libPath,
    jr: {
      licenseSN: licenseSN,
      licenseKey: licenseKey
    }
  },
  renderTo: document.body,
  appearance: UIExtension.appearances.ribbon,
  addons: [
    libPath + '/uix-addons/read-aloud'
  ]
});
pdfui.getReadAloudService().then(function(service) {
  service.setSpeechSynthesis(new CustomPDFTextToSpeechSynthesis());
});
</script>
```

The Difference of `PDFTextToSpeechSynthesis` and `AbstractPDFTextToSpeechSynthesis`

The Method 1 customizes speech synthesizer by implementing the interface `PDFTextToSpeechSynthesis`. It needs to manually manage state changes as well as iterate the list of 'utterances' by `for await...of`. Each item in the 'Utterance' list is a text block obtained from `PDFPage`. In some cases, the text block may just contain a part of a word or sentence, which requires merging text blocks to build up a complete word and sentence for better speech synthesizing. This merging operation can be completed in the `play()` method.

The Method 2 customizes speech synthesizer by inheriting the `AbstractPDFTextToSpeechSynthesis` abstract class. It doesn't require to manually manage state and iterate utterances list, but needs correctly call `window.SpeechSynthesisUtterance` to generate speech and play the voice based on the received text and parameters. These received text blocks will be automatically merged by `AbstractPDFTextToSpeechSynthesis`. However currently it is tough to guarantee that all the combined text blocks in different language environments would comprise complete words or sentences, as such if you are strict with reading correctness with each sentence and word, you are recommended to use Method 1.

Integrate with 3rd Party TTS Service

We take [@google-cloud/text-to-speech](https://cloud.google.com/text-to-speech) as an example in this section.

Server

To start with Google Cloud Text-to-Speech server library with favorite programming language, refer to <https://cloud.google.com/text-to-speech/docs/quickstarts>.

Client

```
var readAloud = UIExtension.PDFViewCtrl.readAloud;
var PDFTextToSpeechSynthesisStatus = readAloud.PDFTextToSpeechSynthesisStatus;
var AbstractPDFTextToSpeechSynthesis = readAloud.AbstractPDFTextToSpeechSynthesis;
var SPEECH_SYNTHESIS_URL = '<server url>'; // the server API address

var ThirdpartyPDFTextToSpeechSynthesis = AbstractPDFTextToSpeechSynthesis.extend({
  init: function() {
    this.audioElement = null;
  },
  supported: function() {
    return typeof window.HTMLAudioElement === 'function' && document.createElement('audio') instanceof
window.HTMLAudioElement;
  },
  doPause: function() {
    if(this.audioElement) {
      this.audioElement.pause();
    }
  },
  doStop: function() {
    if(this.audioElement) {
      this.audioElement.pause();
      this.audioElement.currentTime = 0;
      this.audioElement = null;
    }
  },
  doResume: function() {
    if(this.audioElement) {
      this.audioElement.play();
    }
  },
  onCurrentPlayingOptionsUpdated: function() {
    if(!this.audioElement) {
      return;
    }
    var options = this.currentPlayingOptions;
    if (this.status === PDFTextToSpeechSynthesisStatus.playing) {
      if(options.volume >= 0 && options.volume <= 1) {
        this.audioElement.volume = options.volume;
      }
    }
  },
  speakText: function(text, options) {
    var audioElement = document.createElement('audio');
    this.audioElement = audioElement;
    if(options.volume >= 0 && options.volume <= 1) {
      audioElement.volume = options.volume;
    }
    return this.speechSynthesis(text, options).then(function(src) {
      return new Promise(function(resolve, reject) {
        audioElement.src = src;
        audioElement.onended = function() {
```

```

        resolve();
    };
    audioElement.onabort = function() {
        resolve();
    };
    audioElement.onerror = function(e) {
        reject(e);
    };
    audioElement.play();
    }).finally(function() {
        URL.revokeObjectURL(src);
    });
});
},
// If the server API request method or parameter form is not consistent with the following implementation, it
will need to be adjusted accordingly.
speechSynthesis: function(text, options) {
    var url = SPEECH_SYNTHESIS_URL + '?' + this.buildURIQueries(text, options);
    return fetch(url).then(function(response) {
        if(response.status >= 400) {
            return response.json().then(function(json) {
                return Promise.reject(JSON.parse(json).error);
            });
        }
        return response.blob();
    }).then(function (blob) {
        return URL.createObjectURL(blob);
    });
},
buildURIQueries: function(text, options) {
    var queries = [
        'text=' + encodeURIComponent(text)
    ];
    if(!options) {
        return queries.join('&');
    }
    if(typeof options.rate === 'number') {
        queries.push('rate=' + options.rate);
    }
    if(typeof options.spitch === 'number') {
        queries.push('spitch=' + options.spitch);
    }
    if(typeof options.lang === 'string') {
        queries.push('lang=' + encodeURIComponent(options.lang));
    }
    if(typeof options.voice === 'string') {
        queries.push('voice=' + encodeURIComponent(options.voice));
    }
    if(typeof options.external !== 'undefined') {
        queries.push('external=' + encodeURIComponent(JSON.stringify(options.external)));
    }
}

```

```
    return queries.join('&');  
  }  
});
```

Use the custom speech synthesizer:

```
pdfui.getReadAloudService().then(function(service) {  
  service.set(new ThirdpartyPDFTextToSpeechSynthesis());  
});
```

Collaboration

Getting started with collaboration

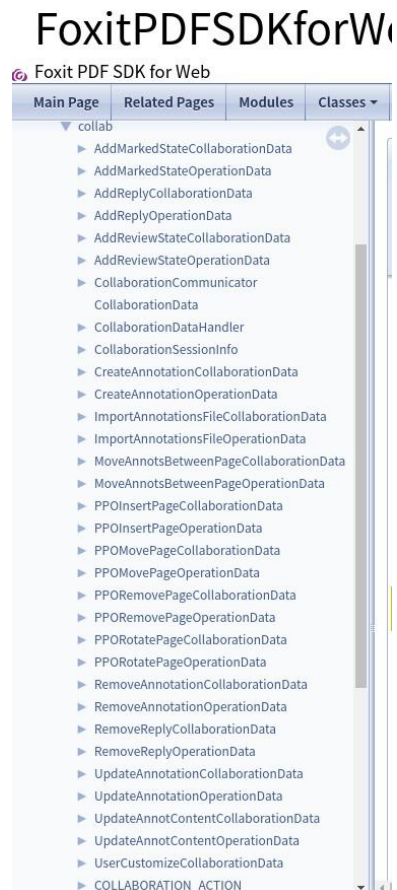
Quickly run collaboration example

Please refer to [quickly-run-samples](#) to start the service, and then open the following address in the browser to access the collaboration example:

<http://127.0.0.1:8080/examples/UIExtension/collaboration/index.html>

Working principle

First, open the API reference, there are collaboration related interfaces and classes as shown below:



Among them, the operative keys are these:

6. CollaborationCommunicator

This interface is the communication bridge between the browser and the server. It is responsible for connecting to the server, creating a collaborative session, synchronizing data, etc. In the `/examples/UIExtension/collaboration` directory, there is a sockjs based communicator, you may refer to its implementation for more details.

7. CollaborationDataHandler

Suppose we have two client devices: client A and client B. When client A creates an annotation, its related annotation data will be sent to the server, where data will be further forwarded to the client B. When client B receives the data, it immediately repeats the same operation as the client A based on the data content. At this point, the data synchronization is realized. The above operation process of client B is implemented by the `CollaborationDataHandler` interface, which defines two methods:

- `accept(data: CollaborationData): Promise<boolean>`

This method will be called and used to determine whether the current `CollaborationDataHandler` interface should handle the data when the data arrives.

- `receive(data: CollaborationData, builtinHandler?: CollaborationDataHandler)`

When the `accept()` method returns true, this method will be called with two parameters. The first is the currently received collaborative data and the second is a built-in data handler. In some custom cases, you should decide whether to call the built-in data handler, since if the built-in data handler is called when the operation action is not defined in `COLLABORATION_ACTION`, there is no longer to do other synchronizing operations.

8. CollaborationSessionInfo

When the client opens a PDF file, it will send a request to the server to create a new session. This session contains a unique shareId and information about the current PDF file. In this way, after sharing the link with the shareId to other users, the other users can open this link and see the original PDF file and start the collaboration session.

9. UserCustomizeCollaborationData

Our API design allows users to customize collaborative operations. You can send custom collaborative data through the `PDFViewer.collaborate` API, and then receive the custom collaborative data in the custom `CollaborationDataHandler` to perform synchronization.

10. COLLABORATION_ACTION

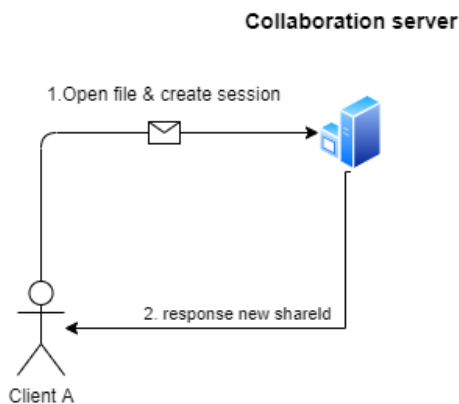
Enumerations of the built-in collaboration actions.

How does the collaboration work? The following section explain a complete process in details.

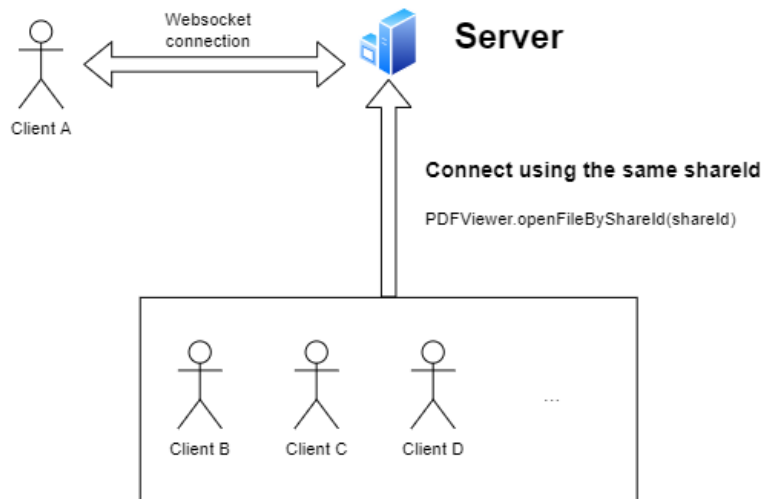
Note: Our collaboration example uses Node.js as the backend server and use websocket as a communication channel to establish a long connection for collaboration.

3. Client A opens a PDF document, where the document information will be sent to server to start a new session with a unique shareId. In the browser, you will see the address like this:

<http://localhost:8080/examples/UIExtension/collaboration/index.html?shareId=2MF8Rp7-Q>.



4. Client A shares the collaborative link to other clients, who will be connected to the same server once they open the link.

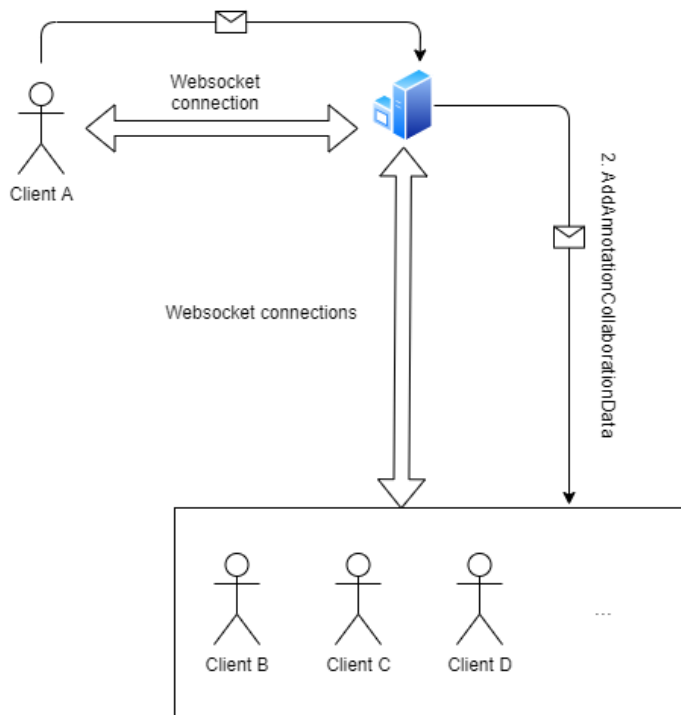


- Client A performs an action to create an annotation for example. The action operation data is sent to server synchronously on the background, and then forwarded to other clients.

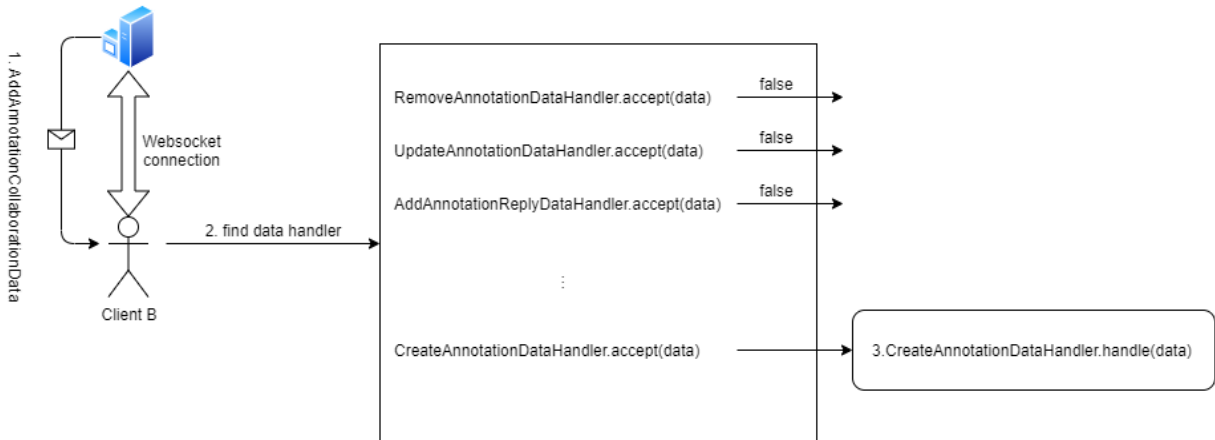
1. Create Annotation

```
pdfViewer.collaborate(
```

```
  COLLABORATION_ACTION.CREATE_ANNOT,  
  data as CreateAnnotationOperationData
```



6. When other clients receive the synchronizing data, Web SDK will find the first data handler that accept the operation data, and process the received data using the `CollaborationDataHandler`, and then output the same action of the Client A on screen.



At this point, a complete collaborative operation is completed!

The infinite loop problems

Looking at the following code, first, register the `DataEvent.annotationAdded` event to send collaborative data after the event is triggered, then register a click event on a button to create annotation via `PDFPage.addAnnot` API. The call of the `PDFPage.addAnnot` method will trigger the `DataEvent.annotationAdded` event. This logic looks perfect. However, assuming that there are two client A and B at this time. The button of client A is clicked first and a collaborative data is sent, and then client B also calls the `PDFPage.addAnnot` method after receiving the collaborative data, and then the problem appears, because the `PDFPage.addAnnot` triggered the `DataEvent.annotationAdded` event, and the event callback sent a collaborative data, all clients would continue to send request after receiving the collaborative data. At this process an infinite loop problem rises.

```

var DataEvents = PDFViewCtrl.PDF.DataEvents;
pdfViewer.eventEmitter.on(DataEvents.annotationAdded, function(annots){
    var doc = pdfViewer.getCurrentPDFDoc();
    doc.exportAnnotsToJSON(annots).then(function(annotsJSONArray) {
        pdfViewer.collaborate(PDFViewCtrl.collab.COLLABORATION_ACTION.CREATE_ANNOT, {
            annots: annotsJSONArray
        })
    });
});
var $button = jQuery('#create-square-button');
$button.on('click', function() {

```

```
var doc = pdfViewer.getCurrentPDFDoc();
doc.getPageByIndex(0).then(function(page) {
  page.addAnnot({
    color: 0xff0000,
    rect: { left: 300, right: 400, top: 300, bottom: 200 },
    flags: 4,
    type: 'square'
  });
});

var $button = jQuery('#create-line-button');
$button.on('click', function() {
  var doc = pdfViewer.getCurrentPDFDoc();
  doc.getPageByIndex(0).then(function(page) {
    page.addAnnot({
      type: 'line',
      startStyle: 0,
      endStyle: PDF.constant.Ending_Style.Square,
      startPoint: {x: 0, y: 0},
      endPoint: {x: 100, y: 100},
      rect: {
        left: 0,
        right: 100,
        top: 0,
        bottom: 100
      }
    });
  });
});
```

To avoid the problem, we should move off the call of `pdfViewer.collaborate` from the `DataEvents` callback and place it in the click event callback.

```
var $button = jQuery('#create-square-button');
$button.on('click', function() {
  var doc = pdfViewer.getCurrentPDFDoc();
  doc.getPageByIndex(0).then(function(page) {
    page.addAnnot({
      color: 0xff0000,
      rect: { left: 300, right: 400, top: 300, bottom: 200 },
      flags: 4,
      type: 'square'
    }).then(function(annots) {
      return doc.exportAnnotsToJSON(annots);
    }).then(function(annotsJSONArray) {
      pdfViewer.collaborate(PDFViewCtrl.collab.COLLABORATION_ACTION.CREATE_ANNOT, {
        annots: annotsJSONArray
      });
    });
  });
});
```

```
});  
  
var $button = jQuery('#create-line-button');  
$button.on('click', function() {  
    var doc = pdfViewer.getCurrentPDFDoc();  
    doc.getPageByIndex(0).then(function(page) {  
        page.addAnnot({  
            type: 'line',  
            startStyle: 0,  
            endStyle: PDF.constant.Ending_Style.Square,  
            startPoint: {x: 0, y: 0},  
            endPoint: {x: 100, y: 100},  
            rect: {  
                left: 0,  
                right: 100,  
                top: 0,  
                bottom: 100  
            }  
        })  
    }).then(function(annots) {  
        return doc.exportAnnotsToJSON(annots);  
    }).then(function(annotsJSONArray) {  
        pdfViewer.collaborate(PDFViewCtrl.collab.COLLABORATION_ACTION.CREATE_ANNOT, {  
            annots: annotsJSONArray  
        })  
    })  
});  
});  
});
```

Best Practice

Foxit PDF SDK for Web runs in a browser sandbox in a network environment. Choosing a correct website operation scheme and Foxit PDF SDK for Web configuration can make Foxit PDF SDK for Web run faster. The following section give references on website operation optimization and Foxit PDF SDK for Web configuration.

Website assets optimization

Gizp and Brotli compression

Compression is a way to shrink the assests size and reduce the downloading time. The following table shows the compressed size using gzip and brotil on `UIExtension.css` and `UIExtension.full.js`.

File	Original size	Gzip	Brotli
UIExtension.css	1.2M	213kb	156kb
UIExtension.full.js	2.6M	534kb	443kb

NOTE: Although the brotli compression algorithm provided by Google is superior to gzip in compression ratio. But brotli is not natively supported by all browsers, such as Microsoft's IE. Decompression of brotli in IE requires the use of a JavaScript engine. This time-consuming process offsets the advantages of Brotli and consumes website loading performance.

Cache

Caching resource files can avoid downloading the same assets again and again. The `/lib` library in the SDK and the font files in `/external` are recommended for front-end caching. To learn more, check out [Google](#) and [Mozilla](#) for HTTP cache.

Foxit PDF SDK for Web configuration

Read only

If the following scenario is your current needs, it is recommended that you load the Foxit PDF SDK for Web read-only to improve rendering performance.

Applicable scenario:

- complex PDF documents generated by CAD
- page rendering speed is high preference
- no page editing requirements

Code Example:

```
<script src="path/to/UIExtension.full.js"></script>
<script src="path/to/allInOne.js"></script>
<script>
  var pdfui = new UIExtension.PDFUI({
    ...
    viewerOptions:{
      customs: {
        getDocPermissions: function () {
          return 0; // 0 means ReadOnly
        }
      }
    }
    ...
  })
</script>
```

or

```
<script src="path/to/PDFViewCtrl.full.js"></script>
<script>
  var pdfviewer = new PDFViewCtrl.PDFViewer({
    ...
    customs: {
      getDocPermissions: function () {
        return 0; // 0 means ReadOnly
      }
    }
    ...
  })
</script>
```

Brotli compression

The core of Foxit PDF SDK for Web is the wasm/asm module compiled by emscripten. The module size is 8M / 13M, and the loading time varies depending on the browser performance. These two modules are compressed using Brotli by default. But Brotli is not natively supported by all browsers, such as Microsoft's IE, click [here](#) to see the browser support for Brotli. Decompressing brotli in IE needs to use the browser's JavaScript engine. This process takes time, and may offset the advantages of Brotli and then result a performance penalty.

It is recommended that you select the most suitable configuration by enabling and disabling Brotli in your test environment.

Code Example:

```
<script src="path/to/UIExtension.full.js"></script>
<script src="path/to/allInOne.js"></script>
<script>
  var pdfui = new UIExtension.PDFUI({
    ...
    viewerOptions:{
      jr: {
        brotli:{
          core:false,// the default value is true which means to enable brotli, false means no brotli
compression
        }
      }
    }
    ...
  })
</script>
```

or

```
<script src="path/to/PDFViewCtrl.full.js"></script>
<script>
  var pdfviewer = new PDFViewCtrl.PDFViewer({
    ...
    jr: {
      brotli:{
        core:false,// the default value is true which means to enable brotli, false means no brotli compression
      }
    }
    ...
  })
</script>
```

Preload webassembly artifacts

Starting from version 7.1.1, Foxit PDF SDK for Web provides a script file called "preload-jr-worker.js" to load webWorker scripts and wasm/asm in advance, this can greatly save document rendering time.

Code Example:

```
<body>
  <div id="pdf-ui"></div>
  <script>
    var licenseSN = "Your license SN";
    var licenseKey = "Your license Key";
  </script>

  <!-- Add the preload-jr-worker.js-->
  <script src="/lib/preload-jr-worker.js"></script>
  <script>
    var readyWorker = preloadJrWorker({
      workerPath: './lib/',
      enginePath: './lib/jr-engine/gsdk',
      fontPath: './external/brotli',
      licenseSN: licenseSN,
      licenseKey: licenseKey
    })
  </script>

  <script src="/lib/UIExtension.full.js"></script>
  <script>

    var pdfui = new UIExtension.PDFUI({
      viewerOptions: {
        libPath: './lib', // the library path of web sdk.
        jr: {
          readyWorker: readyWorker,
        }
      },
      renderTo: '#pdf-ui', // the div (id="pdf-ui").
      appearance: UIExtension.appearances.adaptive,
      addons: [
        '...'
      ]
    });
    ...
```

Tiling size

Foxit PDF SDK for Web performs raster scan when rendering the page. If the currently rendered page layout is too large, the rendering speed of the page will be extremely slow. It is recommended that you enable tileSize rendering mode when opening this large page layout. Currently, the supported tileSize range is 500-3000px. In our internal comprehensive test, the rendering speed is optimal with the tileSize being set as 1200px. But it may vary with your document complex. You can set different tileSize such as 200, 3600, and etc. in your test environment according to the needs of the actual scenario to obtain the most suitable configuration scheme.

Applicable scenario:

- Complex documents with large page layout

Code Example:

```
<script src="path/to/UIExtension.full.js"></script>
<script src="path/to/allInOne.js"></script>
<script>
  var pdfui = new UIExtension.PDFUI({
    ...
    viewerOptions:{
      tileSize:1200,
      ...
    }
    ...
  })
</script>
```

or

```
<script src="path/to/PDFViewCtrl.full.js"></script>
<script>
  var pdfviewer = new PDFViewCtrl.PDFViewer({
    ...
    tileSize:1200,
    ...
  })
</script>
```

Tiling size and zoom

Foxit PDF SDK for Web opens PDF with fitWidth by default. For large but simple documents, fitWidth zoom ratio improves rendering speed. However, for CAD type of documents with large layout and objects on a page, fitWidth may slow down the page rendering on the contrary. That is because there are more page objects are required to render in the same height of viewport -- see example

below. For this type of document, the solution we recommend is to reduce the rendering object by adjusting the page zoom ratio and tileSize, thus increasing the rendering speed. You can do test in your environment to get the most appropriate configuration for your actual scenario.

Example

Take the visual range of 800 * 600, page object 3000 * 4000, and tileSize 200 as an example, let's see how many page objects are required to render in the different zoom ratios.

Zoom	Origin of Coordinate Space	Page Objects
fitWidth	the top-left corner	3000*2400
0.5	the top-left corner	1600*1200
1	the top-left corner	800*600

Code Example:

```
<script src="path/to/PDFViewCtrl.full.js"></script>
<script>
  var pdfviewer = new PDFViewCtrl.PDFViewer({
    ...
    defaultScale = '0.5',// or '1'
    tileSize:200,
    ...
  })
</script>
```

Rendering mode

PDF has a feature of high fidelity that may compromise performance speed. To balance speed and high fidelity, Foxit PDF SDK for Web provides two rendering modes for partial annotations and forms: native (fidelity mode) and canvas (quick mode).

- **native:** Uses the WebAssembly as the rendering engine that requires a higher performance than canvas rendering.
- **canvas:** Uses HTML canvas as the rendering engine, which has some distortion in appearance, but more faster and interactive. The distortion is not perceptible to users in most cases. If you have high speed requirements, this mode is your choice.

Currently, the annotations and form widgets that support switching render mode are listed below:

- Note

- Highlight
- FileAttachment
- Sound
- PushButton
- RadioButton
- CheckBox
- TextField
- comboBox

Code Example:

```
<script src="path/to/PDFViewCtrl.full.js"></script>
<script>
  var pdfviewer = new PDFViewCtrl.PDFViewer({
    ...
    annotRenderingMode:{
      highlight: 'canvas',
      choiceButton:'canvas',
      pushButton:'native',
      textField:'native',
      sound:'canvas',
      fileAttachment:'canvas',
      note:'canvas',
    }
    ...
  })
</script>
```

Document loading

Synchronous loading

Synchronous loading is to first obtain the complete binary stream of the file for loading, which is a compromised way of memory and performance. For documents between 50M and 500M, this method is recommended.

Code Example:

```
<script src="path/to/UIExtension.full.js"></script>
<script src="path/to/allInOne.js"></script>
<script>
  var pdfui = new UIExtension.PDFUI({...})
  var blob = getBlob();
  pdfui.openPDFByFile(blob)
</script>
```

Asynchronous loading

Asynchronous loading does not require a complete file stream, only the required part is obtained during loading. When the file is too large (greater than 500MB) and cannot be put in memory at all, or when you only need to request part of the document at a time, it is recommended to load the document in this way to get a good performance experience.

Code Example:

```
<script src="path/to/UIExtension.full.js"></script>
<script>
  var pdfui = new UIExtension.PDFUI({...})
  pdfui.openPDFByHttpRequest({
    range:{
      url:'../../docs/FoxitPDFSDKforWeb_DemoGuide.pdf',
    }
  })
</script>
```

Loading document from memory arrayBuffer

Loading from arrayBuffer is to store the entire file stream to and load from in wasm/asm memory. For small local documents (less than 500MB), or when the entire document stream can be obtained in a short time, it is recommended to load in this way. This method has the advantages of high reading efficiency and fast loading speed. To enable this method, pass in the callback function `getLoadingMode()` at the time of constructing the PDFUI. When it returns 1, it means that it is loaded from memory arrayBuffer.

Code Example:

```
<script src="path/to/UIExtension.full.js"></script>
<script>
  var pdfui = new UIExtension.PDFUI({
    ...
    customs:{
      getLoadingMode:function(file){return 1}
    }
    ...
  })
</script>
```

If you have implemented your own file open control, you can use the following method to load:

```
var pdfui = new UIExtension.PDFUI({...})
...//event bind context
{
  var arrayBuffer=getArrayBuffer();
```

```
pdfui.openPDFByFile(arrayBuffer);  
}  
...
```

I18n Entries Resources Management

Explanation

- SDK: Foxit PDF SDK For Web.
- Addon: The addon features in the uix-addons directory of Foxit PDF SDK for Web.
- Entries: Defined in the JSON configuration file and placed in a directory named after the language code based on the language type.
- Application layer: The upper layer architecture developed by the SDK interface.

Overview

This section provides some details about the management of i18n entries resources. It includes:

- SDK entries resources file, namespace management
- How to add a new language
- How to rewrite some existing entries
- Customize the entries of Addon

SDK I18n Entries Resource Management

The directory structure and the role of the file

In the SDK release package, internationalized entries are placed in the 'lib/locales/' directory and sorted by language code. Create the sub-directories based on the language code:

```
lib/locales
├── en-US
├── ja-JP
└── zh-CN
```

In the language code directory, there are 'ui.json' and 'viewer.json' files. If application layer is developed based on PDFViewCtrl library, it only relies on the 'viewer.json' entry file; if application layer is developed based on UIExtension library, it relies on both the 'ui.json' and 'viewer.json' files.

The directory of the custom entry file

If the default entries of SDK cannot meet the needs of the application layer, so that you need to rewrite the entries, or add new languages. In this case, it is recommended that developers should create a new directory at the application layer to store the custom entries.

The structure of the created directory should be consistent with the entries directory structure of the SDK release package, and the name of the entries file must be `ui_.json` and `viewer_.json`, for example:

```
/custom/locales
├── en-US
│   ├── ui_.json
│   └── viewer_.json
├── ja-JP
│   ├── ui_.json
│   └── viewer_.json
└── zh-CN
    ├── ui_.json
    └── viewer_.json
```

After determining the entry directory path, specify the entry path when constructing a PDFUI or PDFViewer instance:

Based on PDFViewCtrl:

```
new PDFViewer({
  i18nOptions: {
    absolutePath: '/custom/locales/'
  }
})
```

Based on UIExtension:

```
new PDFUI({
  i18n: {
    absolutePath: '/custom/locals'
  },
})
```

Verify the configuration in developer environment

1. Clear your browser caches to ensure the latest i18N resources will be loaded.
2. Refresh your browser, open the Network panel in DevTools, and check if the `ui_.json` or `viewer_.json` request url points your custom language path. If so, it means success.

Add new languages

Based on the above method of customizing the entry file directory, for adding new languages, you should only add the language code directory in the `/custom/locales/` directory, and then write the entry file for the corresponding language according to the en-US entry.

Taking ko-KR for example, after adding new entry, the directory structure will look like:

```
/custom/locales
├── en-US
│   ├── ui.json
│   └── viewer.json
├── ja-JP
│   ├── ui.json
│   └── viewer.json
├── ko-KR
│   ├── ui.json
│   └── viewer.json
└── zh-CN
    ├── ui.json
    └── viewer.json
```

After finishing adding new languages, you can specify the default language when initializing the library:

Based on PDFViewCtrl:

```
const pdfViewer = new PDFViewer({
  i18nOptions: {
    initOption: {
      lng: 'ko-KR'
    }
  }
})
```

Based on UIExtension:

```
const pdfui = new PDFUI({
  i18n: {
    lng: 'ko-KR'
  }
})
```

In addition, you can switch languages dynamically:

```
pdfViewer.changeLanguage('ko-KR');
pdfui.changeLanguage('ko-KR');
```

Rewrite some of the entries

If most of the SDK entries can meet the requirements of the application layer, and just need to do some minor modification, then you can use the functions [addResources](#) and [addResourceBundle](#) of [i18next.js](#) to overwrite the entries.

Based on PDFViewCtrl:

```
pdfViewer.i18n.addResource('en-US', 'viewer_', 'contextmenu.hand.zoomin', 'Custom Zoom in');
pdfViewer.i18n.addResources('en-US', 'viewer_', {
  'contextmenu.hand.zoomin': 'Custom Zoom in',
  'contextmenu.hand.zoomout': 'Custom Zoom out'
});
pdfViewer.i18n.addResourceBundle('en-US', 'viewer_', {
  contextmenu: {
    hand: {
      zoomin: 'Custom Zoom in',
      zoomout: 'Custom Zoom out'
    }
  }
}, true, true);
```

Based on UIExtension:

```
pdfui.waitForInitialization().then(() => {
  pdfui.i18n.addResource('en-US', 'ui_', 'contextmenu.tools.handTool', 'Custom Hand Tool');
  pdfui.i18n.addResources('en-US', 'ui_', {
    'contextmenu.tools.handTool': 'Custom Hand Tool',
    'contextmenu.tools.selectAnnotation': 'Custom Select Annotation Tool'
  });
  pdfui.i18n.addResourceBundle('en-US', 'ui_', {
    contextmenu: {
      tools: {
        handTool: 'Custom Hand Tool',
        selectAnnotation: 'Custom Select Annotation Tool'
      }
    }
  }, true, true);
  // make the above configuration work on the interface.
  pdfui.getRootComponent().then(root => {
    root.localize();
  });
})
```

Customize the entries of Addon

For Addon, please refer to this section [Introduction to addons](#).

The following table lists all Addons and the corresponding entries namespaces:

Addon	i18n namespace
edit-graphics	ega
export-form	export
file-property	file-property
form-designer	form-designer
h-continuous	h-continuous
h-facing	h-facing
h-single	h-single
import-form	import
print	print
recognition-form	recognition-form
text-object	edit-text
thumbnail	thumbnail

When adding/overwriting the entries, you can use the namespaces in the above table to add/overwrite the entries of a specific addon, as follows:

```
pdfui.waitForInitialization().then(() => {  
  pdfui.i18n.addResourceBundle('en-US', 'print', {  
    dialog: {  
      cancel: 'custom cancel'  
    }  
  }, true, true);  
  pdfui.getRootComponent().then(root => {  
    root.localize();  
  });  
})
```

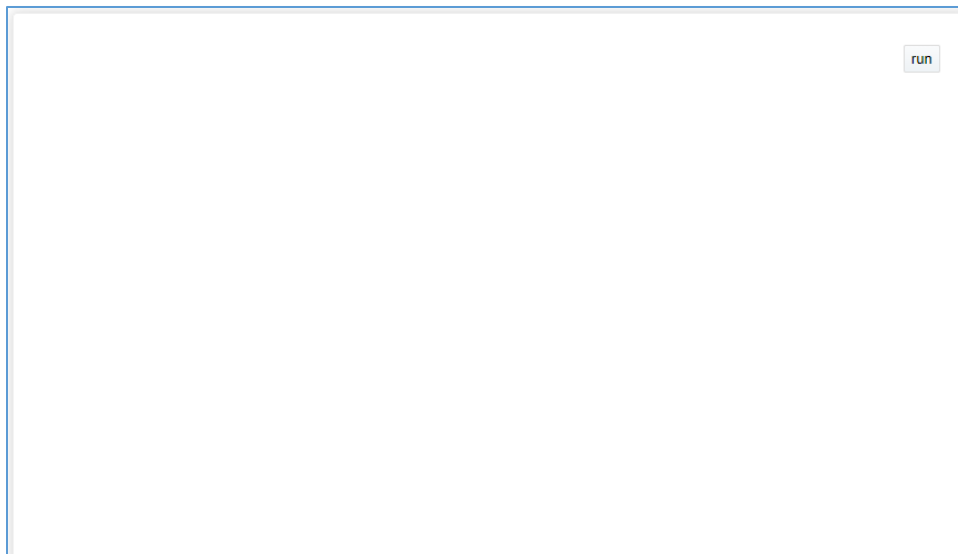
For more details about the addon entries, you can refer to the `uix-addons/{addon-name}/locales/en-US.json` file in the SDK release package.

Troubleshooting

Thumbnail Loading Error

This component is unavailable until "thumbnail" addon is loaded

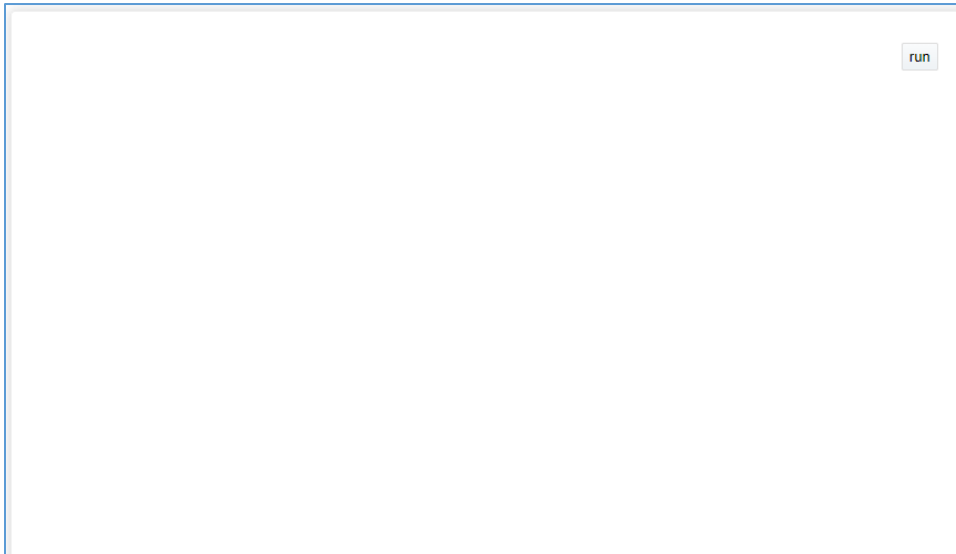
With the release of version 7.3.0, the thumbnail component was modularized as an add-on. As thus, before migrating past versions to this version or higher, thumbnail components should be configured according to actual needs. Directly migrated versions without proper changes on thumbnail component will cause an error on your browser console during the initialization phase. To view the details of the error, open the browser DevTools and click **Run** at the top right of the following demo. Note: the following demo doesn't run on legacy browsers.



Solutions

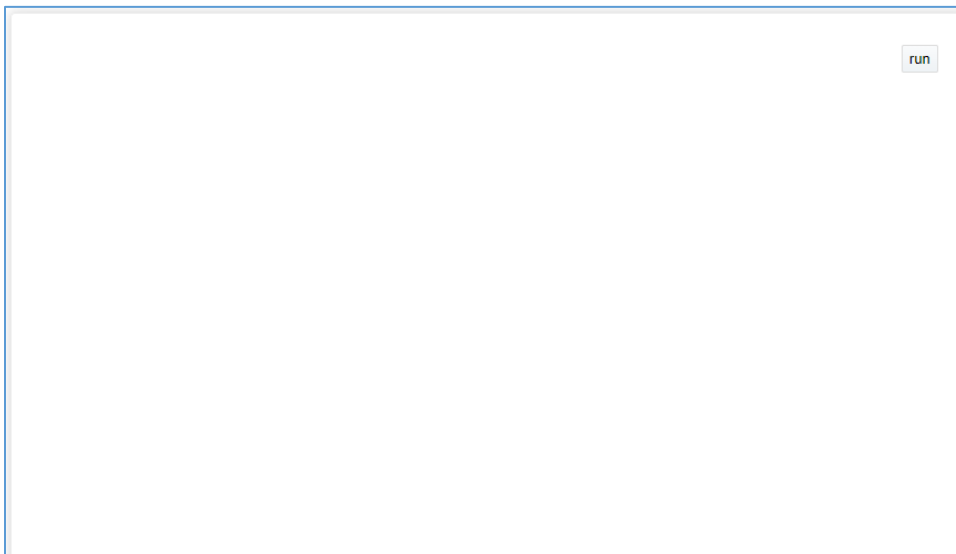
Reference thumbnail addon

If you need Thumbnails, you should reference `/uix-addons/thumbnail` when initializing PDFUI. Below is the code example:



Delete the tag `<thumbnail-sidebar-panel>` in the `layout-template` section

If you don't need thumbnail, then you should delete `<thumbnail-sidebar-panel>` tag to avoid the error. Below is the code example:



Basics

Appearance

`Appearance` is a class that defines the appearance of the UI, it provides a template to specify the layout of the UI and fragments to modify the layout and control the logic of the components.

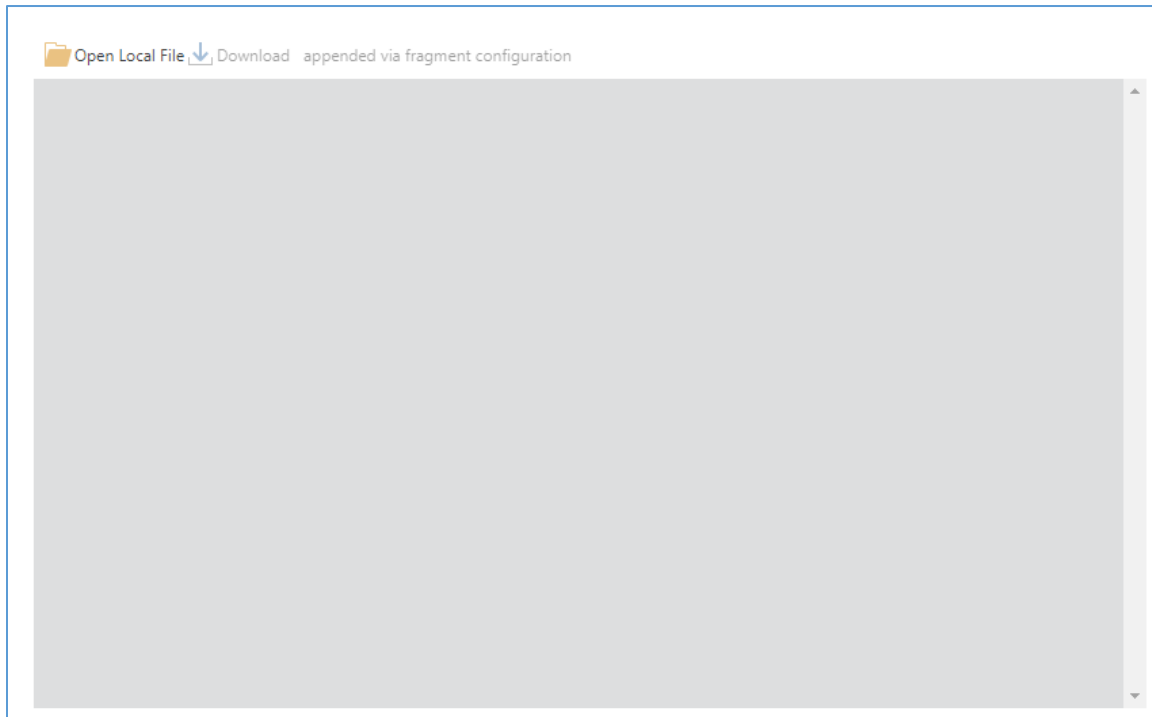
Following is the declaration of the `Appearance` class, its sub-classes should override these methods to define new appearance:

```
class Appearance {
  constructor(pdfui);
  // Layout template.
  public getLayoutTemplate: () => string;
  // Return fragment configuration.
  public getDefaultFragments: () => UIFragmentOptions[];
  // Triggered before inserting the component into the DOM tree.
  public beforeMounted: (root: Component) => void
  // Triggered after inserting the component into the DOM tree.
  public afterMounted: (root: Component) => void
  // Called to disable the component after closing PDF documents.
  protected disableAll: () => void;
  // Called to enable the component after closing PDF documents.
  protected enableAll: () => void;
}
```

Custom Appearance Example

```
<html>
  <template id="layout-template">
    <webpdf>
      <div name="toolbar" style="display: flex; flex-direction: row; padding: 6px;">
        <open-localfile-button></open-localfile-button>
        <download-file-button></download-file-button>
      </div>
      <div class="fv__ui-body">
        <viewer @touch-to-scroll></viewer>
      </div>
    </webpdf>
  </template>
</html>
<script>
  var CustomAppearance = UIExtension.appearances.Appearance.extend({
```

```
getLayoutTemplate: function() {
    return document.getElementById('layout-template').innerHTML;
},
getDefaultFragments: function() {
    return [{
        target: 'toolbar',
        action: 'append',
        template: `<xbutton style="margin: 0 10px">appended via fragment configuration</xbutton>`
    }];
},
beforeMounted: function(root) {
    this.toolbarComponent = root.getComponentByName('toolbar')
},
disableAll: function() {
    this.toolbarComponent.disable();
},
enableAll: function() {
    this.toolbarComponent.enable();
}
});
var libPath = window.top.location.origin + '/lib';
var pdfui = new UIExtension.PDFUI({
    viewerOptions: {
        libPath: libPath,
        jr: {
            licenseSN: licenseSN,
            licenseKey: licenseKey
        }
    },
    renderTo: document.body,
    appearance: CustomAppearance,
    addons: []
});
</script>
```



Device Adaptation

If the UI layout needs to be adaptive to the devices, you should determine the device type based on the characteristic value of your current device, and then pass the different appearance instance to PDFUI. Please refer to the following example:

The following code can be used to simulate the operation of different devices using the device mode of Chrome DevTool on the desktop Chrome browser.

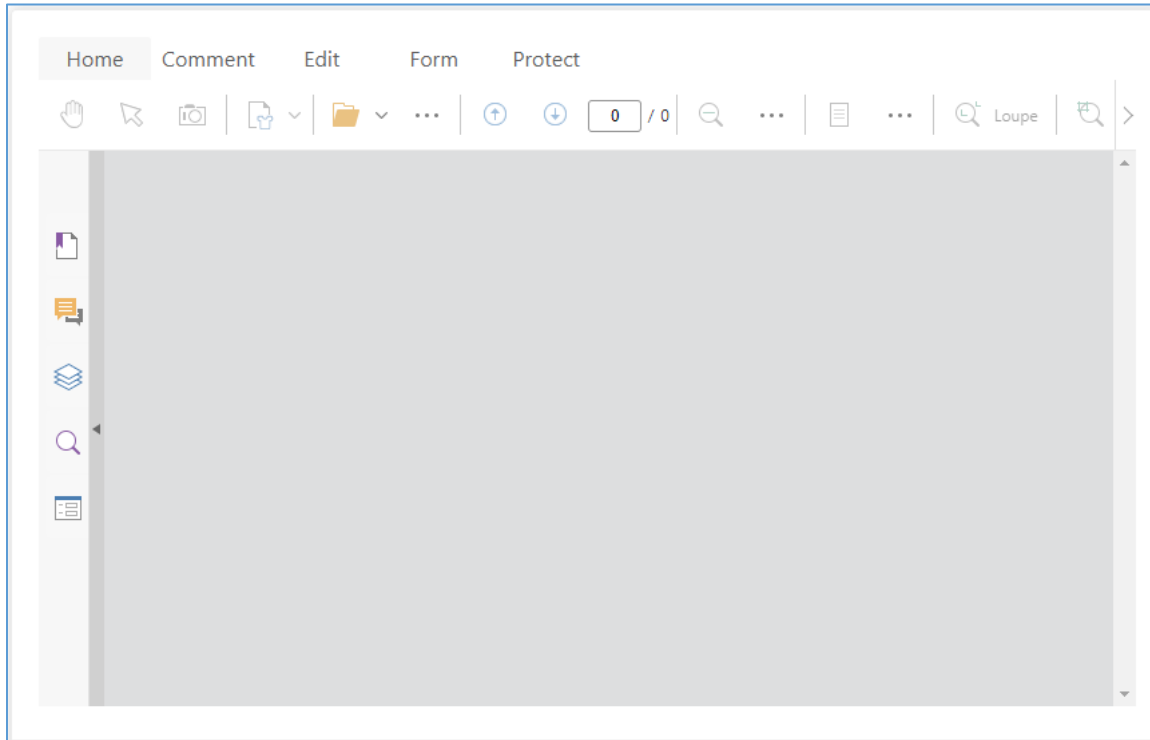
```
<html>
</html>
<script>
  var mobileAppearance = UIExtension.appearances.MobileAppearance;
  var desktopAppearance = UIExtension.appearances.RibbonAppearance;
  var tabletAppearance = UIExtension.appearances.RibbonAppearance;

  var isDesktop = PDFViewCtrl.DeviceInfo.isDesktop;
  var isMobile = PDFViewCtrl.DeviceInfo.isMobile;

  var libPath = window.top.location.origin + '/lib';

  var pdfui = new UIExtension.PDFUI({
    viewerOptions: {
      libPath: libPath,
      jr: {
        licenseSN: licenseSN,
```

```
        licenseKey: licenseKey
    }
},
renderTo: document.body,
// Provide different appearance depending on the device type.
appearance: isDesktop? desktopAppearance : isMobile ? mobileAppearance : tabletAppearance,
addons: []
});
</script>
```



Built-in appearances

```
// desktop appearance
UIExtension.appearances.RibbonAppearance
// mobile appearance
UIExtension.appearances.MobileAppearance
// select ribbon or mobile appearance according to the device type (support both desktop and mobile)
UIExtension.appearances.AdaptiveAppearance
```

Modular

Modules are equivalent to a separate namespace, and UIExtension places all components, controllers and directives in different modules, which can avoid name conflicts. Currently, the modules are used in the following scenarios:

- Root module: The basic components and directives are placed in the root module. Root module does not have module name, and does not need to add module name prefix when using it.
- Business module: Business components and controller.
- The module created by Addon.

Detailed information will be introduced in the related sections of Components.

Create a new module

```
const module = PDFUI.module('module-name', [  
  // ...dependencies  
]);
```

The module name cannot be repeated, otherwise it will report errors.

The second parameter is a dependent module that you can pass a name or module object. If it has no dependent module, you can pass an empty array.

Get module object

11. Get root module object

The root module is the foundation of all modules, and it contains the information of all built-in components and layouts.

```
const root = PDFUI.root();
```

12. Get a custom module object

As with the method of creating module, but it does not have the second parameter. It will report errors when the module name does not exist.

```
const module = PDFUI.module('module-name');
```

The methods of the module object

1. Register new component

```
// Register a custom component.  
module.registerComponent(class ComponentClass extends UIExtension.Component{  
  static getName() {  
    return 'custom-component';  
  }  
});  
// 或者
```



```
module.registerComponent(UIExtension.Component.extend('custom-component', {  
  //  
}));  
module.getComponentClass('custom-component');
```

Use the custom component in the template:

```
<module-name:custom-component></module-name:custom-component>
```

2. Register a pre-configured component

```
module.registerPreConfiguredComponent('pre-configured-btn', {  
  template: '<xbutton name="pre-configured-btn"></xbutton>',  
  config: [{  
    target: 'pre-configured-btn',  
    callback: function() {  
      alert('button click')  
    }  
  ]  
});
```

Use the component in the template:

```
<module-name:pre-configured-btn></module-name:pre-configured-btn>
```

3. Register Controller

```
module.registerController(class CustomController extends Controller {  
  static getName() {  
    return 'CustomController';  
  }  
  handle() {  
    alert("")  
  }  
});
```

Or

```
module.controller('CustomController', {  
  handle: function() {  
    alert("")  
  }  
});
```

Use the controller in the template:

```
<module-name:custom-component @controller="module-name:CustomController"></module-  
name:custom-component>
```

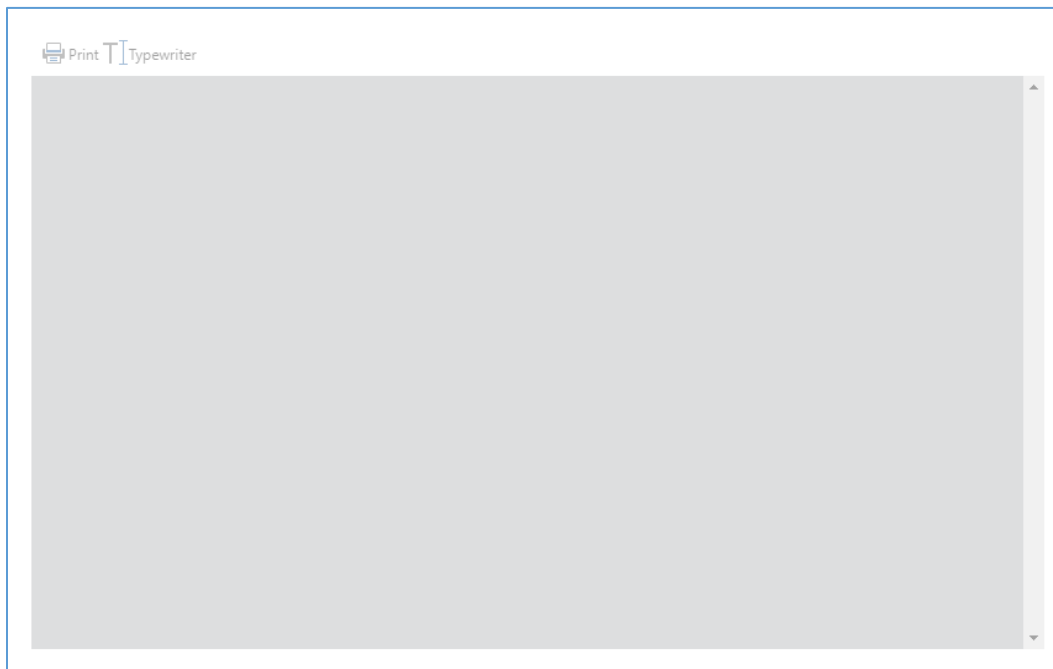
The layout template

Example

In Foxit PDF SDK for Web, templates are written with HTML that contains UIExtension specific elements, attributes and directives. UIExtension combines the layout template with information from the component, controller and directive to render UI in the browser. The following code snippet shows a template with UIExtension components and directives.

```
<webpdf>
  <div class="toolbar" style="display:flex;flex-direction:row;padding:6px">
    <print:print-button></print:print-button>
    <xbutton name="create-typewriter-button" @controller="states:CreateTypewriterController" icon-
class="fv_icon-toolbar-typewriter">toolbar.create.typewriter</xbutton>
  </div>
  <div class="fv_ui-body">
    <viewer @touch-to-scroll></viewer>
  </div>
  <template name="template-container">
    <print:print-dialog></print:print-dialog>
  </template>
</webpdf>
```

Click "run" button to view the running result:



```
<html>
  <template id="layout-template-container">
```

```
<webpdf>
  <div name="mytoolbar" class="toolbar" style="display:flex;flex-direction:row;padding:6px">
    <print:print-button></print:print-button>
    <xbutton name="create-typewriter-button" @controller="states:CreateTypewriterController" icon-
class="fv__icon-toolbar-typewriter">toolbar.create.typewriter</xbutton>
  </div>
  <div class="fv__ui-body">
    <viewer @touch-to-scroll></viewer>
  </div>
  <template name="template-container">
    <print:print-dialog></print:print-dialog>
  </template>
</webpdf>
</template>
</html>
<script>
var CustomAppearance = UIExtension.appearances.Appearance.extend({
  getLayoutTemplate: function() {
    return document.getElementById('layout-template-container').innerHTML;
  },
  beforeMounted: function(root) {
    this.toolbarComponent = root.getComponentByName('mytoolbar')
  },
  disableAll: function() {
    this.toolbarComponent.disable();
  },
  enableAll: function() {
    this.toolbarComponent.enable();
  }
});
var libPath = window.top.location.origin + '/lib';
var pdfui = new UIExtension.PDFUI({
  viewerOptions: {
    libPath: libPath,
    jr: {
      licenseSN: licenseSN,
      licenseKey: licenseKey
    }
  },
  renderTo: document.body,
  appearance: CustomAppearance,
  addons: [
    libPath + '/uix-addons/print'
  ]
});
</script>
```

Description of the format of layout template format

```
<!-- Layout template must take <webpdf> as the root component. -->
<webpdf>
  <!-- Layout templates support all html tags and properties. -->
  <div class="toolbar" style="display:flex;flex-direction:row;padding:6px">
    <!-- The colon prefix is the name of the component module, and the component name and module name
    must be in lowercase. -->
    <print:print-button></print:print-button>

    <!-- The parameters that begin with @ are used to mark directives, the content that follow with @ is the
    name of directives. If the directive is registered in a non-root module, then the directive name should be written
    in the @module-name:directive-name format. -->

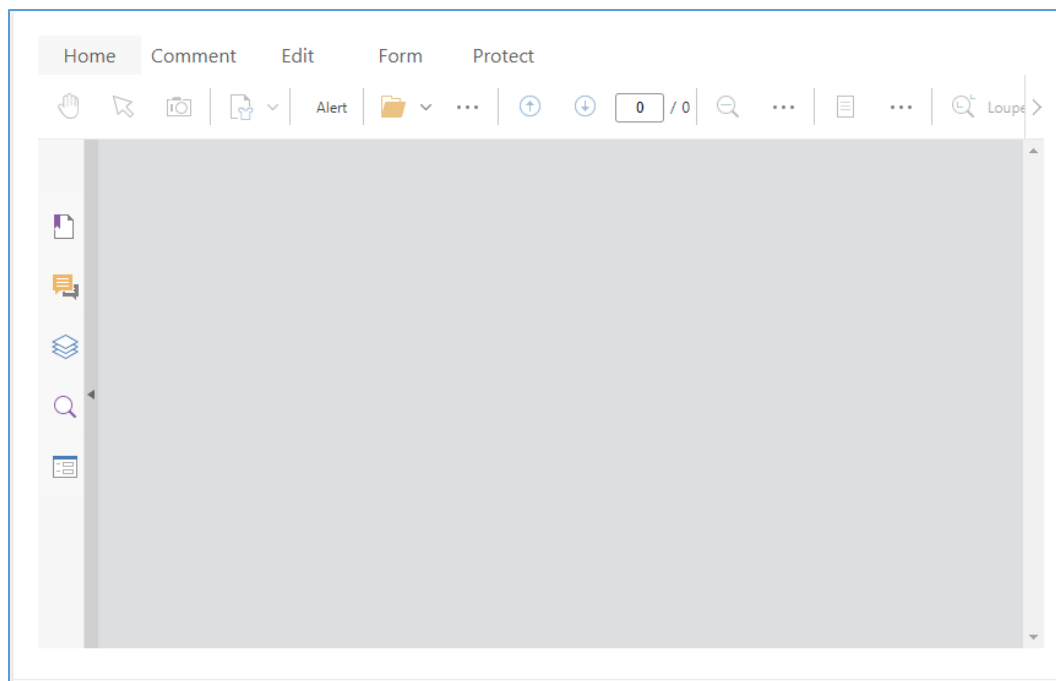
    <xbutton name="create-typewriter-button" @controller="states:CreateTypewriterController" icon-
    class="fv__icon-toolbar-typewriter">toolbar.create.typewriter</xbutton>
  </div>
  <div class="fv__ui-body">
    <!-- Viewer is used to display the area of PDF, so there must be a viewer component in the layout template -
    -->
    <viewer @touch-to-scroll></viewer>
  </div>
  <!-- Template is a re-written html template tag. It is typically used to hold the components that do not need to
  be displayed immediately, such as dialog boxes, right-click menus, floating boxes, and etc.. -->
  <template name="template-container">
    <print:print-dialog></print:print-dialog>
  </template>
</webpdf>
```

How to specify layout templates and implement device adaptation

Please refer to the section [Appearance](#).

Dynamically insert layout templates

Please click "run" button to run the example:



```
<script>
var libPath = window.top.location.origin + '/lib';
var pdfui = new UIExtension.PDFUI({
  viewerOptions: {
    libPath: libPath,
    jr: {
      licenseSN: licenseSN,
      licenseKey: licenseKey
    }
  },
  renderTo: document.body,
  addons: [
    libPath + '/uix-addons/print'
  ]
});
pdfui.getComponentByName('home-tab-group-change-color')
.then(component => {
  // after this component, insert a new group component.
  component.after(`
    <group>
      <xbutton name="alert-btn" class="fv_ui-toolbar-show-text-button">Alert</xbutton>
    </group>
  `, [{
    target: 'alert-btn',
    config: {
      callback: function() {
        alert('Hello world')
      }
    }
  }
  ]
});
```

```
    })  
  }  
</script>
```

The APIs that support inserting templates are as follows:

- Component
 - #after(component|template, fragments)
 - #before(component|template, fragments)
- ContainerComponent
 - #append(component|template, fragments)
 - #prepend(component|template, fragments)
 - #insert(component|template, index, fragments)

For more information about these APIs, please refer to the API Reference: [Component](#) and [ContainerComponent](#).

Insert the layout template when initializing

Please refer to [UI Fragments](#).

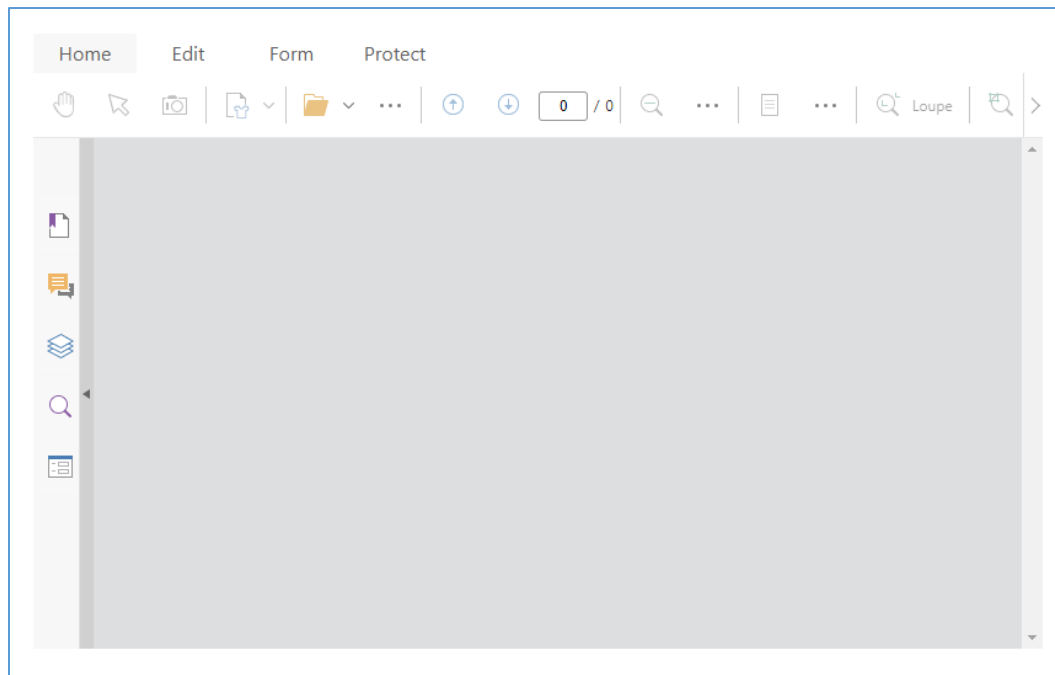
UI fragments

Fragments are a set of UI snippets, which can be used to insert, delete, or modify the components in UI template. It is suitable to facilitate a small amount of UI customization based on built-in templates.

If you need a lot of custom layout and device adaptation, please refer to the methods described in [Appearance](#) and [layout template](#).

simple example

The following code will use fragments configuration to remove the `comment-tab` component from the mobile and desktop/tablet layouts. Click "run" to run the example, and you can use the device mode of Chrome DevTool to simulate the running effect of mobile/tablet.



```
<script>
var CustomAppearance= UIExtension.appearances.AdaptiveAppearance.extend({
  getDefaultFragments: function() {
    var isMobile = PDFViewCtrl.DeviceInfo.isMobile;
    if(isMobile) {
      // Fragment configuration for mobile devices.
      return [{
        target: 'comment-tab',
        action: 'remove'
      }, {
        target: 'comment-tab-li',
        action: 'remove'
      }, {
        target: 'comment-tab-body',
        action: 'remove'
      }
    ];
    } else {
      // Fragment configuration for desktop/tablet devices.
      return [{
        target: 'comment-tab',
        action: 'remove'
      }, {
        target: 'fv--comment-tab-paddle',
        action: 'remove'
      }, {
        target: 'hand-tool',
        config: {
          callback: {
            around: function(callback, args) {
```

```
        try{
            console.info('before callback');
            var ret;
            if(callback instanceof UIExtension.Controller) {
                ret = callback.handle(...args);
            } else {
                ret = callback.apply(this, args);
            }
            console.info('after callback');
            return ret;
        }catch(e) {
            console.error(e, 'an error occurred');
        } finally {
            console.info("");
        }
    }
}
},
});
}
}
});
var libPath = window.top.location.origin + '/lib';

var pdfui = new UIExtension.PDFUI({
    viewerOptions: {
        libPath: libPath,
        jr: {
            licenseSN: licenseSN,
            licenseKey: licenseKey
        }
    },
    renderTo: document.body,
    // Different appearances are available depending on the device type.
    appearance: CustomAppearance,
    addons: []
});
</script>
```

The description of the Fragment configuration parameters

- target: The name of the control, and each name is unique.
- action: Indicates the action mode of the fragment snippets. The default action mode is UIExtension.UIConsts.FRAGMENT_ACTION.EXT. The specifics are as follows:
 - UIExtension.UIConsts.FRAGMENT_ACTION.EXT: Extend the target control.
 - UIExtension.UIConsts.FRAGMENT_ACTION.BEFORE: Insert a new control before the target control.

- UIExtension.UIConsts.FRAGMENT_ACTION.AFTER: Insert a new control after the target control.
- UIExtension.UIConsts.FRAGMENT_ACTION.APPEND: Insert a new control into the target control (the target control must be a container).
- UIExtension.UIConsts.FRAGMENT_ACTION.FILL: Empty the child space of the target control and fill with a new control. Make sure that the target control must be a container.
- UIExtension.UIConsts.FRAGMENT_ACTION.REPLACE: Replace the target control with a new control.
- UIExtension.UIConsts.FRAGMENT_ACTION.REMOVE: Delete the target control.
- **template**: The template of the control. The content is in XML format and action is BEFORE/AFTER/APPEND/FILL/REPLACE.
- **config**: Control configuration object. It is invalid when action is REMOVE.
 - **config.target**: The name of the control in the above template. It is only required when action is BEFORE/AFTER/APPEND/FILL/REPLACE.
 - **config.attrs**: Set the html property of the control.
 - **config.callback**: The business logic implementation of the control. There are three ways to implement it:
 - **function**: The events of control will call this function, and override the built-in callbacks. The basic components that support function are (xbutton, dropdown-button, context-menu-item). If you want to add functionalities based on the built-in callbacks, you can use the second method.
 - **controller class**: Controller class can listen for components lifecycle and handle more component events:

```
{
  target: 'hand-tool',
  config: {
    callback: class extends UIExtension.Controller {
      mounted() {
        super.mounted();
        this.component.element.addEventListener('hover', e => {
          console.info('mouse over', this.component)
        })
      }
      handle() {
        console.info('hand-tool clicked')
      }
    }
  }
}
```

- **decorator object**: it contains a series of function hooks for blocking the execution of the controller handle method, including before, after, thrown, and around.

```
{
  target: 'hand-tool',
  config: {
    callback: {
      before: function() {
        // The function executed before calling the handle method of controller. It can receive all
        parameters of the handle method.
      },
      after: function(returnValue) {
        // The function executed after calling the handle method of controller. It can receive the
        return value and parameters of the handle function.
      },
      thrown: function(error) {
        // The function executed when the handle method of controller throws an exception. It
        can receive the exception object and parameters.
      },
      around: function(callback, args) {
        // It can receive the references and parameters of controller's handle method. Inside the
        around callback, you can execute code before/after running the handle function, or in the catch
        exception block. It also can decide whether to execute the handle method.

        try{
          console.info('before callback');
          var ret;
          if(callback instanceof UIExtension.Controller) {
            ret = callback.handle(...args);
          } else {
            ret = callback.apply(this, args);
          }
          console.info('after callback');
          return ret;
        }catch(e) {
          console.error(e, 'an error occurred');
        } finally {
          console.info("");
        }
      }
    }
  }
}
```

Note

It is recommended that only use fragment for UI fine-tuning. If you want to substantially modify the built-in layout, please refer to the methods described in [Appearance](#) and [layout template](#).

Component selector

UIExtension provides a css-selector like syntax to make easier to search components. It's usually used to configure the target property of [fragments](#) and component search.

Syntax

selector name	example	description
name selector	'componentName', 'component_name','component-name', '1component'	component name selectors can only include single-letter, number, underscore or minus character
type selector	'@div','@dropdown-menu', '@print:print-dialog'	component type means the tag name defined in layout template, a type selector should start with @ character and single-letter, number, underscore or minus. Sometime including the component module name separated with colon character.
star selector	'*'	Selects all components
children selector	'selector1>selector2'	Selects all components which match <code>selector2</code> where the parent is <code>selector1</code>
descendants	'selector1 selector2'	Selects all <code>selector2</code> components inside <code>selector1</code>
attribute selector	[attr=value]	Selects all components with property or attribute name of <code>attr</code> whose value equals to <code>value</code>
attribute selector	[attr^=value]	Selects all components with property or attribute name of <code>attr</code> whose value begins with <code>value</code>
attribute selector	[attr\$=value]	Selects all components with property or attribute name of <code>attr</code> whose value ends with <code>value</code>
attribute selector	[attr*=value]	Selects all components with property or attribute name of <code>attr</code> whose value contains with <code>value</code>

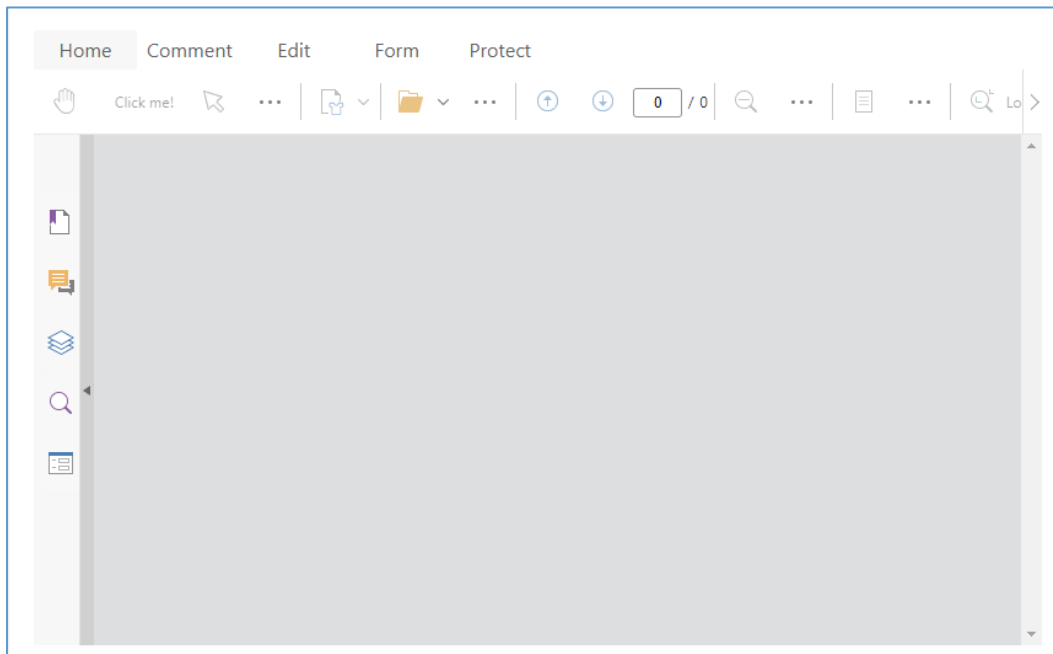
selector name	example	description
attribute selector	[attr!=value]	Selects all components with property or attribute name of <code>attr</code> whose value not equals to <code>value</code>
method selector	selector1::childAt(index)	Selects all components that are all the child at <code>index</code> of their parents selected by <code>selector1</code>
method selector	selector1::parent()	Selects all components that are all the parent component of their children selected by <code>selector1</code>
method selector	selector1::allAfter()	Selects all components of the same level that after the component set selected by <code>selector1</code>
method selector	selector1::allBefore()	Selects all components of the same level that before the component set selected by <code>selector1</code>
index-related selector	selector1::eq(index)	Selects the component by index value in components set selected by <code>selector1</code>
index-related selector	selector1::last()	Selects the last one component of the components set selected by <code>selector1</code>
index-related selector	selector1::first()	Selects the first one component of the components set selected by <code>selector1</code> , It's equivalent to <code>selector1:eq(0)</code>

Examples

```
<html>
</html>
<script>
  UIExtension.PDFUI.module('custom',[])
    .controller('customController', {
      handle: function() {
        const root = this.component.getRoot();
        const contextmenultems = root.querySelectorAll('fv--page-contextmenu>@contextmenu-item');
        contextmenultems.forEach(function(contextmenu) {
          contextmenu.element.style.cssText += 'color: red';
        })
      }
    })
  })
})
```

```
var CustomRibbonAppearance = UIExtension.appearances.RibbonAppearance.extend({
  getDefaultFragments() {
    // remove the export comment dropdown menu!
    return [{
      target: 'home-tab-group-hand::childAt(0)',
      action: 'after',
      template: `<xbutton class="fv__ui-toolbar-show-text-button">Click me!</xbutton>`
    },{
      target: 'commentlist-export-comment::parent()',
      action: 'remove'
    }];
  }
});

var libPath = window.top.location.origin + '/lib';
var pdfui = new UIExtension.PDFUI({
  viewerOptions: {
    libPath: libPath,
    jr: {
      licenseSN: licenseSN,
      licenseKey: licenseKey
    }
  },
  renderTo: document.body,
  appearance: CustomRibbonAppearance,
  addons: []
});
</script>
```



Icon

For icons information, please refer to the guide in the [website](#).

I18n

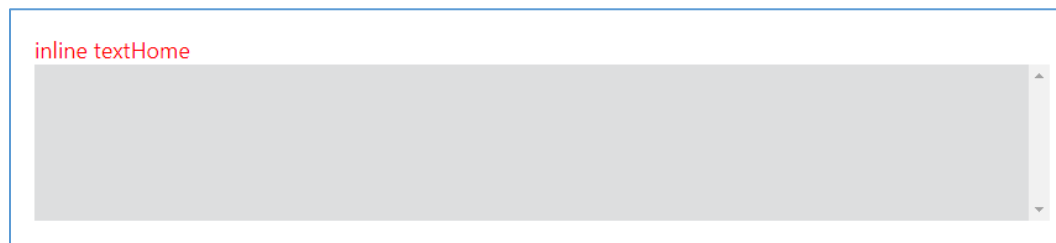
Custom resources

Please refer to this [page](#).

Usage

`<text>` component

`<text>` is a component used to display text. It supports i18n entries. On the DOM tree, it does not create a new HTML Element, but a text node and inserts it into the DOM tree. The font style needs to be enclosed outside. Other tags are set through CSS.

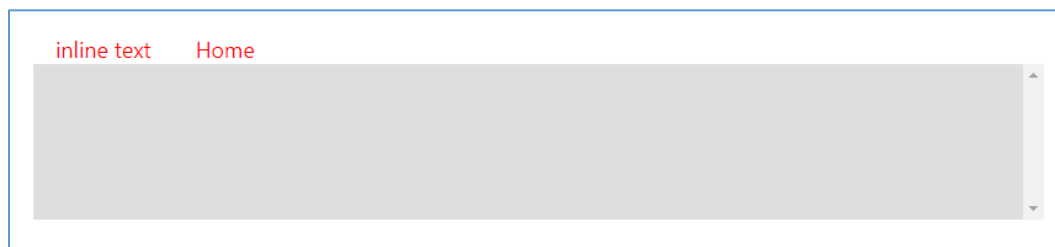


```
<html>
  <template id="layout-template">
    <webpdf>
      <div>
        <span class="span-with-text-component">
          <!-- The text "inline text" will be displayed -->
          <text>inline text</text>
        </span>
        <span class="span-with-text-component">
          <!-- The text "Home" will be displayed -->
          <text>toolbar.tabs.home.title</text>
        </span>
      </div>
      <div class="fv__ui-body">
        <viewer></viewer>
      </div>
    </webpdf>
  </template>
</html>
<style>
```

```
.span-with-text-component {
  color: red;
  font-size: 18px;
  font-style: bold;
}
</style>
<script>
var CustomAppearance = UIExtension.appearances.Appearance.extend({
  getLayoutTemplate: function() {
    return document.getElementById('layout-template').innerHTML;
  },
  disableAll: function(){}
});
var libPath = window.top.location.origin + '/lib';
var pdfui = new UIExtension.PDFUI({
  viewerOptions: {
    libPath: libPath,
    jr: {
      licenseSN: licenseSN,
      licenseKey: licenseKey
    }
  },
  renderTo: document.body,
  appearance: CustomAppearance,
  addons: []
});
</script>
```

data-i18n attribute

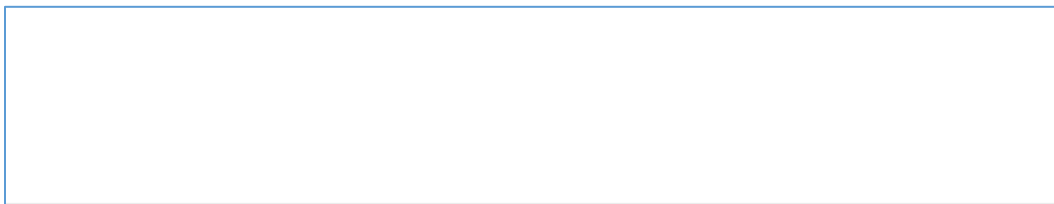
data-i18n attribute is another way to show texts in a HTML element, difference from `<text>` component, `data-i18n` will cover all children and replace to the text.



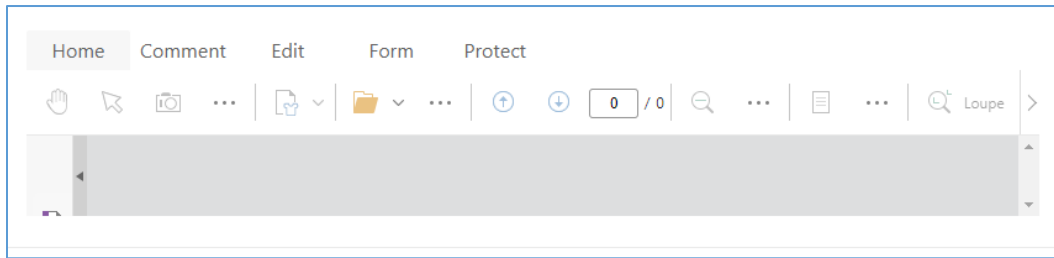
```
<html>
  <template id="layout-template">
    <webpdf>
      <div>
        <!-- The text "inline text" will be displayed -->
        <span class="span-with-text-component" data-i18n="inline text"> </span>
        <!-- The text "Home" will be displayed -->
        <span class="span-with-text-component" data-i18n="toolbar.tabs.home.title"> </span>
      </div>
```

```
<div class="fv__ui-body">
  <viewer></viewer>
</div>
</webpdf>
</template>
</html>
<style>
  .span-with-text-component {
    color: red;
    font-size: 18px;
    font-style: bold;
    padding: 0 1em;
  }
</style>
<script>
var CustomAppearance = UIExtension.appearances.Appearance.extend({
  getLayoutTemplate: function() {
    return document.getElementById("layout-template").innerHTML;
  },
  disableAll: function(){}
});
var libPath = window.top.location.origin + '/lib';
var pdfui = new UIExtension.PDFUI({
  viewerOptions: {
    libPath: libPath,
    jr: {
      licenseSN: licenseSN,
      licenseKey: licenseKey
    }
  },
  renderTo: document.body,
  appearance: CustomAppearance,
  addons: []
});
</script>
```

components supporting



Switch current language via API



```
<html></html>
<style>
  .span-with-text-component {
    color: red;
    font-size: 18px;
    font-style: bold;
    padding: 0 1em;
  }
</style>
<script>
  UIExtension.PDFUI.module('custom', [])
  .controller('SwitchLanguageController', {
    mounted: function() {
      this.updateButtonText();
    },
    updateButtonText: function() {
      const pdfui = this.getPDFUI();
      switch(pdfui.currentLanguage || navigator.language) {
        case 'en':
        case 'en-US':
          this.component.setText('Swith to Chinese');
          break;
        case 'zh':
        case 'zh-CN':
          this.component.setText('切换为英文');
          break;
      }
    },
    handle: function() {
      const pdfui = this.getPDFUI();
      switch(pdfui.currentLanguage) {
        case 'en':
        case 'en-US':
          pdfui.changeLanguage('zh-CN').then(() => {
            this.updateButtonText();
          });
          break;
        case 'zh':
        case 'zh-CN':
          pdfui.changeLanguage('en-US').then(() => {
            this.updateButtonText();
          });
      }
    }
  });
</script>
```

```
        break;
    }
}
});
var CustomAppearance = UIExtension.appearances.AdaptiveAppearance.extend({
    getDefaultFragments: function() {
        return [{
            target: 'home-tab-group-hand',
            action: 'append',
            template: '<xbutton class="fv__ui-toolbar-show-text-button"
@controller="custom:SwitchLanguageController"></xbutton>'
        }];
    }
});
var libPath = window.top.location.origin + '/lib';
var pdfui = new UIExtension.PDFUI({
    viewerOptions: {
        libPath: libPath,
        jr: {
            licenseSN: licenseSN,
            licenseKey: licenseKey
        }
    },
    renderTo: document.body,
    appearance: CustomAppearance,
    addons: []
});
</script>
```

Components

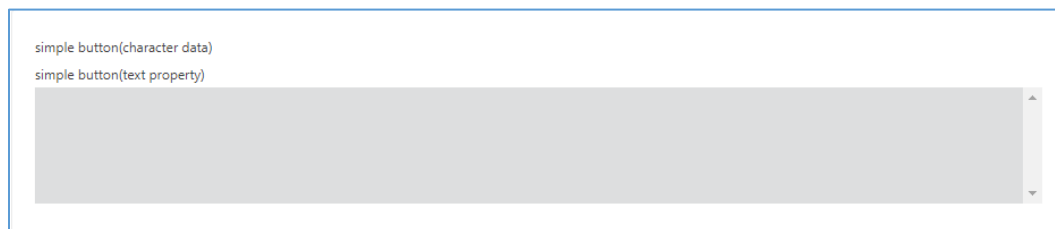
Basic Components

XButton component

Xbutton is the UIExtension button component. It can be used to customize icon, define whether to show text, whether to disable/enable button, etc.

Code examples

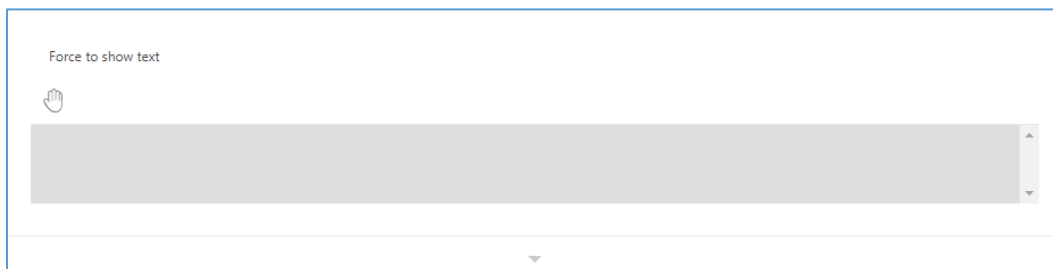
7. Simple xbutton example:



```
<html>
  <template id="layout-template">
    <webpdf>
      <div>
        <xbutton>simple button(character data)</xbutton>
        <xbutton text="simple button(text property)"></xbutton>
      </div>
      <div class="fv__ui-body">
        <viewer></viewer>
      </div>
    </webpdf>
  </template>
</html>
<script>
var CustomAppearance = UIExtension.appearances.Appearance.extend({
  getLayoutTemplate: function() {
    return document.getElementById('layout-template').innerHTML;
  },
  disableAll: function(){}
});
var libPath = window.top.location.origin + '/lib';
var pdfui = new UIExtension.PDFUI({
  viewerOptions: {
```

```
libPath: libPath,  
jr: {  
  licenseSN: licenseSN,  
  licenseKey: licenseKey  
},  
renderTo: document.body,  
appearance: CustomAppearance,  
addons: []  
});  
</script>
```

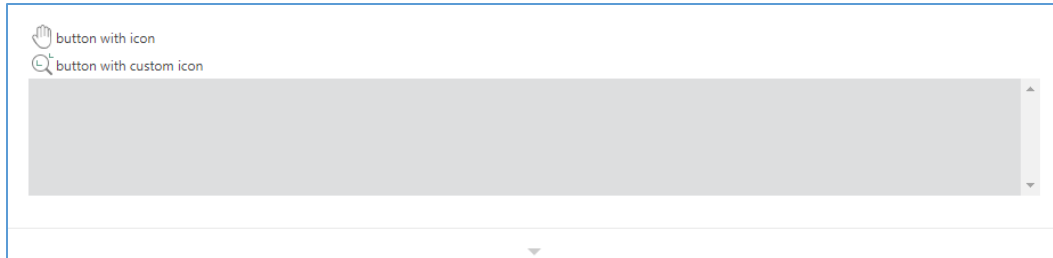
8. Force to show text in built-in toolbar component



```
<html>  
  <template id="layout-template">  
    <webpdf>  
      <toolbar>  
        <xbutton class="fv_ui-toolbar-show-text-button">Force to show text</xbutton>  
        <xbutton icon-class="fv_icon-toolbar-hand">Text will be hidden</xbutton>  
      </toolbar>  
      <viewer></viewer>  
    </webpdf>  
  </template>  
</html>  
<script>  
  var CustomAppearance = UIExtension.appearances.Appearance.extend({  
    getLayoutTemplate: function() {  
      return document.getElementById('layout-template').innerHTML;  
    },  
    disableAll: function(){}  
  });  
  
  var libPath = window.top.location.origin + '/lib';  
  var pdfui = new UIExtension.PDFUI({  
    viewerOptions: {  
      libPath: libPath,  
      jr: {  
        licenseSN: licenseSN,  
        licenseKey: licenseKey  
      }  
    }  
  });
```

```
    },  
    renderTo: document.body,  
    appearance: CustomAppearance,  
    addons: []  
  });  
</script>
```

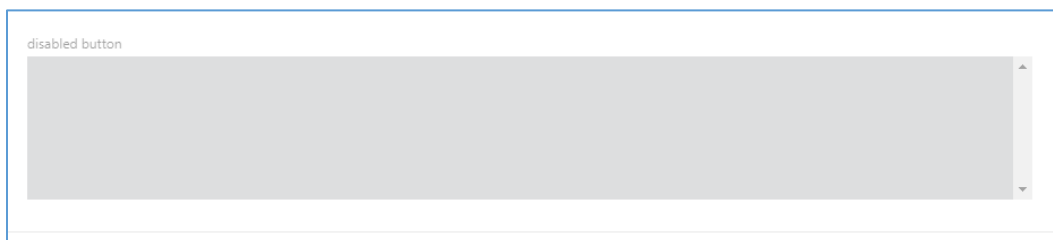
9. Customize icon-class



```
<html>  
  <template id="layout-template">  
    <webpdf>  
      <div>  
        <xbutton icon-class="fv__icon-toolbar-hand">button with icon</xbutton>  
        <xbutton icon-class="custom-icon-css-class">button with custom icon</xbutton>  
      </div>  
    <viewer></viewer>  
  </webpdf>  
</template>  
</html>  
<script>  
  var CustomAppearance = UIExtension.appearances.Appearance.extend({  
    getLayoutTemplate: function() {  
      return document.getElementById('layout-template').innerHTML;  
    },  
    disableAll: function(){}  
  });  
  var libPath = window.top.location.origin + '/lib';  
  var pdfui = new UIExtension.PDFUI({  
    viewerOptions: {  
      libPath: libPath,  
      jr: {  
        licenseSN: licenseSN,  
        licenseKey: licenseKey  
      }  
    },  
    renderTo: document.body,  
    appearance: CustomAppearance,  
    addons: []  
  });  
</script>
```

```
<style>
  .custom-icon-css-class {
    background-repeat: no-repeat;
    background-position: center;
    background-image: url(data:image/png;charset=utf-
8;base64,iVBORw0KGgoAAAANSUHEUgAAABgAAAAAYCAAAADgdz34AAAAAGXRFWHRTb2Z0d2FyZQBBZG9iZSBJb
WFnZVJlYWRS5cclIPAAAA3ZpVFh0WE1MOmNvbS5hZG9iZS54bXAAAAAADw/eHBhY2tldCBiZWdpbj0i77u/liBpZD0
iVzVNME1wQ2VoaUh6cmVTek5UY3prYzlkIj8+IDx4OnhtcG1ldGEgeG1sbnM6eD0iYWVrYmU6bnM6bWV0YS8iIHg
6eG1wdGs9IkFkb2JlIFhNUCBDb3JlIDUuNi1jMTQyIjE2MDkyNCwgMjAxNy8wNy8xMy0wMTowNjozOSAgICAgI
CAgIj4gPHJkZjZpSREYgeG1sbnM6cmRmPSJodHRwOi8vd3d3LnczLm9yZy8xOTk5LzAyLzlyLXJkZi1zeW50YXgtbnMjIj
4gPHJkZjZpEZXNjcmldGlvbiByZGY6YWJvdXQ9IilgeG1sbnM6eG1wTU09Imh0dHA6Ly9ucy5hZG9iZS5jb20veGFwLz
EuMC9tbS8iIHhtbG5zOnN0UmVmPSJodHRwOi8vbnMuYWRvYmUuY29tL3hhcC8xLjAvc1R5cGUvUmVzb3V5Y2VSZ
WYjIiB4bWxuczp4bXA9Imh0dHA6Ly9ucy5hZG9iZS5jb20veGFwLzEuMC8iIHhtcE1NOk9yaWdpbmFsRG9jdW1lbnR
JRD0ieG1wLmRpdZDplMzAwMTU1Yi04ODI1LTlWNDItYTIwNy0yNmQwZTVhNmJhMTUilHhtcE1NOkRvY3VtZW50SU
Q9InhtcC5kaWQ6Qjk0NjgyREYyM0E4MTFFOTgxREFDQTNEMjBCNDM5NTgilHhtcE1NOkluc3RhbmNlSUQ9InhtcC5
paWQ6Qjk0NjgyREYyM0E4MTFFOTgxREFDQTNEMjBCNDM5NTgilHhtcDpDcmVhdG9yVG9vbD0iQWRvYmUgUGh
vdG9zaG9wIENDIDlwMTggKfDpbmRvd3Mplj4gPHhtcE1NOkRlcmllZWRGcm9tIHNOUmVmOmduc3RhbmNlSUQ9I
nhtcC5paWQ6ZWlyZTI2YTI2MTZlMTg1NTU0OTJmNmEyNGEyMDg1liBzdFJlZjZpEZXNjcmldGlvbi4gPC9yZG
Y6UkRGPIA8L3g6eG1wbWV0YT4gPD94cGFja2V0IGVuZD0icil/PjS81+AAAAHoSURBVHjavFa9SwJhGH8vWjRc+kLFp
RIKUoMMKrcKobGprHBpa6m9RRz6AxqiqSmij6WtQaHRK7ioPIOmJjGxoiXOLfs913u CZ3f3StoDP54773me393z9Sr
VajXWSeIHZa2EWwe79W41mFlt5VDOp0m8hVgFzGBBBoBX4Bo4Ac5SqdsXE7H0Ww0QfBjqPBQKRcPhMasEAsztdj
NN01ixWGSqqrjCoXALm2WQPBTfclC+LTkS8OByPB4fjMVilm+Wy+VYNput4HKWSlwUGWKQNRDwtNwg+JRdcBOJ
gstpq3SZi5xASoSck5Ad2ZOfaBcll5FI/Wbn4tCRhNsnRQmm/H5//eZD+3Qk8Hq9pCZECfqpW1oRI8tFqk+U4I1asRW
pVquk3q2emwdNKZVKi8Fg8lddkhqmsrfHw3aXNhocyUyqQd0YFNwdFYTwVE+n68T7K9tOX4B7HU/0RSdYkoVW
ZaF0kN2sL8nPyECPijTCZTcSKh57DT62xXZKtdNELLzNhFPp9P30VUUNSI9hC9OeXGAwwBKrDAI2FDDSSrA8e0Ta
PUwrbFJ4Sag3BFTAKPALzQEWlwGaNm3/ycZlxMwkRtOPAeeFBn4Bx4LITJxqRzAF3gGY3aH8Rmrjlfz/0pU7/bfkWY
ACxTcQvcW9G6AAAAABJRUSErkjggg=);
  }
</style>
```

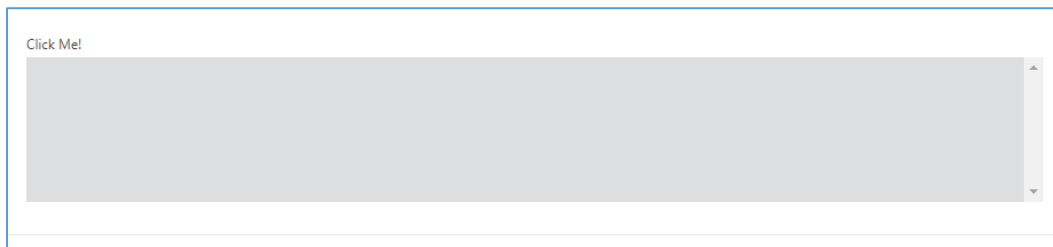
10. Disable button



```
<html>
  <template id="layout-template">
    <webpdf>
      <div>
        <xbutton disabled="true">disabled button</xbutton>
      </div>
    </webpdf>
  </template>
</html>
```

```
<script>
var CustomAppearance = UIExtension.appearances.Appearance.extend({
  getLayoutTemplate: function() {
    return document.getElementById('layout-template').innerHTML;
  },
  disableAll: function(){}
});
var libPath = window.top.location.origin + '/lib';
var pdfui = new UIExtension.PDFUI({
  viewerOptions: {
    libPath: libPath,
    jr: {
      licenseSN: licenseSN,
      licenseKey: licenseKey
    }
  },
  renderTo: document.body,
  appearance: CustomAppearance,
  addons: []
});
</script>
```

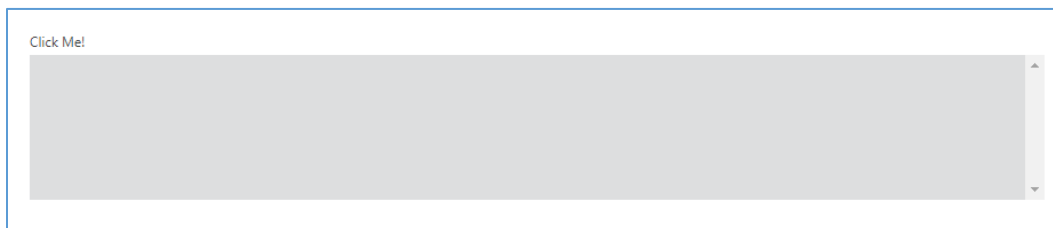
11. Click event handler



```
<html>
<template id="layout-template">
  <webpdf>
    <div>
      <xbutton name="alert-btn">Click Me!</xbutton>
    </div>
  </webpdf>
</template>
</html>
<script>
var CustomAppearance = UIExtension.appearances.Appearance.extend({
  getLayoutTemplate: function() {
    return document.getElementById('layout-template').innerHTML;
  },
  getDefaultFragments: function() {
    return [{
      target: 'alert-btn',
      config: {
```

```
        callback: function() {
            alert('click button!');
        }
    }
    });
},
disableAll: function(){}
});
var libPath = window.top.location.origin + '/lib';
var pdfui = new UIExtension.PDFUI({
    viewerOptions: {
        libPath: libPath,
        jr: {
            licenseSN: licenseSN,
            licenseKey: licenseKey
        }
    },
    renderTo: document.body,
    appearance: CustomAppearance,
    addons: []
});
</script>
```

12. Use controller to handle click event

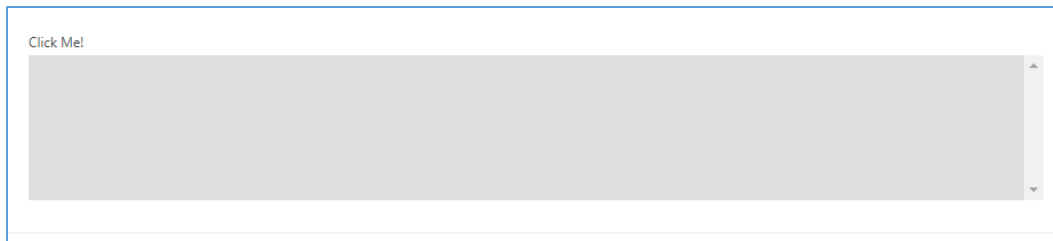


```
<html>
  <template id="layout-template">
    <webpdf>
      <div>
        <xbutton name="alert-btn">Click Me!</xbutton>
      </div>
    </webpdf>
  </template>
</html>
<script>
  var CustomAppearance = UIExtension.appearances.Appearance.extend({
    getLayoutTemplate: function() {
      return document.getElementById('layout-template').innerHTML;
    },
    getDefaultFragments: function() {
      return [{
        target: 'alert-btn',
```



```
    config: {
      callback: UIExtension.controllers.Controller.extend({
        handle: function() {
          alert("Click button!");
        }
      })
    }
  });
},
disableAll: function(){}
});
var libPath = window.top.location.origin + '/lib';
var pdfui = new UIExtension.PDFUI({
  viewerOptions: {
    libPath: libPath,
    jr: {
      licenseSN: licenseSN,
      licenseKey: licenseKey
    }
  },
  renderTo: document.body,
  appearance: CustomAppearance,
  addons: []
});
</script>
```

13. Use controller directive



```
<html>
  <template id="layout-template">
    <webpdf>
      <div>
        <xbutton name="alert-btn" @controller="custom-module:ClickButtonController">Click Me!</xbutton>
      </div>
    </webpdf>
  </template>
</html>
<script>
  var module = UIExtension.PDFUI.module('custom-module', []);
  module.controller('ClickButtonController', {
    handle: function() {
      alert("Click button!");
    }
  });
</script>
```

```

    }
  });

  var CustomAppearance = UIExtension.appearances.Appearance.extend({
    getLayoutTemplate: function() {
      return document.getElementById('layout-template').innerHTML;
    },
    disableAll: function(){}
  });

  var libPath = window.top.location.origin + '/lib';
  var pdfui = new UIExtension.PDFUI({
    viewerOptions: {
      libPath: libPath,
      jr: {
        licenseSN: licenseSN,
        licenseKey: licenseKey
      }
    },
    renderTo: document.body,
    appearance: CustomAppearance,
    addons: []
  });
</script>

```

API

Xbutton object properties

Properties	Description	Type
disabled	Button disabled status	boolean
isVisible	Button visibility status	boolean

Methods

Method	Description	Version
setText(text: String): void	Set button text. It supports l18n entry.	7.0
setIconCls(cssClass: String): void	Set icon's css-class of a button	7.0
disable(): void	Disable button. The disabled icon will not respond to the click event	7.0

Method	Description	Version
enable(): void	Enable button. The enabled button will respond to the click event	7.0
show(): void	Show the hidden button	7.0
hide(): void	Hide the button	7.0
destroy(): void	Destroy the button component	7.0

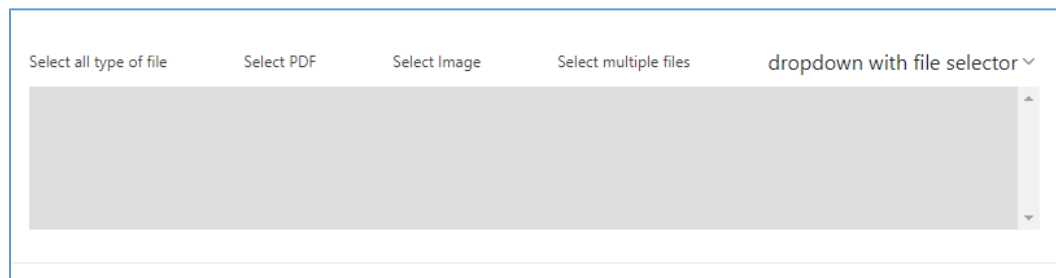
Events

Name	Description	Sample	Version
click	Click button to trigger	button.on('click', () => {})	7.0

File selector

The usage of File selector is almost same as [button](#). It inherits from the XbuttonComponent and supports the [accept](#) property and the [change](#) event.

Code example



```
<html>
  <template id="layout-template">
    <webpdf>
      <div class="file-selector-container">
        <!-- accepts all type of files -->
        <file-selector accept="*.*)">Select all type of file</file-selector>
        <!-- accepts PDF files -->
        <file-selector accept=".pdf">Select PDF</file-selector>
        <!-- accepts image files -->
        <file-selector accept=".png;.jpg;.bmp" @controller="custom:SelectSingleFileController">Select
Image</file-selector>
        <!-- select multiple files -->
```

```

        <file-selector @controller="custom:SelectMultipleFileController" accept="image/*" multiple>Select
multiple files</file-selector>
        <!-- use in dropdown -->
        <dropdown style="width: auto" text="dropdown with file selector" separate="false">
            <file-selector accept=".xpdf;.pdf" text="import PDF/XPDF" icon-class="fv__icon-sidebar-import-
comment"></file-selector>
        </dropdown>
    </div>
    <div class="fv__ui-body">
        <viewer></viewer>
    </div>
</webpdf>
</template>
</html>
<style>
    .file-selector-container {
        display: flex;
        flex-wrap: wrap;
    }
    .file-selector-container > .fv__ui-fileselector {
        flex: 1 1 auto;
    }
</style>
<script>
    UIExtension.PDFUI.module('custom', [])
        .controller('SelectSingleFileController', {
            handle: function(file) {
                alert('Selected file: ' + file.name);
            }
        })
        .controller('SelectMultipleFileController', {
            handle: function(files) {
                alert('Selected files: \r\n' + files.map(it => it.name));
            }
        })
    var CustomAppearance = UIExtension.appearances.Appearance.extend({
        getLayoutTemplate: function() {
            return document.getElementById('layout-template').innerHTML;
        },
        disableAll: function(){}
    });
    var libPath = window.top.location.origin + '/lib';
    var pdfui = new UIExtension.PDFUI({
        viewerOptions: {
            libPath: libPath,
            jr: {
                licenseSN: licenseSN,
                licenseKey: licenseKey
            }
        },
        renderTo: document.body,

```

```
appearance: CustomAppearance,  
addons: []  
});  
</script>
```

API

You may check [button](#) for more details.

Events

Name	Description	Example	Version
change	Triggered when the button is clicked. If the file selector turns on multiple selection, file is an array, otherwise it is a single file instance	<code>fileSelector.on('change', (file) => { if(Array.isArray(file)) {} else {} })</code>	7.4

Dropdown component

Code examples

Basic example



```
<html>  
  <template id="layout-template">  
    <webpdf>  
      <div>
```

```

        <dropdown icon-class="fv__icon-toolbar-shape" text="Dropdown">
            <xbutton icon-class="fv__icon-toolbar-square">Square</xbutton>
            <xbutton icon-class="fv__icon-toolbar-circle">Circle</xbutton>
            <li class="fv__ui-dropdown-separator"></li>
            <file-selector>Select a file</file-selector>
            <li class="my-dropdown-list-item">
                </li>
            </dropdown>
        </div>
        <div class="fv__ui-body">
            <viewer></viewer>
        </div>
    </webpdf>
</template>
</html>
<style>
    .my-dropdown-list-item {
        padding: 10px 0;
        text-align: center;
    }
    .fv__ui-dropdown {
        width: auto;
    }
</style>
<script>
    var CustomAppearance = UIExtension.appearances.Appearance.extend({
        getLayoutTemplate: function() {
            return document.getElementById('layout-template').innerHTML;
        },
        disableAll: function(){}
    });
    var libPath = window.top.location.origin + '/lib';
    var pdfui = new UIExtension.PDFUI({
        viewerOptions: {
            libPath: libPath,
            jr: {
                licenseSN: licenseSN,
                licenseKey: licenseKey
            }
        },
        renderTo: document.body,
        appearance: CustomAppearance,
        addons: []
    });
</script>

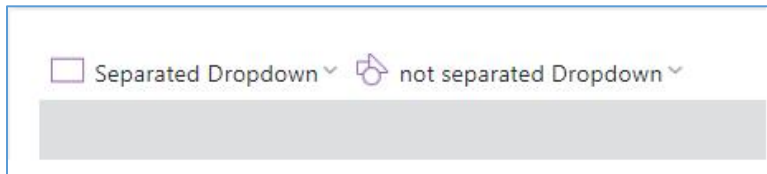
```

Separation

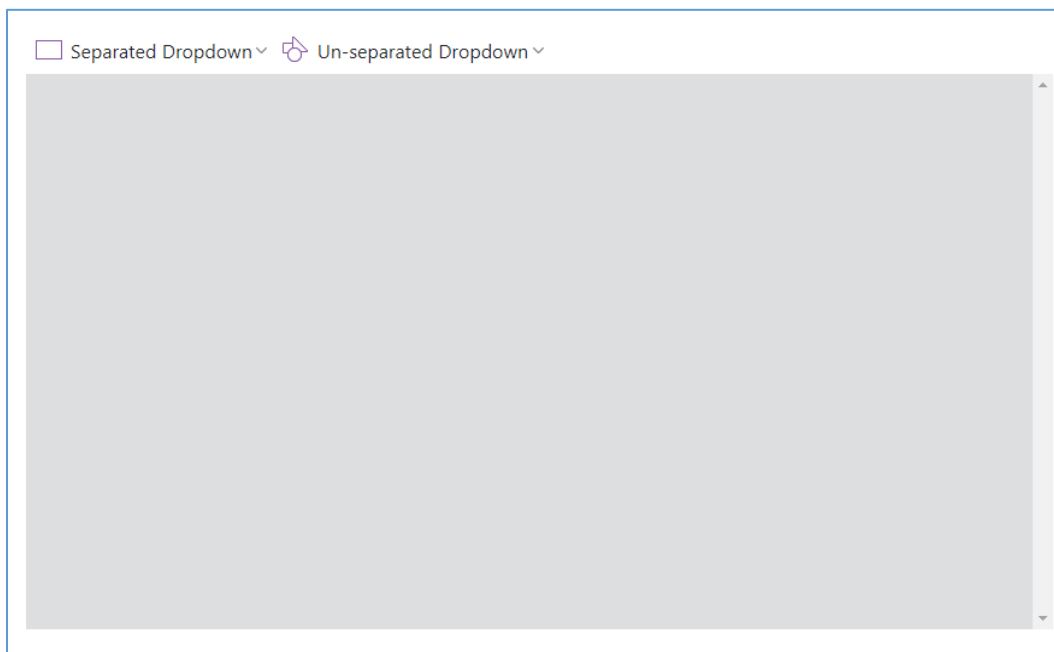
A dropdown button can be divided into left and right parts. The left part consists of icons and text, and the right part is a drop-down arrow. When the **separate** parameter is set to false, you can click

any one of the two parts to display the drop-down list. When the **separate** parameter is set to true, you can only click the right part (drop-down arrow) to display the drop-down list.

In the following demo, you will see two dropdown buttons as shown below:



Try to click the 'Separated Dropdown' button, you will notice the dropdown list can display only when the arrow is clicked. This is because the dropdown button has been separated, and only clicking-on-arrow can trigger the dropdown list. But you can make the dropdown list display by clicking any area on the **Un-separated Dropdown** button.



```
<html>
  <template id="layout-template">
    <webpdf>
      <div>
        <!-- By default, the value of dropdown's 'separate' option is true -->
        <!-- Set selected="0" means when you click on the dropdown button, it will trigger the event for the
first item in the dropdown list -->
        <dropdown name="separate-dropdown" icon-class="fv_icon-toolbar-square" text="Separated
Dropdown" selected="0">
          <xbutton name="separate-dropdown-square-btn" icon-class="fv_icon-toolbar-
square">Square</xbutton>
```

```

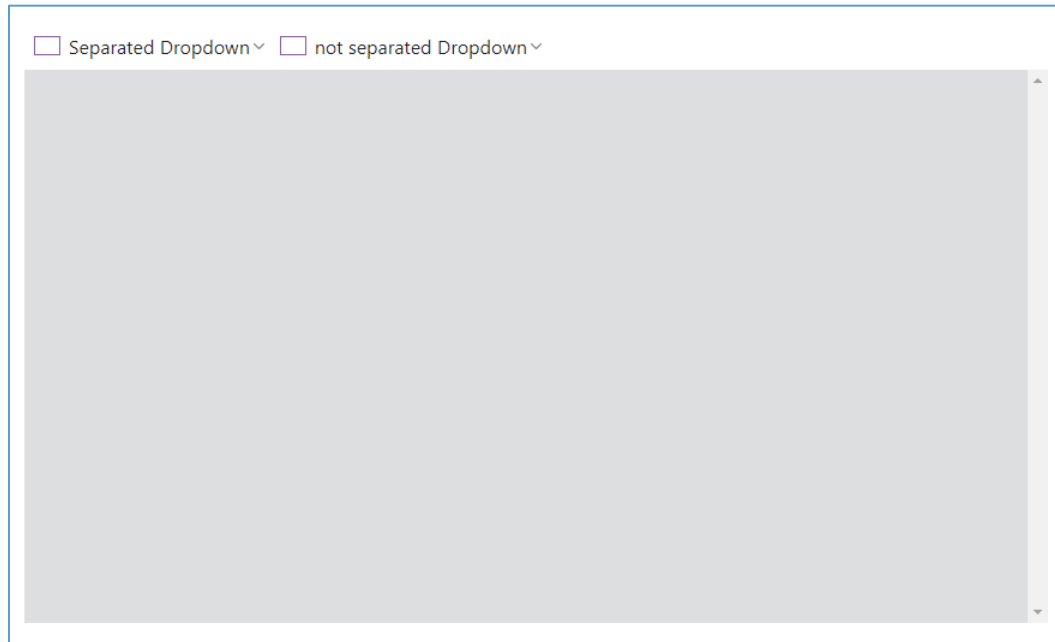
        <xbutton icon-class="fv__icon-toolbar-circle">Circle</xbutton>
        <file-selector>Select a file</file-selector>
        <li class="my-dropdown-list-item">
            html &lt;li&gt; tag
        </li>
    </dropdown>
    <dropdown name="non-separate-dropdown" icon-class="fv__icon-toolbar-shape" text="Un-separated
Dropdown" separate="false">
        <xbutton icon-class="fv__icon-toolbar-square">Square</xbutton>
        <xbutton icon-class="fv__icon-toolbar-circle">Circle</xbutton>
    </dropdown>
</div>
<div class="fv__ui-body">
    <viewer></viewer>
</div>
</webpdf>
</template>
</html>
<style>
.my-dropdown-list-item {
    padding: 10px 0;
    text-align: center;
}
.fv__ui-dropdown {
    width: auto;
}
</style>
<script>
var CustomAppearance = UIExtension.appearances.Appearance.extend({
    getLayoutTemplate: function() {
        return document.getElementById('layout-template').innerHTML;
    },
    getDefaultFragments: function() {
        return [{
            target: 'separate-dropdown-square-btn',
            config: [{
                callback: function() {
                    alert('Click on separate Dropdown');
                }
            }]
        }];
    },
    disableAll: function(){}
});
var libPath = window.top.location.origin + '/lib';
var pdfui = new UIExtension.PDFUI({
    viewerOptions: {
        libPath: libPath,
        jr: {
            licenseSN: licenseSN,
            licenseKey: licenseKey

```



```
    }  
  },  
  renderTo: document.body,  
  appearance: CustomAppearance,  
  addons: []  
});  
</script>
```

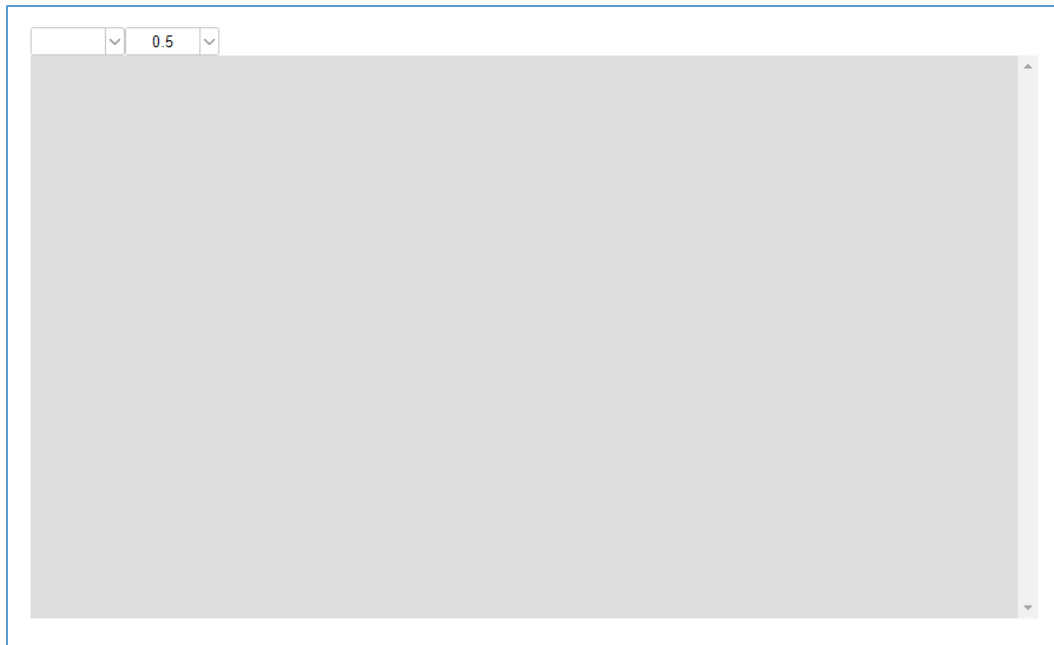
Use as select component



```
<html>  
  <template id="layout-template">  
    <webpdf>  
      <div>  
        <!-- Specify selected="0" that is the initial value -->  
        <dropdown name="separate-dropdown" icon-class="fv__icon-toolbar-square" text="Separated  
Dropdown" selected="0">  
          <xbutton name="separate-dropdown-square-btn" icon-class="fv__icon-toolbar-  
square">Square</xbutton>  
          <xbutton icon-class="fv__icon-toolbar-circle">Circle</xbutton>  
        </dropdown>  
        <dropdown name="not-separate-dropdown" icon-class="fv__icon-toolbar-shape" text="not separated  
Dropdown" separate="false" selected="0">  
          <xbutton icon-class="fv__icon-toolbar-square">Square</xbutton>  
          <xbutton icon-class="fv__icon-toolbar-circle">Circle</xbutton>  
        </dropdown>  
      </div>  
      <div class="fv__ui-body">  
        <viewer></viewer>  
      </div>
```

```
</webpdf>
</template>
</html>
<style>
.my-dropdown-list-item {
    padding: 10px 0;
    text-align: center;
}
.fv__ui-dropdown {
    width: auto;
}
</style>
<script>
var CustomAppearance = UIExtension.appearances.Appearance.extend({
    getLayoutTemplate: function() {
        return document.getElementById('layout-template').innerHTML;
    },
    getDefaultFragments: function() {
        return [{
            target: 'separate-dropdown @xbutton,not-separate-dropdown @xbutton',
            config: [{
                callback: function() {
                    this.component.parent.select(this.component);
                }
            }]
        }];
    },
    disableAll: function(){}
});
var libPath = window.top.location.origin + '/lib';
var pdfui = new UIExtension.PDFUI({
    viewerOptions: {
        libPath: libPath,
        jr: {
            licenseSN: licenseSN,
            licenseKey: licenseKey
        }
    },
    renderTo: document.body,
    appearance: CustomAppearance,
    addons: []
});
</script>
```

Editable

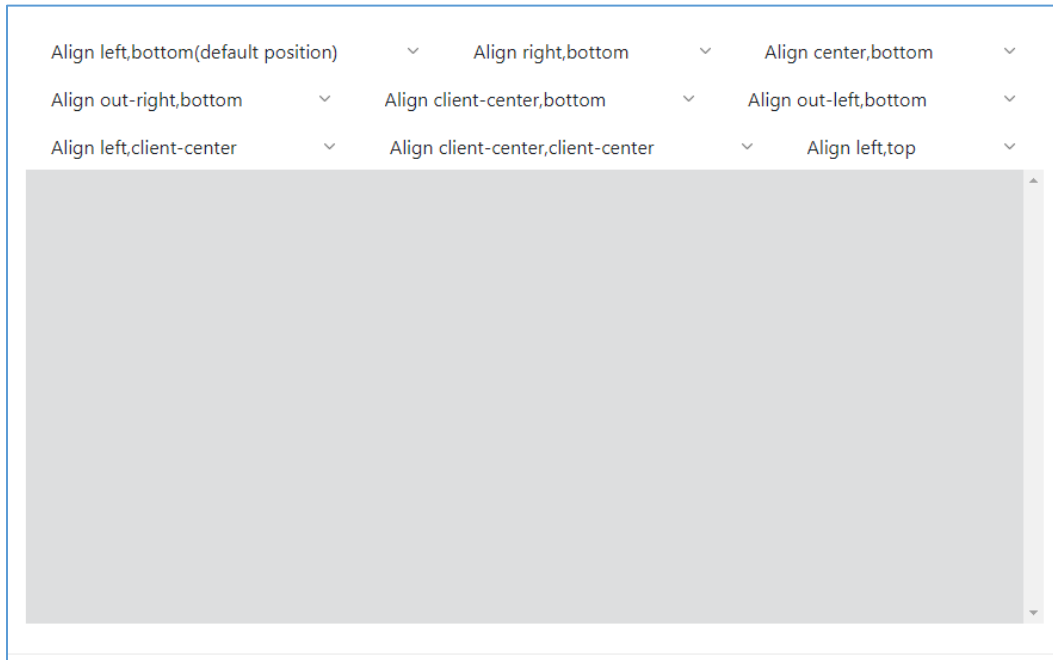


```
<html>
  <template id="layout-template">
    <webpdf>
      <div>
        <dropdown name="font-editable-dropdown" editable>
          <xbutton>Helvetica</xbutton>
          <xbutton>Courier</xbutton>
          <xbutton>Times-Bold</xbutton>
          <xbutton>宋体</xbutton>
        </dropdown>
        <dropdown name="zoom-editable-dropdown" editable @controller="custom:ZoomPageController">
          <xbutton @controller="custom:ScaleRatioController" scale="0.5">50%</xbutton>
          <xbutton @controller="custom:ScaleRatioController" scale="0.75">75%</xbutton>
          <xbutton @controller="custom:ScaleRatioController" scale="1">100%</xbutton>
        </dropdown>
      </div>
      <div class="fv__ui-body">
        <viewer></viewer>
      </div>
    </webpdf>
  </template>
</html>
<style>
  .fv__ui-dropdown {
    width: 80px;
  }
</style>
<script>
  UIExtension.PDFUI.module('custom', [])
    .controller('ScaleRatioController', {
```

```
handle: function() {
    const scaleRatio = parseFloat(this.component.element.getAttribute('scale'));
    debugger;
    this.component.parent.setEditValue(scaleRatio);
}
})
.controller('ZoomPageController', {
    mounted: function() {
        const component = this.component;
        const firstChild = component.getChildAt(0);
        const scaleRatio = parseFloat(firstChild.element.getAttribute('scale'))
        component.setEditValue(scaleRatio);
        component.on('change', function(newValue, oldValue) {
            if(isNaN(parseFloat(newValue))) {
                alert('Illegal scale value: ' + newValue);
                component.setEditValue(oldValue);
                return;
            }
            alert('scale value changed to: ' + newValue)
        })
    }
});
var CustomAppearance = UIExtension.appearances.Appearance.extend({
    getLayoutTemplate: function() {
        return document.getElementById('layout-template').innerHTML;
    },
    getDefaultFragments() {
        return [{
            target: 'zoom-editable-dropdown',
            config: {
                editOptions: {
                    type: 'number',
                    min: 0,
                    max: 10,
                    step: 0.01
                }
            }
        }];
    },
    disableAll: function(){}
});
var libPath = window.top.location.origin + '/lib';
var pdfui = new UIExtension.PDFUI({
    viewerOptions: {
        libPath: libPath,
        jr: {
            licenseSN: licenseSN,
            licenseKey: licenseKey
        }
    },
    renderTo: document.body,
```

```
appearance: CustomAppearance,
addons: []
});
</script>
```

Position the dropdown list



```
<html>
<template id="layout-template">
  <webpdf>
    <div class="flex-with-gap">
      <dropdown text="Align left,bottom(default position)" align="left" valign="bottom">
        <li>left bottom</li>
      </dropdown>
      <dropdown text="Align right,bottom" align="right" valign="bottom">
        <li>right bottom</li>
      </dropdown>
      <dropdown text="Align center,bottom" align="center" valign="bottom">
        <li>center bottom</li>
      </dropdown>
    </div>
    <div class="flex-with-gap">
      <dropdown text="Align out-right,bottom" align="out-right" valign="bottom">
        <li>out-right bottom</li>
      </dropdown>
      <dropdown text="Align client-center,bottom" align="client-center" valign="bottom">
        <li>client-center bottom</li>
      </dropdown>
      <dropdown text="Align out-left,bottom" align="out-left" valign="bottom">
        <li>out-left bottom</li>
      </dropdown>
    </div>
  </webpdf>
</template>
```

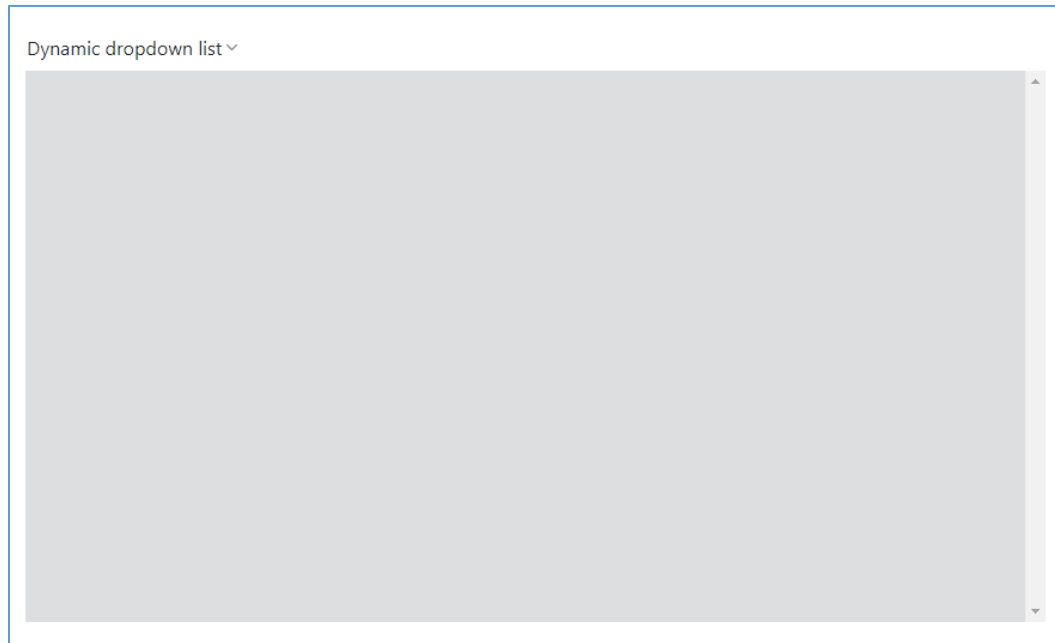
```

        </dropdown>
    </div>
    <div class="flex-with-gap">
        <dropdown text="Align left,client-center" align="left" valign="client-center">
            <li>left client-center</li>
        </dropdown>
        <dropdown text="Align client-center,client-center" align="client-center" valign="client-center">
            <li>client-center</li>
        </dropdown>
        <dropdown text="Align left,top" align="left" valign="top">
            <li>left top</li>
        </dropdown>
    </div>
    <div class="fv__ui-body">
        <viewer></viewer>
    </div>
</webpdf>
</template>
</html>
<style>
.fv__ui-dropdown {
    width: auto;
}
.flex-with-gap {
    display: flex;
    flex-direction: row;
    justify-content: center;
}
.flex-with-gap>.fv__ui-dropdown {
    margin: 0 20px;
    flex: 1 1 auto;
}
</style>
<script>
var CustomAppearance = UIExtension.appearances.Appearance.extend({
    getLayoutTemplate: function() {
        return document.getElementById('layout-template').innerHTML;
    },
    disableAll: function(){}
});
var libPath = window.top.location.origin + '/lib';
var pdfui = new UIExtension.PDFUI({
    viewerOptions: {
        libPath: libPath,
        jr: {
            licenseSN: licenseSN,
            licenseKey: licenseKey
        }
    },
    renderTo: document.body,
    appearance: CustomAppearance,

```

```
addons: []  
});  
</script>
```

Dynamic dropdown list



```
<html>  
  <template id="layout-template">  
    <webpdf>  
      <div>  
        <dropdown separate="false" @controller="custom:DropdownItemListController as ctrl" text="Dynamic  
dropdown list">  
          <li style="padding-left: 1em;">Click button to create more</li>  
          <li @foreach="item in ctrl.items track by id">  
            <text @sync.text="item.text"></text>  
          </li>  
          <xbutton icon-class="fv__icon-toolbar-add-sign" @controller="custom:AddItemController">Add  
dropdown item</xbutton>  
        </dropdown>  
      </div>  
      <div class="fv__ui-body">  
        <viewer></viewer>  
      </div>  
    </webpdf>  
  </template>  
</html>  
<style>  
  .fv__ui-dropdown {  
    width: auto;  
  }
```

```
</style>
<script>
  UIExtension.PDFUI.module('custom', [])
    .controller('DropdownItemListController', {
      init: function() {
        this.items = [{
          id: Date.now().toString(16),
          text: new Date().toLocaleString()
        }];
      },
      addItem: function(data) {
        this.items = this.items.concat(data);
        this.digest();
      }
    })
    .controller('AddItemController', {
      handle: function() {
        const itemListCtrl = this.data.ctrl;
        itemListCtrl.addItem({
          id: Date.now().toString(16),
          text: new Date().toLocaleString()
        });
      }
    })
  var CustomAppearance = UIExtension.appearances.Appearance.extend({
    getLayoutTemplate() {
      return document.getElementById('layout-template').innerHTML;
    },
    disableAll(){}
  });
  var libPath = window.top.location.origin + '/lib';
  var pdfui = new UIExtension.PDFUI({
    viewerOptions: {
      libPath: libPath,
      jr: {
        licenseSN: licenseSN,
        licenseKey: licenseKey
      }
    },
    renderTo: document.body,
    appearance: CustomAppearance,
    addons: []
  });
</script>
```

API

Dropdown component template

Template example:


```
<dropdown text="" icon-class="" editable align="left" valign="bottom" separate="true"
selected="0"></dropdown>
```

The template properties:

Property	Description	Type	Default Value	Version
text	Set text for dropdown button	string	''	7.0
icon-class	Set the icon's css class	string	''	7.0
editable	If editable	boolean	false	7.0
align	Horizontal alignment	'left' 'right' 'out-right' 'out-left' 'center' 'client-center'	'left'	7.0
valign	Vertical alignment	'top' 'bottom' 'center' 'client-center'	'bottom'	7.0
separate	If the dropdown button is separated	boolean	true	7.0

Configure dropdown properties using fragment

Besides the `editOptions`, the others are same as the template properties.

```
{
  target: 'dropdown-name',
  config: {
    editOptions: {
      type: 'text',
      min: 0,
      max: 0,
      step: 0,
      value: ""
    }
  }
}
```

Property	Description	Type	Default Value	Version
editOptions.type	Set edit mode for dropdown. It supports both the text and 'number' edit mode	string	'text'	7.0
editOptions.min	The minimum value for the edit box. It is valid only when the edit mode is 'number'	number		7.0
editOptions.max	The maximum value for the edit box. It is valid only when the edit mode is 'number'	number		7.0
editOptions.step	The step for the edit box. It is valid only when the edit mode is 'number'	number		7.0
editOptions.value	The initial value of the edit box	string number		7.0

Dropdown object properties

Property	Description	Type
disabled	Button disabled status	boolean
isVisible	Button visible status	boolean
isActive	Check if the dropdown list is active	boolean

Methods

Method	Description	Version
setEditValue(text: String number): void	Set the input value. This won't trigger the <code>change</code> event	7.0
disable(): void	Disable dropdown.	7.0
enable(): void	Enable the disabled dropdown	7.0
show(): void	Show the hidden dropdown	7.0

Method	Description	Version
hide(): void	Hide the dropdown	7.0
active(): void	Open the dropdown	7.0
deactive(): void	Close the dropdown	7.0
destroy(): void	Destroy the component	7.0

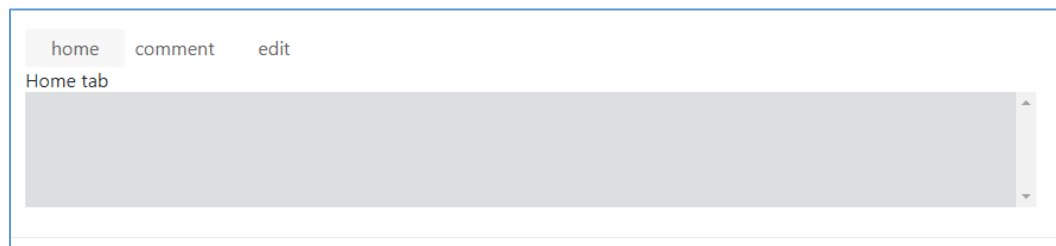
Events

Name	Description	Example	Version
active	Triggered upon the dropdown expands	dropdown.on('active', () => {})	7.0
deactive	Triggered upon the dropdown hides	dropdown.on('deactive', () => {})	7.0
change	Triggered upon the mouse enters and focus loses	dropdown.on('change', (newValue,oldValue) => {})	7.0

Tab component

Code examples

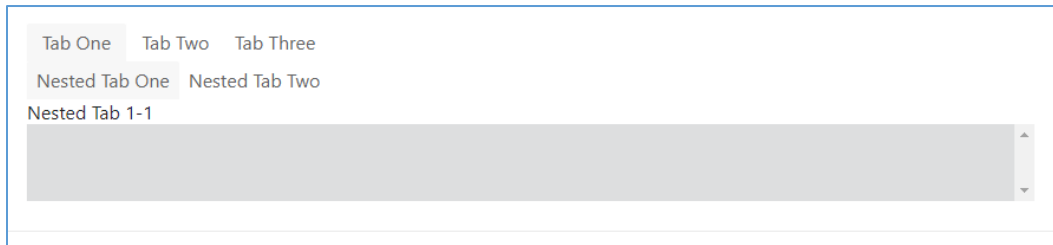
Basic tab example



```
<html>
  <template id="layout-template">
    <webpdf>
      <div>
        <div>
          <div class="tabs">
            <gtab group="top-toolbar-tab" body="home-tab" active>home</gtab>
            <gtab group="top-toolbar-tab" body="comment-tab">comment</gtab>
```

```
    <gtab group="top-toolbar-tab" body="edit-tab">edit</gtab>
  </div>
  <div class="tab-bodies">
    <div name="home-tab">
      Home tab
    </div>
    <div name="comment-tab">
      Comment tab
    </div>
    <div name="edit-tab">
      Edit tab
    </div>
  </div>
</div>
</div>
<div class="fv__ui-body">
  <viewer></viewer>
</div>
</webpdf>
</template>
</html>
<script>
  var CustomAppearance = UIExtension.appearances.Appearance.extend({
    getLayoutTemplate: function() {
      return document.getElementById('layout-template').innerHTML;
    },
    disableAll: function(){}
  });
  var libPath = window.top.location.origin + '/lib';
  var pdfui = new UIExtension.PDFUI({
    viewerOptions: {
      libPath: libPath,
      jr: {
        licenseSN: licenseSN,
        licenseKey: licenseKey
      }
    },
    renderTo: document.body,
    appearance: CustomAppearance,
    addons: []
  });
</script>
```

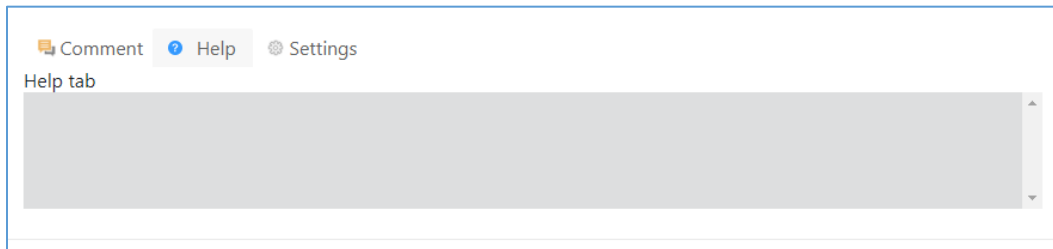
Nested tab



```
<html>
  <template id="layout-template">
    <webpdf>
      <div>
        <div>
          <div class="tabs">
            <gtab group="top-toolbar-tab" body="tab1" active>Tab One</gtab>
            <gtab group="top-toolbar-tab" body="tab2">Tab Two</gtab>
            <gtab group="top-toolbar-tab" body="tab3">Tab Three</gtab>
          </div>
          <div class="tab-bodies">
            <div name="tab1">
              <div class="tabs">
                <gtab group="nested-tab1" body="nested-tab1-1" active>Nested Tab One</gtab>
                <gtab group="nested-tab1" body="nested-tab1-2">Nested Tab Two</gtab>
              </div>
              <div name="nested-tab1-1">Nested Tab 1-1</div>
              <div name="nested-tab1-2">Nested Tab 1-2</div>
            </div>
            <div name="tab2">
              Tab Two
            </div>
            <div name="tab3">
              <div class="tabs">
                <gtab group="nested-tab3" body="nested-tab3-1">Nested Tab3 One</gtab>
                <gtab group="nested-tab3" body="nested-tab3-2" active>Nested Tab3 Two</gtab>
                <gtab group="nested-tab3" body="nested-tab3-3">Nested Tab3 Two</gtab>
              </div>
              <div name="nested-tab3-1">Nested Tab 3-1</div>
              <div name="nested-tab3-2">Nested Tab 3-2</div>
              <div name="nested-tab3-3">Nested Tab 3-3</div>
            </div>
          </div>
        </div>
      </div>
      <div class="fv_ui-body">
        <viewer></viewer>
      </div>
    </webpdf>
  </template>
</html>
<script>
```

```
var CustomAppearance = UIExtension.appearances.Appearance.extend({
  getLayoutTemplate: function() {
    return document.getElementById('layout-template').innerHTML;
  },
  disableAll: function(){}
});
var libPath = window.top.location.origin + '/lib';
var pdfui = new UIExtension.PDFUI({
  viewerOptions: {
    libPath: libPath,
    jr: {
      licenseSN: licenseSN,
      licenseKey: licenseKey
    }
  },
  renderTo: document.body,
  appearance: CustomAppearance,
  addons: []
});
</script>
```

Leading icon



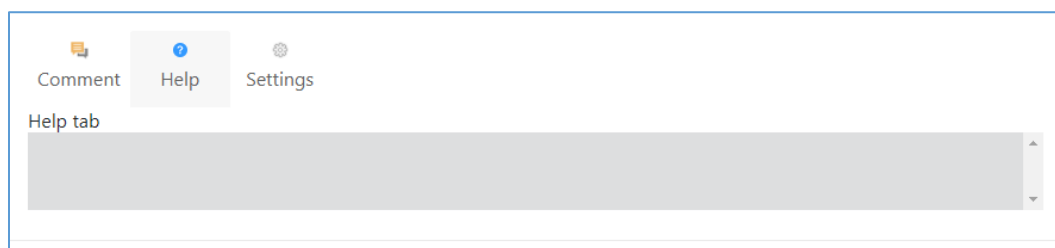
```
<html>
  <template id="layout-template">
    <webpdf>
      <div>
        <div>
          <div class="tabs">
            <gtab group="top-toolbar-tab" body="comment-tab" active icon-class="fv__icon-sidebar-comment-
list">Comment</gtab>
            <gtab group="top-toolbar-tab" body="help-tab" active icon-class="fv__icon-dialog-level-
question">Help</gtab>
            <gtab group="top-toolbar-tab" body="settings-tab" icon-class="fv__icon-comment-item-menu-
settings">Settings</gtab>
          </div>
          <div class="tab-bodies">
            <div name="comment-tab">
              Comment tab
            </div>
            <div name="help-tab">
              Help tab
            </div>
          </div>
        </div>
      </div>
    </webpdf>
  </template>
</html>
```

```

        </div>
        <div name="settings-tab">
            Settings tab
        </div>
    </div>
</div>
<div class="fv__ui-body">
    <viewer></viewer>
</div>
</webpdf>
</template>
</html>
<script>
    var CustomAppearance = UIExtension.appearances.Appearance.extend({
        getLayoutTemplate: function() {
            return document.getElementById('layout-template').innerHTML;
        },
        disableAll: function(){}
    });
    var libPath = window.top.location.origin + '/lib';
    var pdfui = new UIExtension.PDFUI({
        viewerOptions: {
            libPath: libPath,
            jr: {
                licenseSN: licenseSN,
                licenseKey: licenseKey
            }
        },
        renderTo: document.body,
        appearance: CustomAppearance,
        addons: []
    });
</script>

```

Top icon



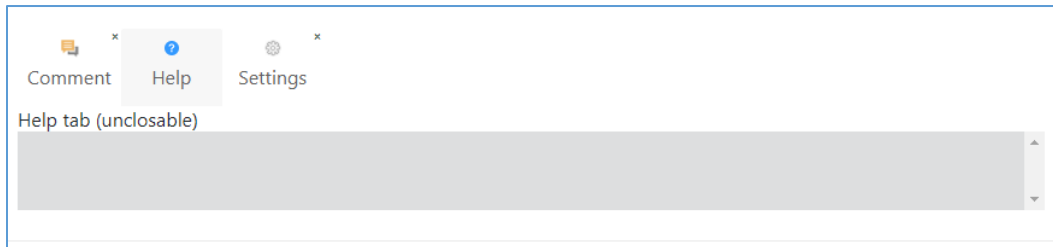
```

<html>
    <template id="layout-template">
        <webpdf>
            <div>
                <div>

```

```
<div class="tabs">
  <gtab class="stacked" group="top-toolbar-tab" body="comment-tab" active icon-class="fv__icon-
sidebar-comment-list">Comment</gtab>
  <gtab class="stacked" group="top-toolbar-tab" body="help-tab" active icon-class="fv__icon-dialog-
level-question">Help</gtab>
  <gtab class="stacked" group="top-toolbar-tab" body="settings-tab" icon-class="fv__icon-comment-
item-menu-settings">Settings</gtab>
</div>
<div class="tab-bodies">
  <div name="comment-tab">
    Comment tab
  </div>
  <div name="help-tab">
    Help tab
  </div>
  <div name="settings-tab">
    Settings tab
  </div>
</div>
</div>
<div class="fv__ui-body">
  <viewer></viewer>
</div>
</webpdf>
</template>
</html>
<script>
  var CustomAppearance = UIExtension.appearances.Appearance.extend({
    getLayoutTemplate: function() {
      return document.getElementById('layout-template').innerHTML;
    },
    disableAll: function(){}
  });
  var libPath = window.top.location.origin + '/lib';
  var pdfui = new UIExtension.PDFUI({
    viewerOptions: {
      libPath: libPath,
      jr: {
        licenseSN: licenseSN,
        licenseKey: licenseKey
      }
    },
    renderTo: document.body,
    appearance: CustomAppearance,
    addons: []
  });
</script>
```

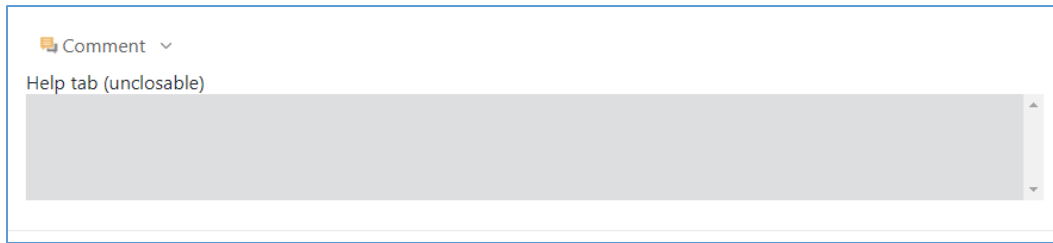
Closable tabs



```
<html>
  <template id="layout-template">
    <webpdf>
      <div>
        <div>
          <div class="tabs">
            <!-- Comment tab will be destroy after closed -->
            <gtab class="stacked" group="top-toolbar-tab" body="comment-tab" icon-class="fv__icon-sidebar-
comment-list" closable destroy-on-close>Comment</gtab>
            <gtab class="stacked" group="top-toolbar-tab" body="help-tab" active icon-class="fv__icon-dialog-
level-question">Help</gtab>
            <gtab @controller="custom:SettingsTabController" name="settings-tab-ctrl" class="stacked"
group="top-toolbar-tab" body="settings-tab" icon-class="fv__icon-comment-item-menu-settings"
closable>Settings</gtab>
            <xbutton visible="false" name="open-settings-tab-btn" class="open-settings-tab-btn"
@controller="custom:ReopenTabController" icon-class="fv__icon-toolbar-add-sign" @tooltip tooltip-
title="Reopen settings tab"></xbutton>
          </div>
          <div class="tab-bodies">
            <div name="comment-tab">
              Comment tab(closable,and will be destroyed after closing)
            </div>
            <div name="help-tab">
              Help tab (unclosable)
            </div>
            <div name="settings-tab">
              Settings tab (closable)
            </div>
          </div>
        </div>
      </div>
      <div class="fv__ui-body">
        <viewer></viewer>
      </div>
    </webpdf>
  </template>
</html>
<style>
  .tabs {
    display: flex;
    align-items: center;
  }
</style>
```

```
.open-settings-tab-btn {
  display: inline-flex;
  width: 32px;
  height: 32px;
}
</style>
<script>
  UIExtension.PDFUI.module('custom',[])
  .controller('ReopenTabController', {
    handle: function() {
      this.getComponentByName('settings-tab-ctrl').open();
    }
  })
  .controller('SettingsTabController', {
    mounted: function() {
      this.component.on('close', () => {
        this.getComponentByName('open-settings-tab-btn').show();
      });
      this.component.on('open', () => {
        this.getComponentByName('open-settings-tab-btn').hide();
      });
    }
  });
  var CustomAppearance = UIExtension.appearances.Appearance.extend({
    getLayoutTemplate: function() {
      return document.getElementById('layout-template').innerHTML;
    },
    disableAll: function(){}
  });
  var libPath = window.top.location.origin + '/lib';
  var pdfui = new UIExtension.PDFUI({
    viewerOptions: {
      libPath: libPath,
      jr: {
        licenseSN: licenseSN,
        licenseKey: licenseKey
      }
    },
    renderTo: document.body,
    appearance: CustomAppearance,
    addons: []
  });
</script>
```

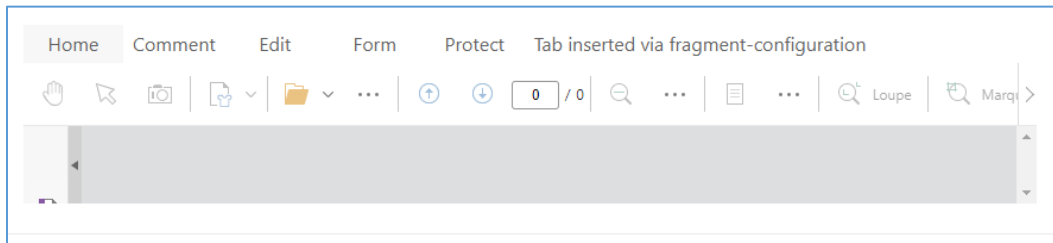
Tabs with dropdown



```
<html>
  <template id="layout-template">
    <webpdf>
      <div>
        <div>
          <div class="tabs">
            <gtab group="top-toolbar-tab" body="comment-tab" icon-class="fv__icon-sidebar-comment-
list">Comment</gtab>
            <dropdown>
              <li>
                <gtab group="top-toolbar-tab" body="help-tab" active icon-class="fv__icon-dialog-level-
question">Help</gtab>
              </li>
              <li>
                <gtab name="settings-tab-ctrl" group="top-toolbar-tab" body="settings-tab" icon-
class="fv__icon-comment-item-menu-settings">Settings</gtab>
              </li>
            </dropdown>
          </div>
          <div class="tab-bodies">
            <div name="comment-tab">
              Comment tab(closable and will be destroy after closed)
            </div>
            <div name="help-tab">
              Help tab (unclosable)
            </div>
            <div name="settings-tab">
              Settings tab (closable)
            </div>
          </div>
        </div>
      </div>
      <div class="fv__ui-body">
        <viewer></viewer>
      </div>
    </webpdf>
  </template>
</html>
<style>
  .tabs {
    display: flex;
    align-items: center;
```

```
}
</style>
<script>
  var CustomAppearance = UIExtension.appearances.Appearance.extend({
    getLayoutTemplate: function() {
      return document.getElementById('layout-template').innerHTML;
    },
    disableAll: function(){}
  });
  var libPath = window.top.location.origin + '/lib';
  var pdfui = new UIExtension.PDFUI({
    viewerOptions: {
      libPath: libPath,
      jr: {
        licenseSN: licenseSN,
        licenseKey: licenseKey
      }
    },
    renderTo: document.body,
    appearance: CustomAppearance,
    addons: []
  });
</script>
```

Insert a tab using fragment-configuration



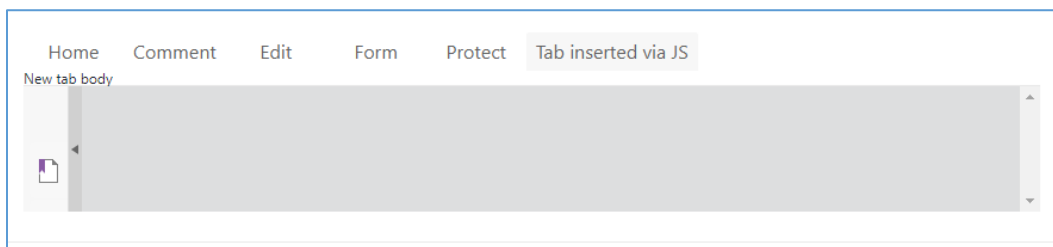
```
<html>
</html>
<style>
  .tabs {
    display: flex;
    align-items: center;
  }
</style>
<script>
  var FRAGMENT_ACTION = UIExtension.UIConsts.FRAGMENT_ACTION;
  var CustomAppearance = UIExtension.appearances.RibbonAppearance.extend({
    getDefaultFragments: function() {
      return [{
        target: 'toolbar-tabs',
        action: FRAGMENT_ACTION.APPEND,
        template: '<gtab name="new-tab" group="toolbar-tab" body="new-tab-body">Tab inserted via
fragment-configuration</gtab>'
      }];
    }
  });
```

```

    }, {
      target: 'toolbar-tab-bodies',
      action: FRAGMENT_ACTION.APPEND,
      template: '<div name="new-tab-body" style="line-height:1"><text>New tab body</text></div>'
    });
  }
});
var libPath = window.top.location.origin + '/lib';
var pdfui = new UIExtension.PDFUI({
  viewerOptions: {
    libPath: libPath,
    jr: {
      licenseSN: licenseSN,
      licenseKey: licenseKey
    }
  },
  renderTo: document.body,
  appearance: CustomAppearance,
  addons: []
});
</script>

```

Dynamically insert a tab using JavaScript



```

<html>
</html>
<style>
  .tabs {
    display: flex;
    align-items: center;
  }
</style>
<script>
  var libPath = window.top.location.origin + '/lib';
  var pdfui = new UIExtension.PDFUI({
    viewerOptions: {
      libPath: libPath,
      jr: {
        licenseSN: licenseSN,
        licenseKey: licenseKey
      }
    },
    renderTo: document.body,

```

```

        appearance: UIExtension.appearances.RibbonAppearance,
        addons: []
    });
    pdfui.getRootComponent().then(root => {
        // the component name can be found in 'examples/UIExtension/layout-templates/built-in-pc-layout-
        template.tpl'
        var tabs = root.getComponentByName('toolbar-tabs');
        var tabBodies = root.getComponentByName('toolbar-tab-bodies');
        // insert a div named in 'new-tab-body'
        tabBodies.append('<div name="new-tab-body" style="line-height:1"><text>New tab body</text></div>');
        // insert a tab into 'toolbar-tabs' and specifies the tab body name as 'new-tab-body'
        tabs.append('<gtab name="new-tab" body="new-tab-body" group="toolbar-tab">Tab inserted via
JS</gtab>');
        // activate new tab
        var newTab = tabs.getComponentByName('new-tab');
        newTab.active();
    })
</script>

```

API

Tab component template

Template example:

```

<div class="tabs">
  <gtab group="mytabs" body="tab1" class="stacked" icon-class="fv_icon-comment-item-menu-settings"
active>Tab Text 1</gtab>
  <gtab group="mytabs" body="tab2" class="stacked" closable destroy-on-close>Tab Text 2</gtab>
</div>
<div class="tab-bodies">
  <div name="tab1">Tab One</div>
  <div name="tab2">Tab Two</div>
</div>

```

The tab component template properties:

Property	Description	Type	Default Value	Version
group	Only one tab can be activated in a group, similar to the radio name	string	--	7.4.0
body	Tab body's name	string	--	7.4.0
closable	Define if the tab is closable	boolean	false	7.4.0

Property	Description	Type	Default Value	Version
destroy-on-close	Whether to destroy tab after the tab is closed. Once the tab is destroyed, the body component will be destructed also.			
Boolean		false		7.4.0
class="stacked"	Define the tab's 'icon and text' to be displayed in two lines, otherwise in one line.	--	--	7.4.0

Tab object properties

Property	Description	Type
disabled	Disabled status	boolean
isVisible	Visibility status	boolean
isActive	Activity status	boolean
isClosed	Closes	boolean

Methods

Method	Description	Version
disable(): void	Disable a tab	7.0.0
enable(): void	Make a disabled tab enabled	7.0.0
show(): void	Make a hidden tab shown	7.0.0
hide(): void	Hide a tab	7.0.0
active(): void	Activate a tab	7.0.0
deactive(): void	Make a tab inactive	7.0.0
destroy(): void	Destroy a tab	7.0.0

Method	Description	Version
close(): void	Close a tab. The closed tab will be hidden, and the adjacent tab will be activated	7.4.0
open(): void	Make a closed tab shown	7.4.0

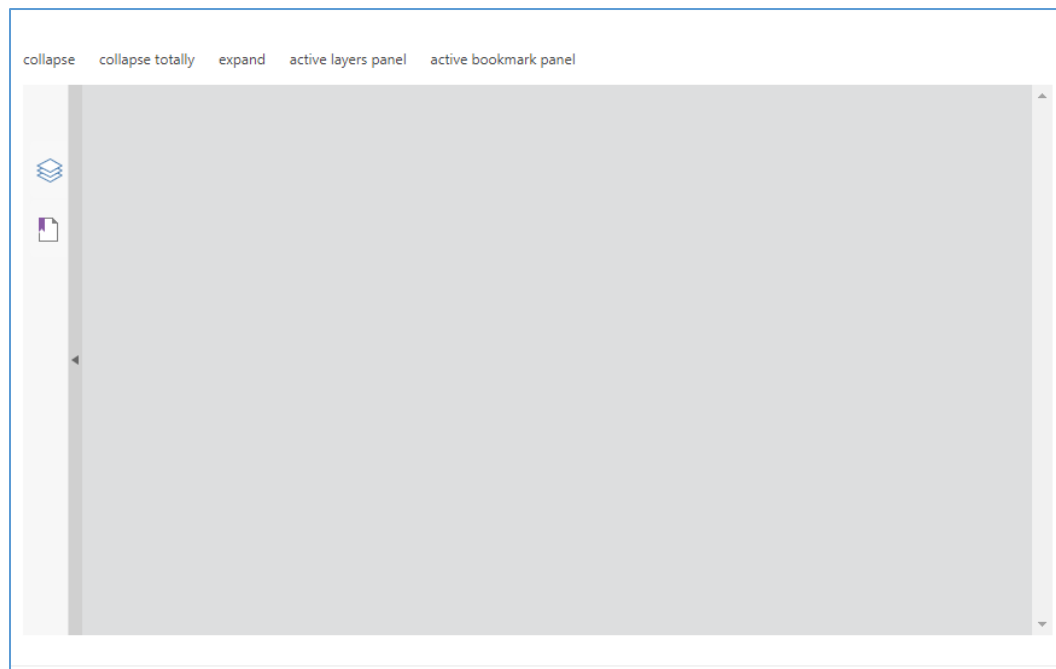
Events

Name	Description	Sample	Version
active	Triggered when a tab is activated	<code>gtab.on('active', () => {})</code>	7.0.0
deactive	Triggered when a tab is deactivated	<code>gtab.on('deactive', () => {})</code>	7.0.0
close	Triggered when a tab is closed	<code>gtab.on('close', () => {})</code>	7.4.0
open	Triggered when a tab is opened	<code>gtab.on('open', () => {})</code>	7.4.0

Sidebar Component

Code examples

Basic example



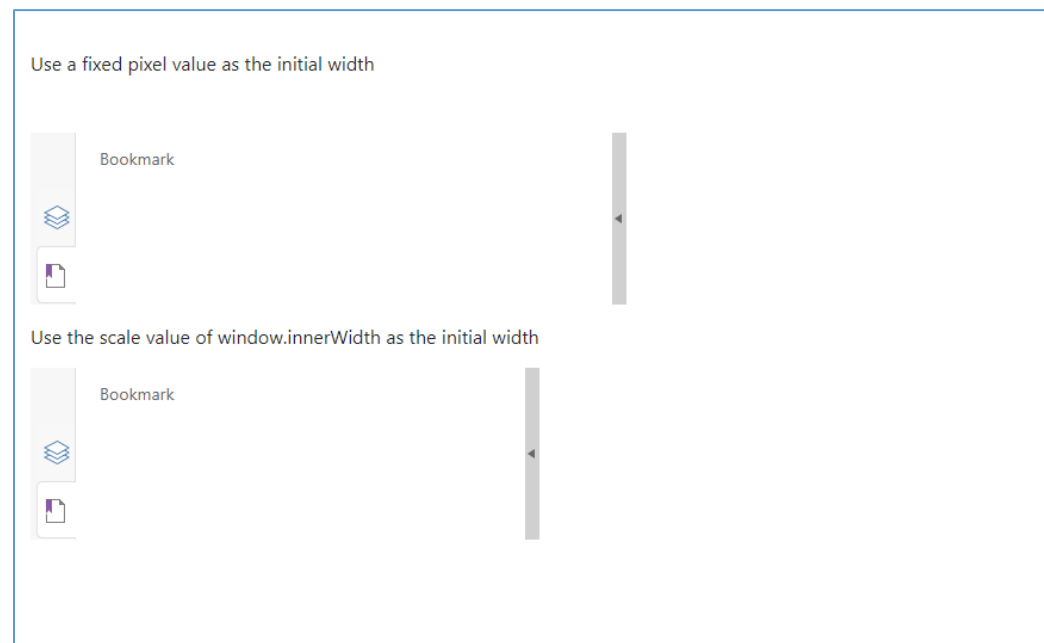

```

<html>
  <template id="layout-template">
    <webpdf>
      <div class="btn-container">
        <xbutton @controller="custom:SidebarActionController" action="collapse">collapse</xbutton>
        <xbutton @controller="custom:SidebarActionController" action="collapse.totally">collapse
totally</xbutton>
        <xbutton @controller="custom:SidebarActionController" action="expand">expand</xbutton>
        <xbutton @controller="custom:SidebarActionController" action="active.layers">active layers
panel</xbutton>
        <xbutton @controller="custom:SidebarActionController" action="active.bookmark">active bookmark
panel</xbutton>
      </div>
      <div class="fv__ui-body">
        <sidebar name="my-sidebar">
          <sidebar-panel name="sidebar-layers" icon-class="fv__icon-sidebar-page-manager"
title="Layers"></sidebar-panel>
          <sidebar-panel name="sidebar-bookmark" active icon-class="fv__icon-sidebar-bookmark"
title="Bookmark"></sidebar-panel>
        </sidebar>
        <viewer></viewer>
      </div>
    </webpdf>
  </template>
</html>
<style>
  .btn-container {
    display: flex;
    padding: 10px 0;
  }
  .btn-container>.fv__ui-button + .fv__ui-button {
    margin-left: 20px;
  }
</style>
<script>
  UIExtension.PDFUI.module('custom', [])
  .controller('SidebarActionController', {
    handle: function() {
      var action = this.component.getAttribute('action');
      var sidebar = this.getComponentByName('my-sidebar');
      switch(action) {
        case 'collapse':
          sidebar.collapse();
          break;
        case 'collapse.totally':
          sidebar.collapseTotally();
          break;
        case 'expand':
          sidebar.expand();
          break;
        case 'active.layers':

```

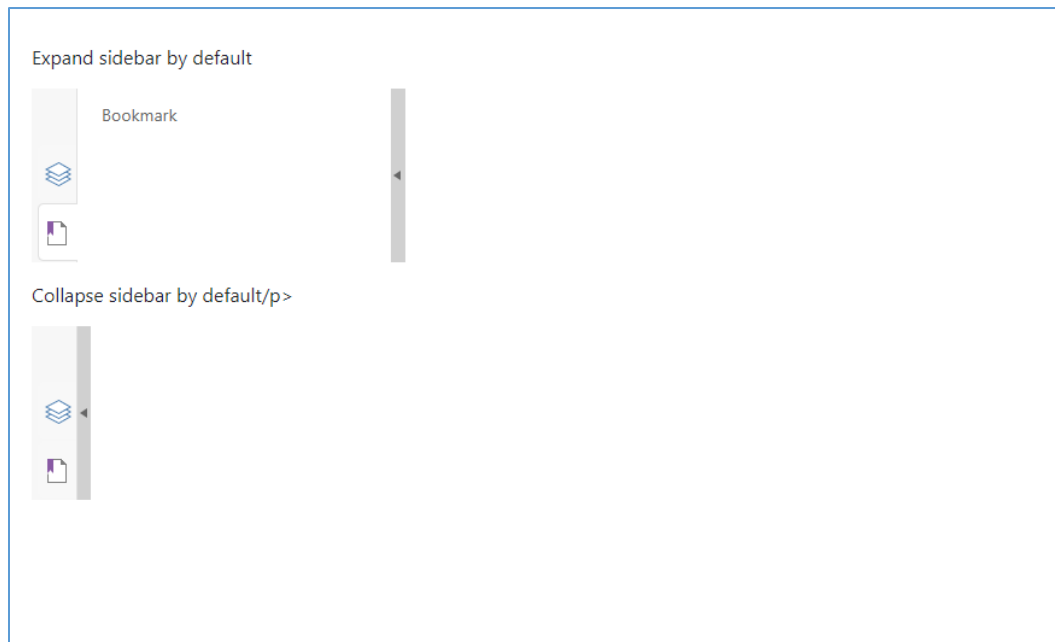
```
        sidebar.getComponentByName('sidebar-layers').active();
        break;
    case 'active.bookmark':
        sidebar.getComponentByName('sidebar-bookmark').active();
        break;
    }
}
})
var CustomAppearance = UIExtension.appearances.Appearance.extend({
    getLayoutTemplate: function() {
        return document.getElementById('layout-template').innerHTML;
    },
    disableAll: function(){}
});
var libPath = window.top.location.origin + '/lib';
var pdfui = new UIExtension.PDFUI({
    viewerOptions: {
        libPath: libPath,
        jr: {
            licenseSN: licenseSN,
            licenseKey: licenseKey
        }
    },
    renderTo: document.body,
    appearance: CustomAppearance,
    addons: []
});
</script>
```

Sidebar width



```
<html>
  <template id="layout-template">
    <webpdf>
      <p> Use a fixed pixel value as the initial width </p>
      <div>
        <sidebar width="500" open>
          <sidebar-panel icon-class="fv__icon-sidebar-page-manager" title="Layers"></sidebar-panel>
          <sidebar-panel active icon-class="fv__icon-sidebar-bookmark" title="Bookmark"></sidebar-panel>
        </sidebar>
      </div>
      <p> Use the scale value of window.innerWidth as the initial width </p>
      <div>
        <sidebar width="0.5" open>
          <sidebar-panel icon-class="fv__icon-sidebar-page-manager" title="Layers"></sidebar-panel>
          <sidebar-panel active icon-class="fv__icon-sidebar-bookmark" title="Bookmark"></sidebar-panel>
        </sidebar>
      </div>
      <div class="hide">
        <viewer></viewer>
      </div>
    </webpdf>
  </template>
</html>
<script>
  var CustomAppearance = UIExtension.appearances.Appearance.extend({
    getLayoutTemplate: function() {
      return document.getElementById('layout-template').innerHTML;
    },
    disableAll: function(){}
  });
  var libPath = window.top.location.origin + '/lib';
  var pdfui = new UIExtension.PDFUI({
    viewerOptions: {
      libPath: libPath,
      jr: {
        licenseSN: licenseSN,
        licenseKey: licenseKey
      }
    },
    renderTo: document.body,
    appearance: CustomAppearance,
    addons: []
  });
</script>
```

Expand and collapse sidebar

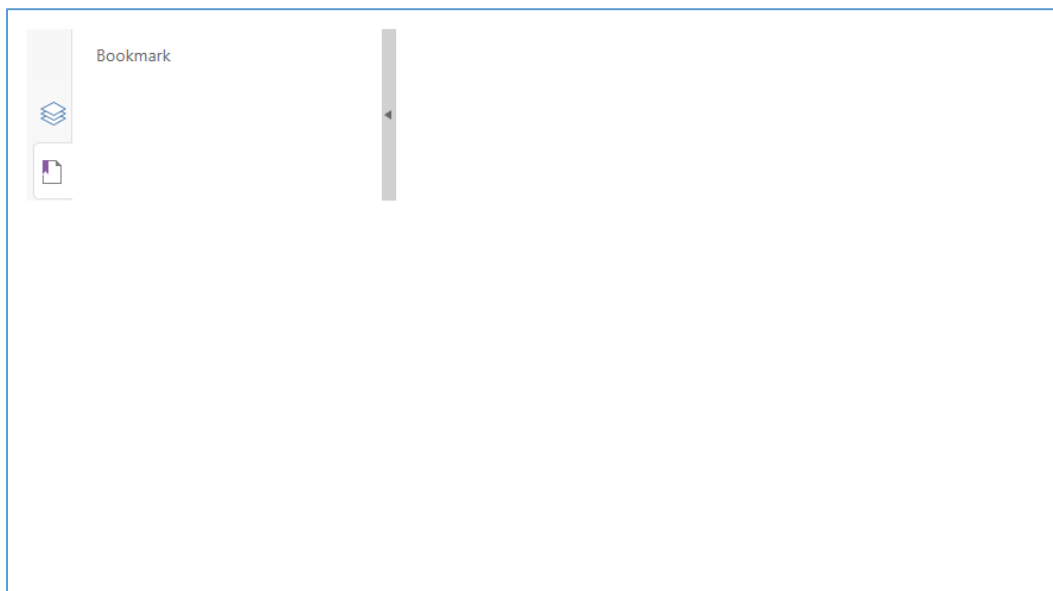


```
<html>
  <template id="layout-template">
    <webpdf>
      <p>Expand sidebar by default</p>
      <div>
        <sidebar open>
          <sidebar-panel icon-class="fv__icon-sidebar-page-manager" title="Layers"></sidebar-panel>
          <sidebar-panel active icon-class="fv__icon-sidebar-bookmark" title="Bookmark"></sidebar-panel>
        </sidebar>
      </div>
      <p>Collapse sidebar by default/p>
      <div>
        <sidebar>
          <sidebar-panel icon-class="fv__icon-sidebar-page-manager" title="Layers"></sidebar-panel>
          <sidebar-panel active icon-class="fv__icon-sidebar-bookmark" title="Bookmark"></sidebar-panel>
        </sidebar>
      </div>
      <div class="hide">
        <viewer></viewer>
      </div>
    </webpdf>
  </template>
</html>
<script>
  var CustomAppearance = UIExtension.appearances.Appearance.extend({
    getLayoutTemplate: function() {
      return document.getElementById('layout-template').innerHTML;
    },
    disableAll: function(){}
  });
```

```
var libPath = window.top.location.origin + '/lib';
var pdfui = new UIExtension.PDFUI({
  viewerOptions: {
    libPath: libPath,
    jr: {
      licenseSN: licenseSN,
      licenseKey: licenseKey
    }
  },
  renderTo: document.body,
  appearance: CustomAppearance,
  addons: []
});
</script>
```

Sidebar buttons' tooltip

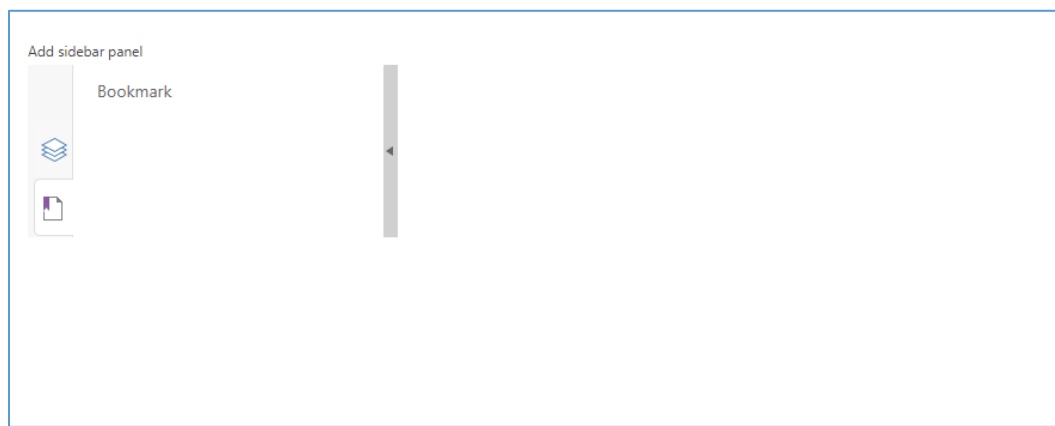
Hover your mouse over a button to show the tooltip.



```
<html>
  <template id="layout-template">
    <webpdf>
      <div>
        <sidebar open>
          <sidebar-panel @tooltip tooltip-title="Layers sidebar panel" tooltip-placement="right" icon-
class="fv__icon-sidebar-page-manager" title="Layers"></sidebar-panel>
          <sidebar-panel @tooltip tooltip-title="Bookmark sidebar panel" tooltip-placement="right" active
icon-class="fv__icon-sidebar-bookmark" title="Bookmark"></sidebar-panel>
        </sidebar>
      </div>
      <div class="hide">
```

```
<viewer></viewer>
</div>
</webpdf>
</template>
</html>
<script>
var CustomAppearance = UIExtension.appearances.Appearance.extend({
  getLayoutTemplate: function() {
    return document.getElementById('layout-template').innerHTML;
  },
  disableAll: function(){}
});
var libPath = window.top.location.origin + '/lib';
var pdfui = new UIExtension.PDFUI({
  viewerOptions: {
    libPath: libPath,
    jr: {
      licenseSN: licenseSN,
      licenseKey: licenseKey
    }
  },
  renderTo: document.body,
  appearance: CustomAppearance,
  addons: []
});
</script>
```

Dynamically insert a sidebar panel using JavaScript



```
<html>
<template id="layout-template">
  <webpdf>
    <div>
      <xbutton @controller="custom:InsertSidebarController">Add sidebar panel</xbutton>
    </div>
  </div>
</div>
```

```

        <sidebar open name="sidebar-component-name">
            <sidebar-panel @tooltip tooltip-title="Layers sidebar panel" tooltip-placement="right" icon-
class="fv__icon-sidebar-page-manager" title="Layers"></sidebar-panel>
            <sidebar-panel @tooltip tooltip-title="Bookmark sidebar panel" tooltip-placement="right" active
icon-class="fv__icon-sidebar-bookmark" title="Bookmark"></sidebar-panel>
        </sidebar>
    </div>
    <div class="hide">
        <viewer></viewer>
    </div>
</webpdf>
</template>
</html>
<script>
    UIExtension.PDFUI.module('custom', [])
        .controller('InsertSidebarController', {
            mounted: function() {
                this.count = 0;
            },
            handle: function() {
                if(this.count >= 3) {
                    return;
                }
                this.count++;
                this.getPDFUI().getComponentByName('sidebar-component-name')
                    .then(sidebar => {
                        sidebar.append(
                            '<sidebar-panel icon-class="fv__icon-sidebar-bookmark" title="Dynamic sidebar panel"></sidebar-
panel>'
                        );
                    })
            }
        });
    var CustomAppearance = UIExtension.appearances.Appearance.extend({
        getLayoutTemplate: function() {
            return document.getElementById('layout-template').innerHTML;
        },
        disableAll: function(){}
    });
    var libPath = window.top.location.origin + '/lib';
    var pdfui = new UIExtension.PDFUI({
        viewerOptions: {
            libPath: libPath,
            jr: {
                licenseSN: licenseSN,
                licenseKey: licenseKey
            }
        },
        renderTo: document.body,
        appearance: CustomAppearance,
        addons: []
    });

```

```
});  
</script>
```

API

Sidebar component template

Template example:

```
<!-- The width value smaller than 1 means that is a scale value of window.innerWidth -->  
<sidebar open width="500">  
  <sidebar-panel icon-class="fv__icon-sidebar-page-manager" title="Layers"></sidebar-panel>  
  <sidebar-panel active icon-class="fv__icon-sidebar-bookmark" title="Bookmark"></sidebar-panel>  
</sidebar>
```

The `sidebar` component template properties:

Property	Decription	Type	Default Value	Version
open	Expand Status	boolean	false	7.0.0
width	The width of the expanded Sidebar will be used as pixel value if it is greater than or equal to 1, and will be calculated by multiplying <code>window.innerWidth</code> if it is less than 1	number	310px	7.0.0

The `sidebar-panel` properties:

Property	Description	Type	Default value	Version
title	The string show at the top of expanded sidebar-panel	string	''	7.0.0
active	If the sidebar-panel is active	boolean	false	7.0.0

Sidebar object properties

The `sidebar` object properties:

Properties	Description	Type
disabled	If the sidebar is disabled	boolean
isVisible	If the sidebar is visible	boolean

Properties	Description	Type
status	Three status: 'SidebarComponent.STATUS_COLLAPSED', 'SidebarComponent.STATUS_COLLAPSED_TOTALLY', 'SidebarComponent.STATUS_EXPANDED'	string

The `sidebar-panel` object properties:

Properties	Description	Type
disabled	If the sidebar-panel is disabled	boolean
isVisible	If the sidebar-panel is visible	boolean
isActive	If the sidebar-panel is active	boolean

Methods

The `sidebar` methods:

Method	Description	Version
isCollapsed(): boolean	Return true if the status is not equal to <code>sidebarcomponent.status_expanded</code>	7.0.0
expand(width: number): void	Expand sidebar	7.0.0
collapse(): void	Hide the sidebar-panel with the sidebar buttons visible	7.0.0
collapseTotally(): void	Hide the sidebar with the sidebar buttons invisible	7.0.0

The `sidebar-panel` methods:

Method	Description	Version
disable(): void	Disable sidebar-panel. Once disabled, it cannot be activated	7.0
enable(): void	Enable sidebar-panel	7.0
show(): void	Show the hidden sidebar-panel	7.0
hide(): void	Hide the sidebar-panel	7.0

Method	Description	Version
destroy(): void	Destroy the sidebar-panel	7.0

Events

The **Sidebar** events:

Event Name	Description	Sample	Version
COMPONENT_EVENTS.EXPAND	Triggered when the sidebar is expanded	sidebar.on(COMPONENT_EVENTS.EXPAND, () => void)	7.0.0
COMPONENT_EVENTS.COLLAPSE	Triggered when the sidebar is collapsed	sidebar.on(COMPONENT_EVENTS.COLLAPSE, () => void)	7.0.0

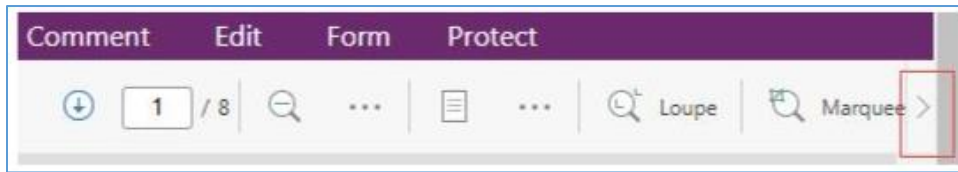
The **Sidebar-panel** events:

Event Name	Description	Sample	Version
active	Triggered when the sidebar-panel is activated	sidebarPanel.on('active', () => {})	7.0.0
deactive	Triggered when the sidebar-panel is deactivated	sidebarPanel.on('deactive', () => {})	7.0.0
shown	Triggered when the sidebar-panel is shown	sidebarPanel.on('shown', () => {})	7.0.0
hidden	Triggered when the sidebar-panel is hidden	sidebarPanel.on('hidden', () => {})	7.0.0

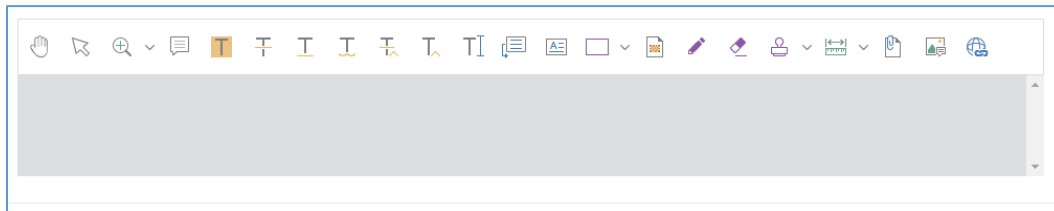
Paddle component

In a case of the toolbar length which goes beyond the screen width, some tools are hidden. Users have to scroll to show the hidden contents. The Paddle component serves in other way to display buttons at the ends of toolbar so that users can click to show the hidden tools instead of using scrollbar.

The arrow is the paddle button:



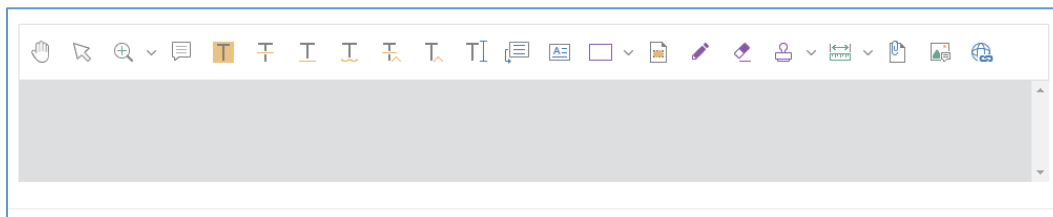
Code examples



```
<html>
  <template id="layout-template">
    <webpdf>
      <toolbar>
        <paddle>
          <div class="flex-div">
            <hand-button></hand-button>
            <selection-button></selection-button>
            <zoom-dropdown></zoom-dropdown>
            <create-note-button></create-note-button>
            <create-text-highlight-button></create-text-highlight-button>
            <create-strikeout-button></create-strikeout-button>
            <create-underline-button></create-underline-button>
            <create-squiggly-button></create-squiggly-button>
            <create-replace-button></create-replace-button>
            <create-caret-button></create-caret-button>
            <create-typewriter-button></create-typewriter-button>
            <create-callout-button></create-callout-button>
            <create-textbox-button></create-textbox-button>
            <create-drawings-dropdown></create-drawings-dropdown>
            <create-area-highlight-button></create-area-highlight-button>
            <create-pencil-button></create-pencil-button>
            <eraser-button></eraser-button>
            <stamp-dropdown></stamp-dropdown>
            <create-measure-dropdown></create-measure-dropdown>
            <create-attachment-button></create-attachment-button>
            <create-image-button></create-image-button>
            <create-link-button></create-link-button>
          </div>
        </paddle>
      </toolbar>
      <div class="fv_ui-body">
        <viewer></viewer>
      </div>
    </webpdf>
  </template>
</html>
```

```
</template>
</html>
<style>
.flex-div {
  display: flex;
}
.fv__ui-toolbar {
  border: 1px solid #ddd;
}
</style>
<script>
var CustomAppearance = UIExtension.appearances.Appearance.extend({
  getLayoutTemplate: function() {
    return document.getElementById('layout-template').innerHTML;
  },
  disableAll: function(){}
});
var libPath = window.top.location.origin + '/lib';
var pdfui = new UIExtension.PDFUI({
  viewerOptions: {
    libPath: libPath,
    jr: {
      licenseSN: licenseSN,
      licenseKey: licenseKey
    }
  },
  renderTo: document.body,
  appearance: CustomAppearance,
  addons: []
});
</script>
```

If you apply the same layout template for desktop and tablet, but only use drag functions on tablet instead of paddle, you can use `exclude-devices` to implement it:



```
<html>
<template id="layout-template">
  <webpdf>
    <toolbar>
      <!-- exclude all tablet devices -->
      <paddle exclude-devices="tablet">
        <div class="flex-div">
          <hand-button></hand-button>
```

```

        <selection-button></selection-button>
        <zoom-dropdown></zoom-dropdown>
        <create-note-button></create-note-button>
        <create-text-highlight-button></create-text-highlight-button>
        <create-strikeout-button></create-strikeout-button>
        <create-underline-button></create-underline-button>
        <create-squiggly-button></create-squiggly-button>
        <create-replace-button></create-replace-button>
        <create-caret-button></create-caret-button>
        <create-typewriter-button></create-typewriter-button>
        <create-callout-button></create-callout-button>
        <create-textbox-button></create-textbox-button>
        <create-drawings-dropdown></create-drawings-dropdown>
        <create-area-highlight-button></create-area-highlight-button>
        <create-pencil-button></create-pencil-button>
        <eraser-button></eraser-button>
        <stamp-dropdown></stamp-dropdown>
        <create-measure-dropdown></create-measure-dropdown>
        <create-attachment-button></create-attachment-button>
        <create-image-button></create-image-button>
        <create-link-button></create-link-button>
    </div>
</paddle>
</toolbar>
<div class="fv__ui-body">
    <viewer></viewer>
</div>
</webpdf>
</template>
</html>
<style>
    .flex-div {
        display: flex;
    }
    .fv__ui-toolbar {
        border: 1px solid #ddd;
    }
    /* use native scrollbar in tablet device */
    .fv__ui-tablet .fv__ui-toolbar {
        overflow-y: auto;
    }
</style>
<script>
    var CustomAppearance = UIExtension.appearances.Appearance.extend({
        getLayoutTemplate: function() {
            return document.getElementById('layout-template').innerHTML;
        },
        disableAll: function(){}
    });
    var libPath = window.top.location.origin + '/lib';
    var pdfui = new UIExtension.PDFUI({

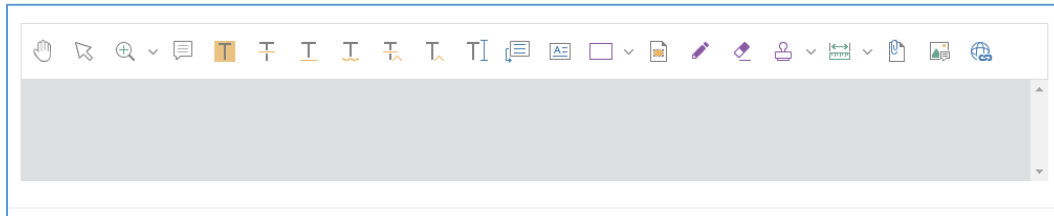
```

```
viewerOptions: {  
  libPath: libPath,  
  jr: {  
    licenseSN: licenseSN,  
    licenseKey: licenseKey  
  }  
},  
renderTo: document.body,  
appearance: CustomAppearance,  
addons: []  
});  
</script>
```

The device values that can be used include the following:

mac, ios, iphone, ipad, ipod, android, webos, kindle, tablet, mobile, desktop, xiaomi, huawei, touch.

By default, continuously clicking the paddle buttons at the ends of the toolbar moves the tool to the beginning or end. However, when the toolbar length is twice the screen width, the middle tools will never be displayed. To avoid this problem, you can set step in the paddle component.



```
<html>  
  <template id="layout-template">  
    <webpdf>  
      <toolbar>  
        <paddle step="200">  
          <div class="flex-div">  
            <hand-button></hand-button>  
            <selection-button></selection-button>  
            <zoom-dropdown></zoom-dropdown>  
            <create-note-button></create-note-button>  
            <create-text-highlight-button></create-text-highlight-button>  
            <create-strikeout-button></create-strikeout-button>  
            <create-underline-button></create-underline-button>  
            <create-squiggly-button></create-squiggly-button>  
            <create-replace-button></create-replace-button>  
            <create-caret-button></create-caret-button>  
            <create-typewriter-button></create-typewriter-button>  
            <create-callout-button></create-callout-button>  
            <create-textbox-button></create-textbox-button>  
            <create-drawings-dropdown></create-drawings-dropdown>  
          </div>  
        </paddle>  
      </toolbar>  
    </webpdf>  
  </template>  
</html>
```

```

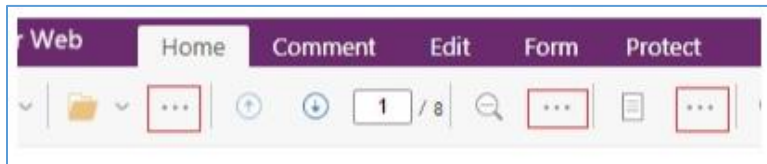
        <create-area-highlight-button></create-area-highlight-button>
        <create-pencil-button></create-pencil-button>
        <eraser-button></eraser-button>
        <stamp-dropdown></stamp-dropdown>
        <create-measure-dropdown></create-measure-dropdown>
        <create-attachment-button></create-attachment-button>
        <create-image-button></create-image-button>
        <create-link-button></create-link-button>
    </div>
</paddle>
</toolbar>
<div class="fv__ui-body">
    <viewer></viewer>
</div>
</webpdf>
</template>
</html>
<style>
.flex-div {
    display: flex;
}
.fv__ui-toolbar {
    border: 1px solid #ddd;
}
</style>
<script>
var CustomAppearance = UIExtension.appearances.Appearance.extend({
    getLayoutTemplate: function() {
        return document.getElementById('layout-template').innerHTML;
    },
    disableAll: function(){}
});
var libPath = window.top.location.origin + '/lib';
var pdfui = new UIExtension.PDFUI({
    viewerOptions: {
        libPath: libPath,
        jr: {
            licenseSN: licenseSN,
            licenseKey: licenseKey
        }
    },
    renderTo: document.body,
    appearance: CustomAppearance,
    addons: []
});
</script>

```

Group component

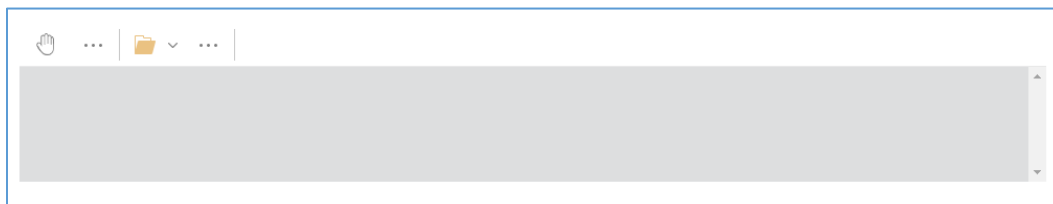
The Group component is commonly used on the Toolbar to separate the components with a vertical line. Each group represents a category, and can be set to shrink when running in a small screen.

Group shrank and hid tools under the dots:



Code example

Getting started



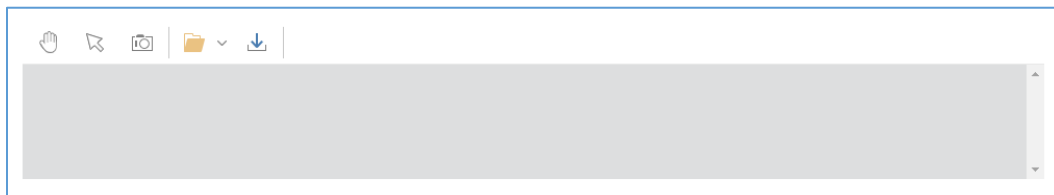
```
<html>
  <template id="layout-template">
    <webpdf>
      <toolbar>
        <group-list>
          <group name="home-tab-group-hand">
            <hand-button></hand-button>
            <selection-button>Selection</selection-button>
            <snapshot-button></snapshot-button>
          </group>
          <group name="home-tab-group-io">
            <open-file-dropdown></open-file-dropdown>
            <download-file-button></download-file-button>
          </group>
        </group-list>
      </toolbar>
      <div class="fv__ui-body">
        <viewer></viewer>
      </div>
    </webpdf>
  </template>
</html>
<style>
</style>
<script>
```



```
var CustomAppearance = UIExtension.appearances.Appearance.extend({
  getLayoutTemplate: function() {
    return document.getElementById('layout-template').innerHTML;
  },
  disableAll: function(){}
});
var libPath = window.top.location.origin + '/lib';
var pdfui = new UIExtension.PDFUI({
  viewerOptions: {
    libPath: libPath,
    jr: {
      licenseSN: licenseSN,
      licenseKey: licenseKey
    }
  },
  renderTo: document.body,
  appearance: CustomAppearance,
  addons: []
});
</script>
```

Designate the shrink-size

In the following example, the `group-list` defines `shrink-size` as 762 pixels, which means all child group with no `shrink-size` marker will contract when the width of the root component `<webpdf>` is smaller than 762 pixels. The child group with `shrink-size="600"` marker will shorten when the root component's width is less than 600 pixels.



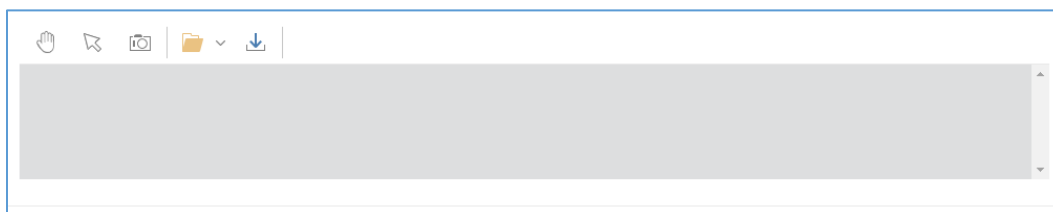
```
<html>
<template id="layout-template">
  <webpdf>
    <toolbar>
      <group-list shrink-size="762">
        <group name="home-tab-group-hand">
          <hand-button></hand-button>
          <selection-button>Selection</selection-button>
          <snapshot-button></snapshot-button>
        </group>
        <group name="home-tab-group-io" shrink-size="600">
          <open-file-dropdown></open-file-dropdown>
          <download-file-button></download-file-button>
        </group>
      </group-list>
```

```

        </toolbar>
        <div class="fv__ui-body">
            <viewer></viewer>
        </div>
    </webpdf>
</template>
</html>
<style>
</style>
<script>
    var CustomAppearance = UIExtension.appearances.Appearance.extend({
        getLayoutTemplate: function() {
            return document.getElementById('layout-template').innerHTML;
        },
        disableAll: function(){}
    });
    var libPath = window.top.location.origin + '/lib';
    var pdfui = new UIExtension.PDFUI({
        viewerOptions: {
            libPath: libPath,
            jr: {
                licenseSN: licenseSN,
                licenseKey: licenseKey
            }
        },
        renderTo: document.body,
        appearance: CustomAppearance,
        addons: []
    });
</script>

```

Designate the retained components after shrinkage



```

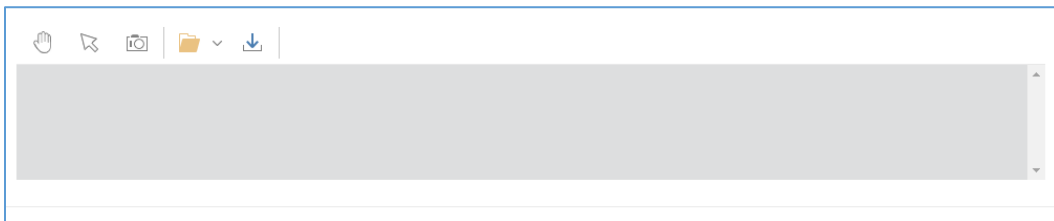
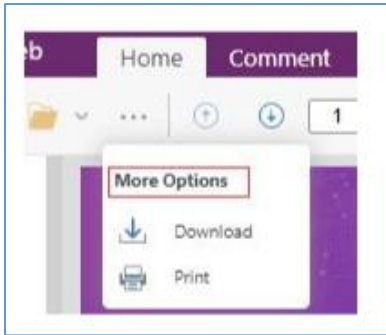
<html>
  <template id="layout-template">
    <webpdf>
      <toolbar>
        <group-list shrink-size="762">
          <!-- retain two components after shrinking -->
          Specify
          <group name="home-tab-group-hand" retain-count="2">
            <hand-button></hand-button>
            <selection-button>Selection</selection-button>
            <snapshot-button></snapshot-button>

```

```
        </group>
        <!-- If the retain-count value is equal to the components count, no shrinkag will occur -->
        <group name="home-tab-group-io" retain-count="2">
            <open-file-dropdown></open-file-dropdown>
            <download-file-button></download-file-button>
        </group>
    </group-list>
</toolbar>
<div class="fv__ui-body">
    <viewer></viewer>
</div>
</webpdf>
</template>
</html>
<style>
</style>
<script>
    var CustomAppearance = UIExtension.appearances.Appearance.extend({
        getLayoutTemplate: function() {
            return document.getElementById('layout-template').innerHTML;
        },
        disableAll: function(){}
    });
    var libPath = window.top.location.origin + '/lib';
    var pdfui = new UIExtension.PDFUI({
        viewerOptions: {
            libPath: libPath,
            jr: {
                licenseSN: licenseSN,
                licenseKey: licenseKey
            }
        },
        renderTo: document.body,
        appearance: CustomAppearance,
        addons: []
    });
</script>
```

Designate the shrink-title

The **More Options** in the image below is the shrink title:



```
<html>
  <template id="layout-template">
    <webpdf>
      <toolbar>
        <group-list shrink-size="762">
          <group name="home-tab-group-hand" retain-count="2" shrink-title="options">
            <hand-button></hand-button>
            <selection-button>Selection</selection-button>
            <snapshot-button></snapshot-button>
          </group>
          <group name="home-tab-group-io" shrink-title="options">
            <open-file-dropdown></open-file-dropdown>
            <download-file-button></download-file-button>
          </group>
        </group-list>
      </toolbar>
      <div class="fv__ui-body">
        <viewer></viewer>
      </div>
    </webpdf>
  </template>
</html>
<style>
</style>
<script>
  var CustomAppearance = UIExtension.appearances.Appearance.extend({
    getLayoutTemplate: function() {
      return document.getElementById('layout-template').innerHTML;
    },
    disableAll: function(){}
  });
  var libPath = window.top.location.origin + '/lib';
```

```
var pdfui = new UIExtension.PDFUI({
  viewerOptions: {
    libPath: libPath,
    jr: {
      licenseSN: licenseSN,
      licenseKey: licenseKey
    }
  },
  renderTo: document.body,
  appearance: CustomAppearance,
  addons: []
});
</script>
```

API

Group component template

Template example:

```
<group-list shrink-size="762">
  <group retain-count="2" shrink-title="options"></group>
  <group retain-count="1" shrink-title="options"></group>
</group-list>
```

The `group-list` template properties:

Property	Description	Type	Default Value	Version
shrink-size	Specify a pixel width. Triggered when the <code><webpdf></code> width is less than the specified width	number	1024	7.0.0

The `group` template properties:

Property	Description	Type	Default Value	Version
retain-count	Define the retained components after shrinkage	number	1	7.0.0
shrink-title	Define the title which shows on the top of the drop-down list after shrinkage	string	' '	7.0.0

Property	Description	Type	Default Value	Version
shrink-size	Specify a pixel width value. Triggered when the <code><webpdf></code> width is less than this value. Once specified, this group will ignore the value defined in the parent group-list	number	the same value in the parent component <code>group-list</code>	7.0.0

Methods

The `<group>` component methods:

Method	Description	Version
<code>setRetainCount(count: number): void</code>	Set the retained count after shrinkage	7.0.0
<code>setShrinkTitle(title: string): void</code>	Set the title which shows on the top of the drop-down list after shrinkage	7.0.0

Events

The `<group>` component events:

Name	Description	Example	Version
shrink	Triggered on shrink or expand	<code>group.on('shink', (isShrunked) => void)</code>	7.4.0

The `<group-list>` component events:

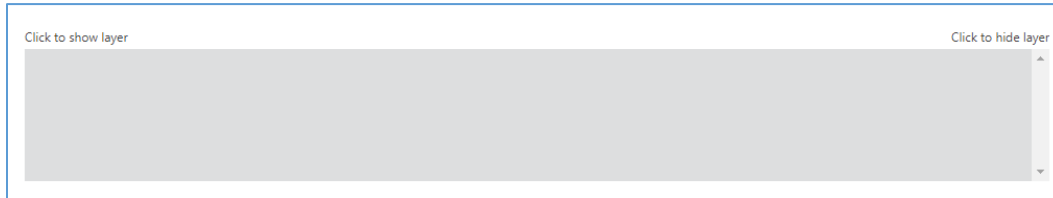
Name	Description	Example	Version
shrink	Triggered on shrink or expand	<code>groupList.on('shink', (groupComponent, isShrunked) => void)</code>	7.4.0

Layer component

Layer is a floating box component, which is typically used to implement dialogs, tooltips, right-click menus, and some other components that need to float on other elements.

Code examples

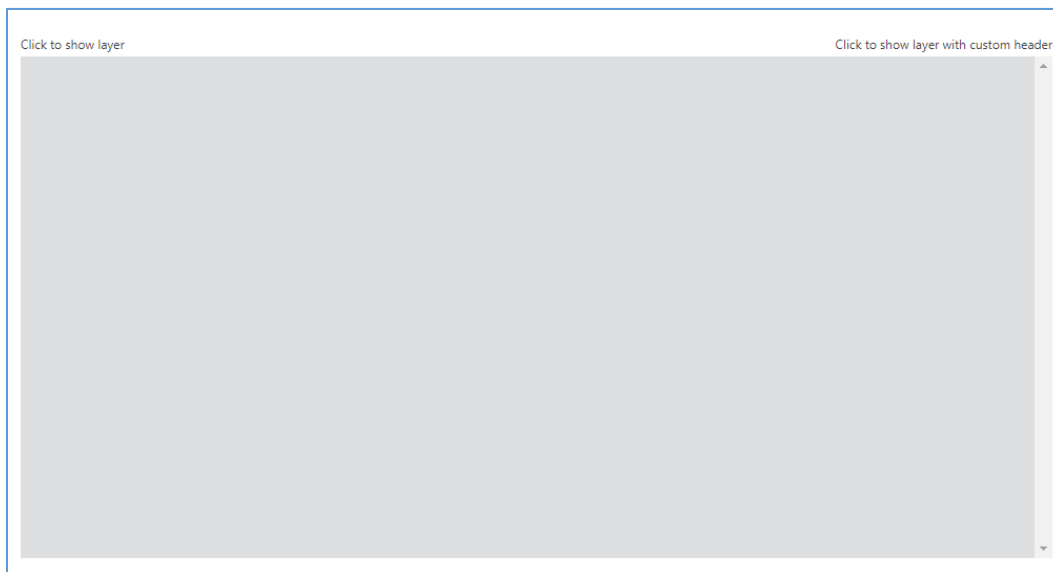
Getting started



```
<html>
  <template id="layout-template">
    <webpdf>
      <div class="flex-container">
        <xbutton action="show-layer" @controller="custom:ShowHideLayerController">Click to show
layer</xbutton>
        <xbutton action="hide-layer" @controller="custom:ShowHideLayerController">Click to hide
layer</xbutton>
      </div>
      <div class="fv__ui-body">
        <viewer></viewer>
      </div>
      <template>
        <layer name="my-layer" class="center">
          <text>Hello! I'm a layer component!</text>
        </layer>
      </template>
    </webpdf>
  </template>
</html>
<style>
  .flex-container {
    display: flex;
    justify-content: space-between;
  }
</style>
<script>
  UIExtension.PDFUI.module('custom', [])
    .controller('ShowHideLayerController', {
      handle: function() {
        const layer = this.getComponentByName('my-layer');
        const action = this.component.getAttribute('action');
        switch(action) {
          case 'show-layer':
            layer.show();
            break;
          case 'hide-layer':
            layer.hide();
            break;
        }
      }
    })
  }
```

```
    }  
  });  
  var CustomAppearance = UIExtension.appearances.Appearance.extend({  
    getLayoutTemplate: function() {  
      return document.getElementById('layout-template').innerHTML;  
    },  
    disableAll: function(){}  
  });  
  var libPath = window.top.location.origin + '/lib';  
  var pdfui = new UIExtension.PDFUI({  
    viewerOptions: {  
      libPath: libPath,  
      jr: {  
        licenseSN: licenseSN,  
        licenseKey: licenseKey  
      }  
    },  
    renderTo: document.body,  
    appearance: CustomAppearance,  
    addons: []  
  });  
</script>
```

Create a layer with header



```
<html>  
  <template id="layout-template">  
    <webpdf>  
      <div class="flex-container">  
        <xbutton target-layer="my-layer" @controller="custom:ShowLayerController">Click to show  
layer</xbutton>
```



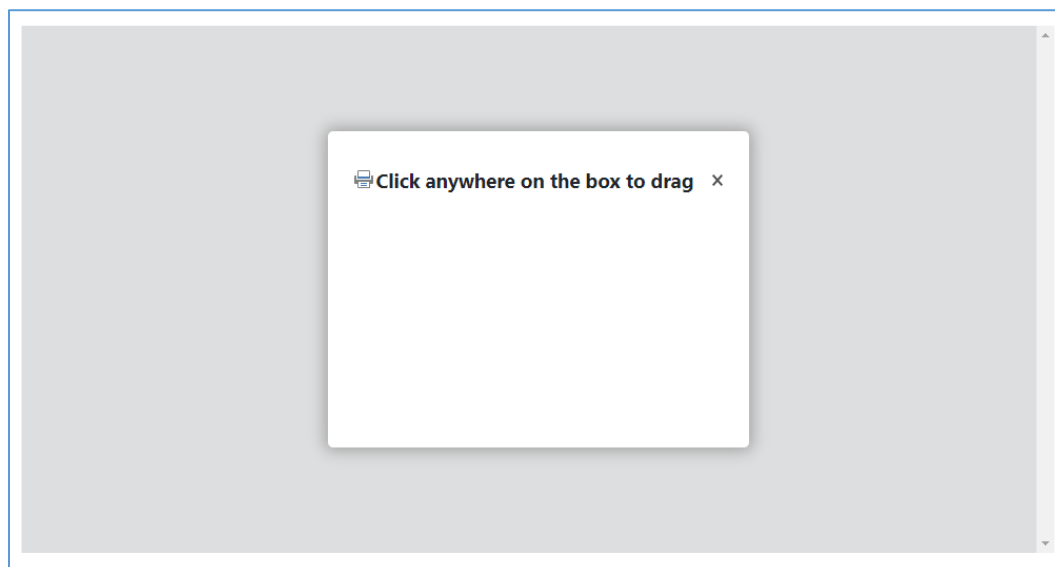
```

        <xbutton target-layer="my-layer-2" @controller="custom:ShowLayerController">Click to show layer
with custom header</xbutton>
    </div>
    <div class="fv__ui-body">
        <viewer></viewer>
    </div>
    <template>
        <layer name="my-layer" class="center my-layer">
            <layer-header title="Layer Title" icon-class="fv__icon-toolbar-print"></layer-header>
        </layer>
        <layer name="my-layer-2" class="center my-layer">
            <div class="my-custom-layer-header">
                <i class="fv__icon-toolbar-print"></i>
                <h2>Custom layer header</h2>
            </div>
        </layer>
    </template>
</webpdf>
</template>
</html>
<style>
.my-layer {
    width: 400px;
    height: 300px;
}
.my-custom-layer-header {
    display: flex;
    align-items: center;
}
.my-custom-layer-header i {
    display: inline-block;
    width: 32px;
    height: 32px;
}
.my-custom-layer-header h2 {
    flex: 1;
    margin: 0 0 0 1em;
}
.flex-container {
    display: flex;
    justify-content: space-between;
}
</style>
<script>
    UIExtension.PDFUI.module('custom', [])
        .controller('ShowLayerController', {
            handle: function() {
                const layerName = this.component.getAttribute('target-layer')
                const layer = this.getComponentByName(layerName);
                layer.show();
            }
        })

```

```
});  
var CustomAppearance = UIExtension.appearances.Appearance.extend({  
  getLayoutTemplate: function() {  
    return document.getElementById('layout-template').innerHTML;  
  },  
  disableAll: function(){}  
});  
var libPath = window.top.location.origin + '/lib';  
var pdfui = new UIExtension.PDFUI({  
  viewerOptions: {  
    libPath: libPath,  
    jr: {  
      licenseSN: licenseSN,  
      licenseKey: licenseKey  
    }  
  },  
  renderTo: document.body,  
  appearance: CustomAppearance,  
  addons: []  
});  
</script>
```

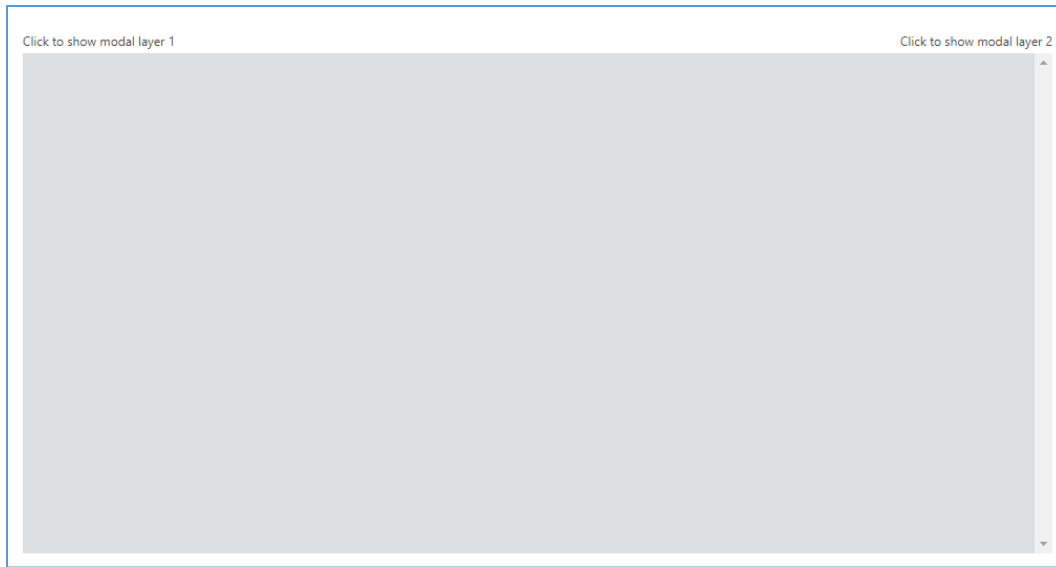
Create a draggable layer



```
<html>  
  <template id="layout-template">  
    <webpdf>  
      <div class="fv__ui-body">  
        <viewer></viewer>  
      </div>  
    </template>  
    <layer name="my-layer1" class="center my-layer" visible>
```

```
        <layer-header @draggable="{type:'parent'}" title="Click header area to drag" icon-class="fv__icon-  
toolbar-print"></layer-header>  
    </layer>  
    <layer name="my-layer2" class="center my-layer" @draggable visible>  
        <layer-header title="Click anywhere on the box to drag" icon-class="fv__icon-toolbar-print"></layer-  
header>  
        </layer>  
    </template>  
</webpdf>  
</template>  
</html>  
<style>  
    .my-layer {  
        width: 400px;  
        height: 300px;  
    }  
    .flex-container {  
        display: flex;  
        justify-content: space-between;  
    }  
</style>  
<script>  
    var CustomAppearance = UIExtension.appearances.Appearance.extend({  
        getLayoutTemplate: function() {  
            return document.getElementById('layout-template').innerHTML;  
        },  
        disableAll: function(){}  
    });  
    var libPath = window.top.location.origin + '/lib';  
    var pdfui = new UIExtension.PDFUI({  
        viewerOptions: {  
            libPath: libPath,  
            jr: {  
                licenseSN: licenseSN,  
                licenseKey: licenseKey  
            }  
        },  
        renderTo: document.body,  
        appearance: CustomAppearance,  
        addons: []  
    });  
</script>
```

Create a modal layer

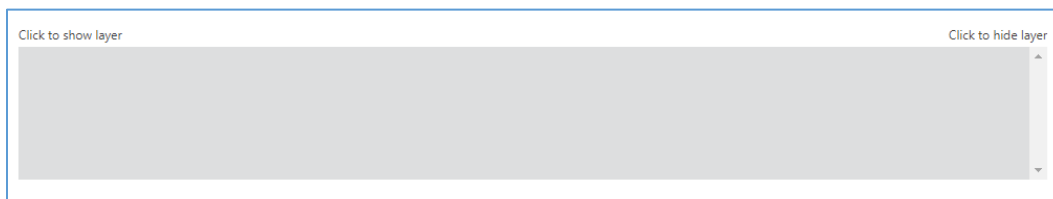


```
<html>
  <template id="layout-template">
    <webpdf>
      <div class="flex-container">
        <xbutton @controller="custom:ShowLayer1Controller">Click to show modal layer 1</xbutton>
        <xbutton @controller="custom:ShowLayer2Controller">Click to show modal layer 2</xbutton>
      </div>
      <div class="fv_ui-body">
        <viewer></viewer>
      </div>
      <template>
        <layer name="my-layer-1" class="center my-layer" modal backdrop>
          <layer-header title="Modal layer with backdrop" icon-class="fv__icon-toolbar-print"></layer-header>
        </layer>
        <layer name="my-layer-2" class="center my-layer" modal>
          <layer-header title="Modal layer without backdrop" icon-class="fv__icon-toolbar-print"></layer-
header>
        </layer>
      </template>
    </webpdf>
  </template>
</html>
<style>
  .my-layer {
    width: 400px;
    height: 300px;
  }
  .flex-container {
    display: flex;
    justify-content: space-between;
  }
</style>
<script>
```

```
UIExtension.PDFUI.module('custom', [])
  .controller('ShowLayer1Controller', {
    handle: function() {
      const layer = this.getComponentByName('my-layer-1');
      layer.show();
    }
  })
  .controller('ShowLayer2Controller', {
    handle: function() {
      const layer = this.getComponentByName('my-layer-2');
      layer.show();
    }
  });
var CustomAppearance = UIExtension.appearances.Appearance.extend({
  getLayoutTemplate: function() {
    return document.getElementById('layout-template').innerHTML;
  },
  disableAll: function(){}
});
var libPath = window.top.location.origin + '/lib';
var pdfui = new UIExtension.PDFUI({
  viewerOptions: {
    libPath: libPath,
    jr: {
      licenseSN: licenseSN,
      licenseKey: licenseKey
    }
  },
  renderTo: document.body,
  appearance: CustomAppearance,
  addons: []
});
</script>
```

Specify a parent node for the layer component

By default, the layer DOM nodes are appended to the end of the root component when the layer is displayed. This may cause the layer DOM hierarchy to display incorrectly in some cases. To avoid this problem, you can specify where to insert the layer DOM when calling `show()`. Here is the code example:



```
<html>
  <template id="layout-template">
```

```

<webpdf>
  <div class="flex-container">
    <xbutton action="show-layer" @controller="custom:ShowHideLayerController">Click to show
layer</xbutton>
    <xbutton action="hide-layer" @controller="custom:ShowHideLayerController">Click to hide
layer</xbutton>
  </div>
  <div class="fv__ui-body">
    <viewer></viewer>
  </div>
  <template>
    <layer name="my-layer" class="center">
      <text>Hello! I'm a layer component!</text>
    </layer>
  </template>
</webpdf>
</template>
</html>
<style>
.flex-container {
  display: flex;
  justify-content: space-between;
}
</style>
<script>
  UIExtension.PDFUI.module('custom', [])
    .controller('ShowHideLayerController', {
      handle: function() {
        const layer = this.getComponentByName('my-layer');
        const action = this.component.getAttribute('action');
        switch(action) {
          case 'show-layer':
            layer.show(document.body); // The layer will be appended to `document.body` when it is
displayed.
            break;
          case 'hide-layer':
            layer.hide();
            break;
        }
      }
    });
  var CustomAppearance = UIExtension.appearances.Appearance.extend({
    getLayoutTemplate: function() {
      return document.getElementById('layout-template').innerHTML;
    },
    disableAll: function(){}
  });
  var libPath = window.top.location.origin + '/lib';
  var pdfui = new UIExtension.PDFUI({
    viewerOptions: {
      libPath: libPath,

```

```

        jr: {
            licenseSN: licenseSN,
            licenseKey: licenseKey
        }
    },
    renderTo: document.body,
    appearance: CustomAppearance,
    addons: []
});
</script>

```

API

Layer component template

Template exmaple:

```

<layer class="center" visible modal backdrop>
  <layer-header title="" icon-class="fv__icon-toolbar-print"></layer-header>
</layer>

```

The `<layer>` component template properties:

Property	Description	Type	Default value	Version
visible	Whether make the layer visible	boolean	false	7.0.0
modal	Whether it is a modal box	boolean	false	7.0.0
backdrop	Whether the modal box uses a black translucent background	boolean	false	7.0.0
class="center"	Center layer	--	--	7.0.0
class="centerv"	Vertically center layer	--	--	7.0.0
class="centerh"	Horizontally center layer	--	--	7.0.0
class="left"	Show the layer on the left	--	--	7.0.0
class="right"	Show the layer on the right	--	--	7.0.0
class="top"	Show the layer on the top	--	--	7.0.0
class="bottom"	Show the layer on the bottom	--	--	7.0.0

The `<layer-header>` component template properties:

Property	Description	Type	Default value	Version
title	Title contents	string	"	7.0.0
icon-class	Title icon	string	"	7.0.0

Methods

Method	Description	Version
show(appendTo: HTMLElement): void	Append the layer components to a specified DOM node, and show.	7.0.0
open(appendTo: HTMLElement): void	Function same as show()	7.0.0
hide(): void	Hide layer	7.0.0
close(): void	Hide and destroy layer	7.0.0

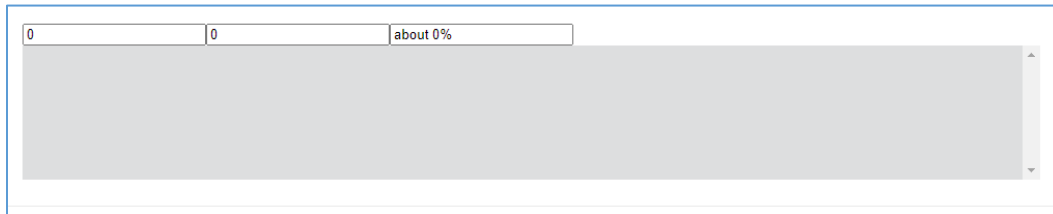
Events

Name	Description	Sample	Version
shown	Triggered after the layer displays	layer.on('shown', () => void)	7.0.0
hidden	Triggered after the layer is hidden	layer.on('hidden', () => void)	7.0.0
closed	Triggered after the layer is hidden and destroyed	layer.on('closed', () => void)	7.0.0

Number component

The number component is used for number inputs. Its features include the input number range, the step gradient, the display effects, etc.

Code example



```
<html>
  <template id="layout-template">
    <webpdf>
      <div class="flex-container">
        <number @tooltip tooltip-placement="right" tooltip-title="Any non numeric characters will be rejected"
min="0"></number>
        <number @tooltip tooltip-title="min=0,max=100,step=0.1" min="0" max="100" step="0.1"></number>
        <number @tooltip tooltip-title="use prefix and suffix" min="0" max="100" step="0.1" prefix="about "
suffix="%"></number>
      </div>
      <div class="fv__ui-body">
        <viewer></viewer>
      </div>
    </webpdf>
  </template>
</html>
<style>
  .flex-container {
    display: flex;
  }
</style>
<script>
  var CustomAppearance = UIExtension.appearances.Appearance.extend({
    getLayoutTemplate: function() {
      return document.getElementById('layout-template').innerHTML;
    },
    disableAll: function(){}
  });
  var libPath = window.top.location.origin + '/lib';
  var pdfui = new UIExtension.PDFUI({
    viewerOptions: {
      libPath: libPath,
      jr: {
        licenseSN: licenseSN,
        licenseKey: licenseKey
      }
    },
    renderTo: document.body,
    appearance: CustomAppearance,
    addons: []
  });
```

```
</script>
```

API

Number component template

```
<number min="0" max="100" step="0.1" prefix="about " suffix="%"></number>
```

Methods

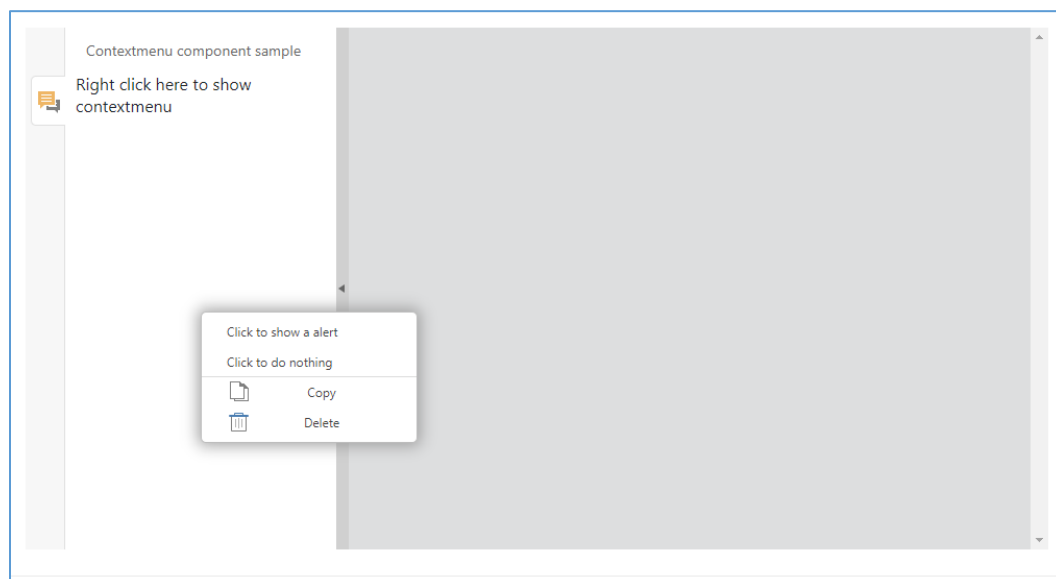
Method	Description	Version
setValue(value: number)	Set the value. If the value is not in the specified range and step, it will be automatically formatted. Calling this method won't trigger the change event.	7.1.0
getValue(): number	Get the current value	7.1.0

Events

Name	Description	Sample	Version
change	Triggered when the user enters a number and blurs focus	<code>number.on('change', (newValue, oldValue) => void)</code>	7.1.0

Contextmenu component

Code examples



```

<html>
  <template id="layout-template">
    <webpdf>
      <div class="fv__ui-body">
        <sidebar open>
          <sidebar-panel icon-class="fv__icon-sidebar-comment-list" @controller="custom:CustomController"
title="Contextmenu component sample">
            <text>Right click here to show contextmenu</text>
          </sidebar-panel>
        </sidebar>
        <viewer></viewer>
      </div>
    </template>
    <contextmenu name="fv--custom-contextmenu">
      <contextmenu-item @controller="custom:AlertDialogController">Click to show a
alert</contextmenu-item>
      <contextmenu-item>Click to do nothing</contextmenu-item>
      <contextmenu-separator></contextmenu-separator>
      <contextmenu-item icon-class="fv__icon-comment-item-menu-copy">Copy</contextmenu-item>
      <contextmenu-item icon-class="fv__icon-comment-item-menu-delete">Delete</contextmenu-item>
    </contextmenu>
  </template>
</webpdf>
</template>
</html>
<script>
  UIExtension.PDFUI.module('custom', [])
    .controller('AlertDialogController', {
      handle: function() {
        this.getPDFUI().alert('Hello World');
      }
    })
    .controller('CustomController', {
      mounted: function() {
        this.component.active();
        var element = this.component.getContainerElement();
        var contextmenu = this.getComponentByName('fv--custom-contextmenu');
        var rect = element.getBoundingClientRect();
        contextmenu.showAt(rect.left + rect.width/2, rect.top + rect.height / 2);
        element.addEventListener('contextmenu', function(e) {
          contextmenu.showAt(e.clientX, e.clientY);
          e.preventDefault();
        });
      }
    });
  var CustomAppearance = UIExtension.appearances.Appearance.extend({
    getLayoutTemplate: function() {
      return document.getElementById('layout-template').innerHTML;
    },
    disableAll: function(){}
  });

```

```
var libPath = window.top.location.origin + '/lib';
var pdfui = new UIExtension.PDFUI({
  viewerOptions: {
    libPath: libPath,
    jr: {
      licenseSN: licenseSN,
      licenseKey: licenseKey
    }
  },
  renderTo: document.body,
  appearance: CustomAppearance,
  addons: []
});
</script>
```

API

Contextmenu component template

Template exmaple:

```
<contextmenu name="fv--custom-contextmenu">
  <contextmenu-item >Click to show a alert</contextmenu-item>
  <contextmenu-item>Click to do nothing</contextmenu-item>
  <contextmenu-separator></contextmenu-separator>
  <contextmenu-item icon-class="fv__icon-comment-item-menu-copy">Copy</contextmenu-item>
  <contextmenu-item icon-class="fv__icon-comment-item-menu-delete">Delete</contextmenu-item>
</contextmenu>
```

Methods

Contextmenu methods

Method	Description	Version
showAt(x: number, y: number):void	Shows on the specified coordinates, where the x and y axes are relative to the browser viewport.	7.2.0

For more information, you may check [layer component](#).

Contextmenun item method:

For details, you may check [button component](#).

Events

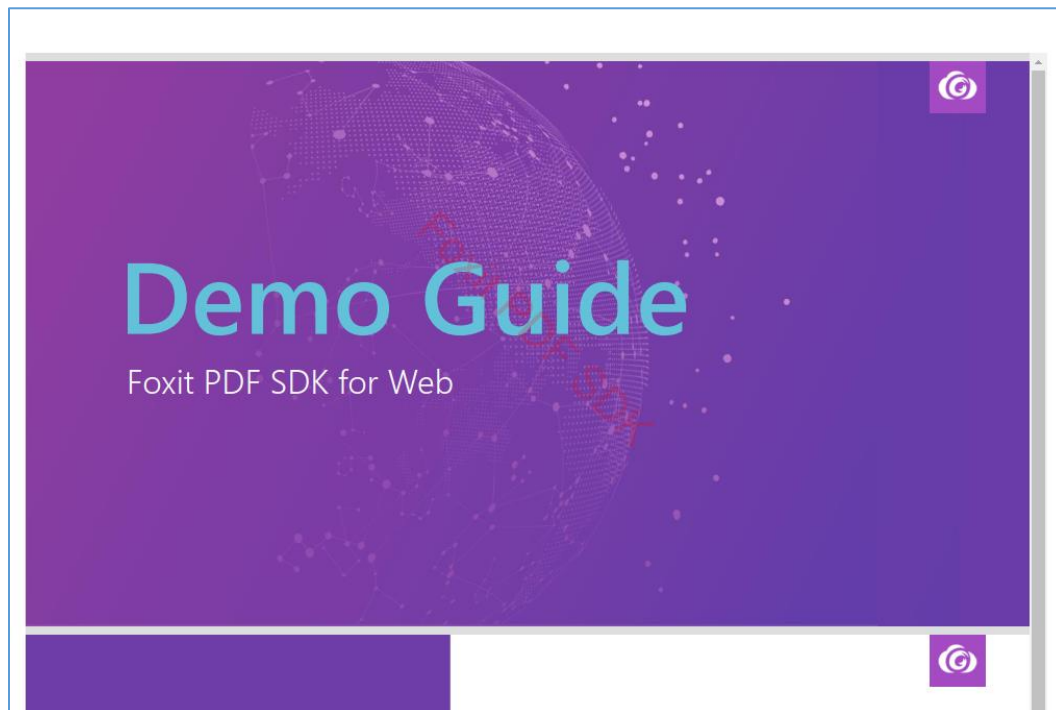
Contextmenu events are same as [layer component](#).

viewer component

The `<viewer>` component is used to render PDF. It is required in the layout template. Below is a basic layout template example:

```
<webpdf>
  <viewer></viewer>
</webpdf>
```

Runnable example:



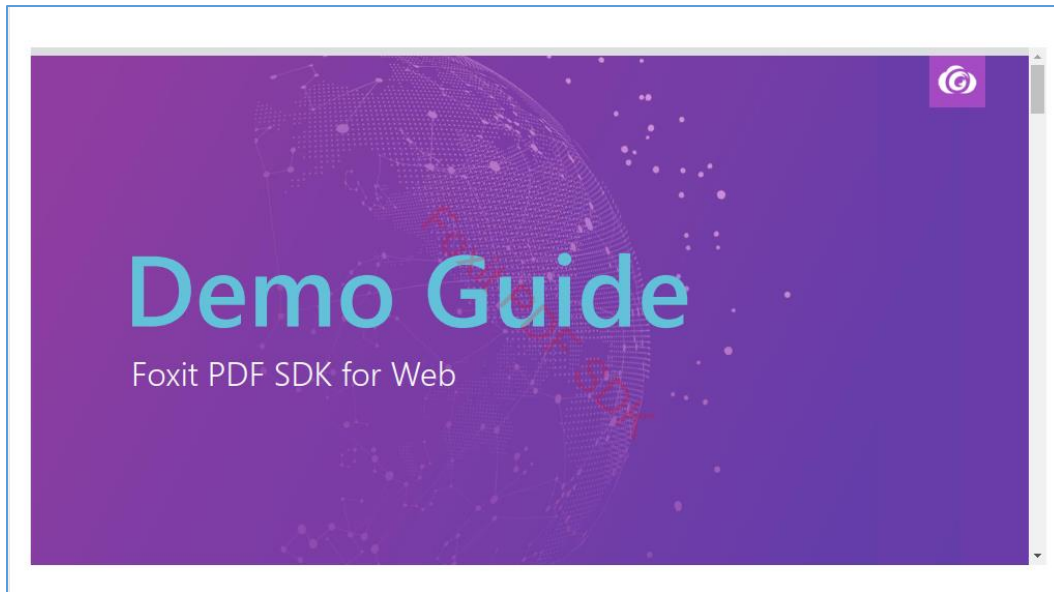
```
<html>
  <div id="pdf-ui"></div>
  <template id="layout-template">
    <webpdf>
      <viewer></viewer>
    </webpdf>
  </template>
</html>
<style>
  html{
    overflow:hidden;
  }
  body {
    height: 4180px;
  }
  #pdf-ui {
```

```
    position: relative;
    top: 50px;
  }
</style>
<script>
  var CustomAppearance = UIExtension.appearances.Appearance.extend({
    getLayoutTemplate: function() {
      return document.getElementById('layout-template').innerHTML;
    },
    disableAll: function(){}
  });
  var libPath = window.top.location.origin + '/lib';
  var pdfui = new UIExtension.PDFUI({
    viewerOptions: {
      libPath: libPath,
      jr: {
        licenseSN: licenseSN,
        licenseKey: licenseKey
      }
    },
    renderTo: '#pdf-ui',
    appearance: CustomAppearance,
    addons: []
  });

  var origin = window.top.location.origin;
  var url = origin + window.top.location.href.slice(origin.length).replace(/((\V.*?)?\Vdocs\V).*/,
'$1FoxitPDFSDKforWeb_DemoGuide.pdf');
  pdfui.openPDFByHttpRequest({
    range: {
      url: url,
    }
  }, { fileName: 'FoxitPDFSDKforWeb_DemoGuide.pdf' })

  window.addEventListener(UIExtension.PDFViewCtrl.DeviceInfo.isDesktop ? 'resize' : 'orientationchange',
function(e) {
  pdfui.redraw().catch(function(err) {console.log(err)});
});
</script>
```

By default, the PDFUI container has no size limit. In order to display the scroller, you should set the height for PDFUI based on your real viewer size.



```
<html>
  <div id="pdf-ui"></div>
  <template id="layout-template">
    <webpdf>
      <viewer></viewer>
    </webpdf>
  </template>
</html>
<style>
  html{
    overflow:hidden;
  }
  body {
    height: 4180px;
  }
  /* PDFUI container style */
  #pdf-ui {
    position: relative;
    top: 50px;
    height: 500px;
  }
</style>
<script>
  var CustomAppearance = UIExtension.appearances.Appearance.extend({
    getLayoutTemplate: function() {
      return document.getElementById('layout-template').innerHTML;
    },
    disableAll: function(){}
  });
  var libPath = window.top.location.origin + '/lib';
  var pdfui = new UIExtension.PDFUI({
    viewerOptions: {
```

```
libPath: libPath,
jr: {
  licenseSN: licenseSN,
  licenseKey: licenseKey
},
renderTo: '#pdf-ui',
appearance: CustomAppearance,
addons: []
});

var origin = window.top.location.origin;
var url = origin + window.top.location.href.slice(origin.length).replace(/((\V.*)?\Vdocs\V).*/,
'$1FoxitPDFSDKforWeb_DemoGuide.pdf');
pdfui.openPDFByHttpRequest({
  range: {
    url: url,
  }
}, { fileName: 'FoxitPDFSDKforWeb_DemoGuide.pdf' })

window.addEventListener(UIExtension.PDFViewCtrl.DeviceInfo.isDesktop ? 'resize' : 'orientationchange',
function(e) {
  pdfui.redraw().catch(function(err) {console.log(err)});
});
</script>
```

To achieve drag, zoom and scroll on a rendered PDF page, you should reference directives in your `<viewer>` component.

Usage snippet:

```
<viewer @zoom-on-pinch @zoom-on-doubletap @zoom-on-wheel @touch-to-scroll></viewer>
```

Directives:

Directive	Function
@zoom-on-pinch	Pinch to zoom
@zoom-on-doubletap	Double click to zoom
@zoom-on-wheel	Ctrl + mouse wheel to zoom
@touch-to-scroll	Drag to scroll

Business Components

Pre-configured component

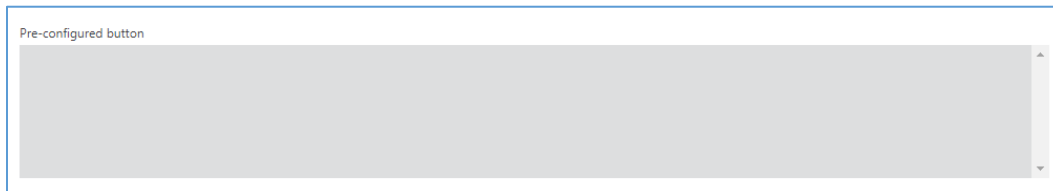
Pre-configured components are set in advance with text, icons, event handling and other information and assigning an alias, and then use alias in layout template directory. It is useful to simplify the template and reuse component in different appearance templates.

Custom pre-configured component

registerPreConfiguredComponent API

```
PDFUI.module('custom', [])
.registerPreConfiguredComponent('alias-button', {
  template: '<xbutton name="pre-configured-button"></xbutton>',
  config: [{
    target: 'pre-configured-button',
    text: 'Pre-configured button'
  }]
})
```

Runnable example:



```
<html>
  <template id="layout-template">
    <webpdf>
      <div>
        <custom:alias-button></custom:alias-button>
      </div>
      <div class="fv__ui-body">
        <viewer></viewer>
      </div>
    </webpdf>
  </template>
</html>
<script>
  UIExtension.PDFUI.module('custom', [])
    .registerPreConfiguredComponent('alias-button', {
      template: '<xbutton name="pre-configured-button">Pre-configured button</xbutton>',
      config: [{
        target: 'pre-configured-button',
        callback: function() {
```

```
        alert('click pre-configured button')
    }
}
});
var CustomAppearance = UIExtension.appearances.Appearance.extend({
    getLayoutTemplate: function() {
        return document.getElementById('layout-template').innerHTML;
    },
    disableAll: function(){}
});
var libPath = window.top.location.origin + '/lib';
var pdfui = new UIExtension.PDFUI({
    viewerOptions: {
        libPath: libPath,
        jr: {
            licenseSN: licenseSN,
            licenseKey: licenseKey
        }
    },
    renderTo: document.body,
    appearance: CustomAppearance,
    addons: []
});
</script>
```

Built-in pre-configured components

<hand-button>

Switch the state-handler to `STATE_HANDLER_HAND`

component usage:

```
<hand-button></hand-button>
```

Equivalent to:

```
<xbutton @tooltip tooltip-title="toolbar.tooltip.hand.title" name="hand-tool" icon-class="fv_icon-toolbar-hand"
@controller="states:HandController"></xbutton>
```

<selection-dropdown>

A dropdown with select-text-image button and select-annotation button.

component usage:

```
<selection-dropdown></selection-dropdown>
```

Equivalent to:

```
<dropdown @tooltip @controller="selection:SelectionDropdownController" name="selection-
dropdown" class="fv_ui-dropdown-hide-text">
```

```
<select-text-image-button></select-text-image-button>  
<select-annotation-button></select-annotation-button>  
</dropdown>
```

<select-text-image-button>

Switch the state-handler to `STATE_HANDLER_SELECT_TEXT_IMAGE`

component usage:

```
<select-text-image-button></select-text-image-button>
```

Equivalent to:

```
<xbutton @controller="states:SelectTextImageController" name="select-text-image" icon-class="fv_icon-  
toolbar-select-text-image">toolbar.buttons.selectTextImage</xbutton>
```

<select-annotation-button>

Switch the state-handler to `STATE_HANDLER_SELECT_ANNOTATION`

component usage:

```
<select-annotation-button></select-annotation-button>
```

Equivalent to:

```
<xbutton @controller="states:SelectAnnotationController" name="select-annotation" icon-class="fv_icon-  
toolbar-select-annotation">toolbar.buttons.selectAnnotation</xbutton>
```

<snapshot-button>

Switch the state-handler to `STATE_HANDLER_SNAPSHOT_TOOL`

component usage:

```
<snapshot-button></snapshot-button>
```

Equivalent to:

```
<xbutton @controller="states:SnapshotToolController" @tooltip tooltip-title="toolbar.buttons.snapshot"  
name="snapshot-button" icon-class="fv_icon-toolbar-snapshot">toolbar.buttons.snapshot</xbutton>
```

<change-color-dropdown>

A dropdown with colors to change background color of PDF viewer.

component usage:

```
<change-color-dropdown></change-color-dropdown>
```

Equivalent to:

```
<dropdown name="change-color-dropdown" @controller="change-color:ChangeColorController" @tooltip
tooltip-title="toolbar.tooltip.changeColor.title" icon-class="fv_icon-toolbar-change-color"
  popup-class="fv_ui-change-color-dropdown-popup" class="fv_ui-change-color-dropdown"
  separate="false">
  <div name="change-color-dropdown-list" class="fv_ui-change-color-dropdown-list">
    <div @foreach="color in colors"
      @sync.attr.class="fv_ui-change-color-dropdown-color-round ' + color.type"
      @sync.attr.style="color.type === 'moon' ? ' : ('background-color:' + color.background)"></div>
    </div>
  </dropdown>
```

<open-file-dropdown>

A dropdown with open-local-file button and open-url-file button.

Component usage:

```
<open-file-dropdown></open-file-dropdown>
```

Equivalent to:

```
<dropdown @controller="OpenFileDropdownController" name="open-file-button-list" class="fv_ui-dropdown-
hide-text" @cannotBeDisabled>
  <file-selector @controller="file:OpenLocalFileController" icon-class="fv_icon-toolbar-open" name="open-
local-file" accept=".pdf,.gif,.jpeg,.jpg,.png,.bmp" @cannotBeDisabled>toolbar.buttons.openfile</file-selector>
  <xbutton @controller="file:OpenRemoteFileController" icon-class="fv_icon-toolbar-open-url" name="open-
from-url" @cannotBeDisabled>toolbar.buttons.openFromUrl</xbutton>
</dropdown>
```

<download-file-button>

A button that clicks to download current opening PDF file.

Component usage:

```
<download-file-button></download-file-button>
```

Equivalent to:

```
<xbutton @tooltip tooltip-title="toolbar.buttons.download" name="download-file-button" icon-class="fv_icon-
toolbar-download" @controller="file:DownloadFileController">toolbar.buttons.download</xbutton>
```

<print:print-button> and <print:print-dialog>

The function of **<print:print-button>** is to click to display the **<print:print-dialog>**

These components are defined in print addon, before using it, you should add the print addon into addons:[]. For more details about addons, please refer to Introduction to addons.

Component usage

At first, define the `<print:print-button>` inside the toolbar or anywhere you need:

```
<print:print-button></print:print-button>
```

Its Equivalent to

```
<xbutton @tooltip tooltip-title="print:button-tooltip.title" name="print-button" icon-class="fv__icon-toolbar-print" @controller="print:ShowPrintDialogController">print:button-tooltip.title</xbutton>
```

At the second, define `<print:print-dialog>` inside a `<template>` tag:

```
<print:print-dialog></print:print-dialog>
```

This component is essential, because if this component is not defined, the user will not be able to see the print configuration dialog.

`<goto-prev-page-button>` and `<goto-next-page-button>`

These buttons are used to jump pages to previous or next.

Component usage

```
<goto-prev-page-button></goto-prev-page-button>  
<goto-next-page-button></goto-next-page-button>
```

Equivalent to:

```
<xbutton @tooltip tooltip-title="toolbar.tooltip.previousPage.title" icon-class="fv__icon-toolbar-prev-page" name="goto-prev-page" @controller="gotoview:GotoPrevPageController"></xbutton>  
<xbutton @tooltip tooltip-title="toolbar.tooltip.nextPage.title" icon-class="fv__icon-toolbar-next-page" name="goto-next-page" @controller="gotoview:GotoNextPageController"></xbutton>
```

`<goto-page-input>`

It is a component with input and text that displays the current page number and the total number of pages. It also allows you to enter page numbers and then press enter to jump to the page.

Component usage

```
<goto-page-input></goto-page-input>
```

Equivalent to:

```
<div class="fv__ui-toolbar-gotopage">  
  <number @controller="gotoview:GotoPageController" min="1" @bind.attr.max="pageNumber" @bind.value="currentPageIndex" name="gotopage-input" @on.change="onchange" @on.keydown="onkeydown"></number>  
  <span class="fv__ui-toolbar-gotopage-sep"></span>  
  <text @controller="gotoview:TotalPageTextController" @sync.text="pageNumber" name="gotopage-total">0</text>  
</div>
```

<zoom-out-button> and <zoom-in-button>

These two buttons are used to zoom in/out of the page.

Component usage

```
<zoom-out-button></zoom-out-button>
<zoom-in-button></zoom-in-button>
```

Equivalent to:

```
<xbutton @tooltip tooltip-title="toolbar.buttons.zoomout" @controller="zoom:ZoomInAndOutController"
action="zoomout" name="zoom-out" icon-class="fv_icon-toolbar-zoom-
out">toolbar.buttons.zoomout</xbutton>
<xbutton @tooltip tooltip-title="toolbar.buttons.zoomin" @controller="zoom:ZoomInAndOutController"
action="zoomin" name="zoom-in" icon-class="fv_icon-toolbar-zoom-in">toolbar.buttons.zoomin</xbutton>
```

<editable-zoom-dropdown>

This dropdown is used to zoom in/out on the page to specify the scale value.

Component usage

```
<editable-zoom-dropdown></editable-zoom-dropdown>
```

Its equivalent to

template:

```
<dropdown name="editable-zoom-dropdown" @controller="zoom:EditableZoomDropdownController"
class="fv_ui-editable_zoom_dropdown" editable="true">
  <dropdown-button icon-class="fv_icon-toolbar-fit-page" action="fitHeight"
@controller="zoom:EditableZoomActionController" name="editable-zoom-dropdown-
fitpage">toolbar.buttons.fitHeight</dropdown-button>
  <dropdown-button icon-class="fv_icon-toolbar-fit-width" action="fitWidth"
@controller="zoom:EditableZoomActionController" name="editable-zoom-dropdown-
fitwidth">toolbar.buttons.fitWidth</dropdown-button>
  <li class="fv_ui-dropdown-separator"></li>
  <dropdown-button @foreach="scale in $pdfui.customScalingValues" @sync.text="scale * 100 + '%"
@controller="zoom:EditableZoomToScaleValueController"></dropdown-button>
</dropdown>
```

config:

```
{
  target: 'editable-zoom-dropdown',
  editOptions: {
    type: 'number',
    min: 25,
    max: 600,
    step: 1,
  }
}
```

```
value: 50,  
template: '${value}%'  
}  
}
```

<zoom-dropdown>

This dropdown is used to zoom in/out on the pages like <editable-zoom-dropdown>, but it is un-editable.

Component usage

```
<zoom-dropdown></zoom-dropdown>
```

Equivalent to:

```
<dropdown name="dropdown-zoom" icon-class="fv__icon-toolbar-zoom-in" class="fv__ui-dropdown-hide-text" selected="0">  
  <dropdown-button name="dropdown-zoom-in" action="zoomin"  
@controller="zoom:DropdownZoomInAndOutController" icon-class="fv__icon-toolbar-zoom-in">toolbar.buttons.zoomin</dropdown-button>  
  <dropdown-button name="dropdown-zoom-out" action="zoomout"  
@controller="zoom:DropdownZoomInAndOutController" icon-class="fv__icon-toolbar-zoom-out">toolbar.buttons.zoomout</dropdown-button>  
  <dropdown-button name="dropdown-zoom-fitpage" action="fitHeight"  
@controller="zoom:ZoomActionController" icon-class="fv__icon-toolbar-fit-page">toolbar.buttons.fitHeight</dropdown-button>  
  <dropdown-button name="dropdown-zoom-fitwidth" action="fitWidth"  
@controller="zoom:ZoomActionController" icon-class="fv__icon-toolbar-fit-width">toolbar.buttons.fitWidth</dropdown-button>  
  <li class="fv__ui-dropdown-separator"></li>  
  <dropdown-button @foreach="scale in $pdfui.customScalingValues" @sync.text="scale * 100 + '%" @controller="zoom:ZoomToScaleValueController"></dropdown-button>  
</dropdown>
```

Page view mode buttons

Page view mode contains a series of buttons which are used to switch page view.

Component usage

```
<single-page-button></single-page-button>  
<continuous-page-button></continuous-page-button>  
<facing-page-button></facing-page-button>  
<continuous-facing-page-button></continuous-facing-page-button>  
<h-continuous:h-continuous-button></h-continuous:h-continuous-button>
```

Equivalent to

```
<xbutton @tooltip tooltip-title="toolbar.tools.single-page" @controller="pagemode:SinglePageModeController" name="single-page" icon-class="fv__icon-toolbar-single-page">toolbar.tools.single-page</xbutton>
```

```
<xbutton @tooltip tooltip-title="toolbar.tools.continuous-page"
@controller="pagemode:ContinuousPageModeController" name="continuous-page" icon-class="fv__icon-
toolbar-continuous-page">toolbar.tools.continuous-page</xbutton>
<xbutton @tooltip tooltip="toolbar.tools.facing" @controller="pagemode:FacingPageModeController"
name="facing-page" icon-class="fv__icon-toolbar-double-page">toolbar.tools.facing</xbutton>
<xbutton @tooltip tooltip="toolbar.tools.continuous-facing"
@controller="pagemode:ContinuousFacingPageModeController" name="continuous-facing-page" icon-
class="fv__icon-toolbar-facing-continuous-page">toolbar.tools.continuous-facing</xbutton>
<xbutton @tooltip tooltip-title="h-continuous:buttons.title" @controller="h-
continuous:HContinuousViewModeController" name="h-continuous-button" icon-class="fv__icon-toolbar-h-
continuous-page">h-continuous:buttons.title</xbutton>
```

`<h-continuous:h-continuous-button>` is defined in h-continuous addon, before using it, you should add the 'h-continuous' addon into addons list. For more details about addon, please refer to Introduction to addons.

`<loupe-tool-button>` and **`<loupe-tool-dialog>`**

A button which is used to switch state-handler to loupe tool when being clicked.

Component usage

```
<loupe-tool-button></loupe-tool-button>
<template>
  <loupe-tool-dialog></loupe-tool-dialog>
</template>
```

Equivalent to

```
<xbutton name="loupe-button" icon-class="fv__icon-toolbar-magnifier" class="fv__ui-toolbar-show-text-button"
@tooltip tooltip-title="toolbar.tooltip.loupe.title"
@controller="loupe:LoupeController">toolbar.tools.loupe</xbutton>
<template>
  <layer name="loupe-tool-dialog" class="fv__ui-loupe-tool-dialog" @resizable>
    <layer-header class="fv__ui-loupe-tool-header" title="loupe.title" @draggable="{type: 'parent'}"></layer-
header>
    <layer-view class="fv__ui-loupe-body">
      </layer-view>
    <layer-toolbar class="loupe-tool-bottom-bar" visible="false">
      <dropdown name="loupe-tool-zoom-dropdown" @controller="loupe:LoupeToolScaleListController"
editable="true">
        <dropdown-button @foreach="scaleItem in scaleList track by value"
@controller="loupe:LoupeToolScaleController" @bind.text="scaleItem.text"></dropdown-button>
      </dropdown>
      <slider name="loupe-tool-zoom-slider" min="50" max="600" step="1"></slider>
      <checkbox name="loupe-tool-lock">loupe.lockButton</checkbox>
    </layer-toolbar>
  </layer>
</template>
```


Note: The `<loupe-tool-button>` and `<loupe-tool-dialog>` should be defined in layout-template at the same time.

`<marquee-tool-button>`

A button which is used to switch the current state-handler to the marquee tool.

Component usage

```
<marquee-tool-button></marquee-tool-button>
```

Equivalent to

```
<xbutton name="marquee-button" icon-class="fv_icon-toolbar-marquee" class="fv_ui-toolbar-show-text-button" @tooltip tooltip-title="toolbar.tooltip.marquee.title"
@controller="marquee:MarqueeToolController">toolbar.buttons.marquee</xbutton>
```

`<fpmodule:contextmenu-item-file-property>`, `<fpmodule:file-property-button>` and `<fpmodule:file-property-dialog>`

`<fpmodule:contextmenu-item-file-property>` and `<fpmodule:file-property-button>` are used to display `<fpmodule:file-property-dialog>` when being clicked. All of these components are defined in the `file-property` addon. Before using it, you should add the `file-property` addon into `addons:[]`. For more details about addons, please refer to Introduction to addons.

Component usage `<fpmodule:file-property-dialog>`

```
<fpmodule:file-property-button></fpmodule:file-property-button>
<template>
  <fpmodule:file-property-dialog></fpmodule:file-property-dialog>
</template>
```

`<fpmodule:contextmenu-item-file-property>` must be used inside of `<contextmenu>`

Buttons to create annotations

The following components are built-in pre-configured components with their initial template which are used to switch current `state-handler` to `state-handlers` for annotation creation.

`<create-drawings-dropdown>`

```
<dropdown @controller="drawings:DrawingsDropdownController" name="create-shape-dropdown"
class="fv_ui-dropdown-hide-text">
  <create-square-button></create-square-button>
  <create-circle-button></create-circle-button>
  <create-line-button></create-line-button>
  <create-arrow-button></create-arrow-button>
  <create-polygon-button></create-polygon-button>
```

```
<create-polyline-button></create-polyline-button>  
<create-cloud-button></create-cloud-button>  
</dropdown>
```

```
<create-note-button>
```

```
<xbutton name="create-text" @tooltip tooltip-title="toolbar.tooltip.note.title"  
@controller="states:CreateTextController" icon-class="fv__icon-toolbar-note">toolbar.create.note</xbutton>
```

```
<create-text-highlight-button>
```

```
<xbutton name="create-highlight" @tooltip tooltip-title="toolbar.tooltip.highlight.title"  
@controller="states:CreateHighlightController" icon-class="fv__icon-toolbar-text-  
highlight">toolbar.create.highlight</xbutton>
```

```
<create-strikeout-button>
```

```
<xbutton name="create-strikeout" @tooltip tooltip="toolbar.tooltip.strikeout.title"  
@controller="states:CreateStrikeoutController" icon-class="fv__icon-toolbar-  
strikeout">toolbar.create.strikeout</xbutton>
```

```
<create-underline-button>
```

```
<xbutton name="create-underline" @tooltip tooltip-title="toolbar.tooltip.underline.title"  
@controller="states:CreateUnderlineController" icon-class="fv__icon-toolbar-  
underline">toolbar.create.underline</xbutton>
```

```
<create-squiggly-button>
```

```
<xbutton name="create-squiggly" @tooltip tooltip="toolbar.tooltip.squiggly.title"  
@controller="states:CreateSquigglyController" icon-class="fv__icon-toolbar-  
squiggly">toolbar.create.squiggly</xbutton>
```

```
<create-replace-button>
```

```
<xbutton name="create-replace" @tooltip tooltip="toolbar.tooltip.replace.title"  
@controller="states:CreateReplaceController" icon-class="fv__icon-toolbar-  
replace">toolbar.create.replace</xbutton>
```

```
<create-caret-button>
```

```
<xbutton name="create-caret" @tooltip tooltip-title="toolbar.tooltip.caret.title"  
@controller="states:CreateCaretController" icon-class="fv__icon-toolbar-insert">toolbar.create.caret</xbutton>
```

```
<create-typewriter-button>
```

```
<xbutton name="freetext-typewriter" @tooltip tooltip-title="toolbar.tooltip.typewriter.title"  
@controller="states:CreateTypewriterController" icon-class="fv__icon-toolbar-  
typewriter">toolbar.create.typewriter</xbutton>
```

```
<create-callout-button>
```

```
<xbutton name="freetext-callout" @tooltip tooltip-title="toolbar.tooltip.callout.title"
@controller="states:CreateCalloutController" icon-class="fv__icon-toolbar-
callout" >toolbar.create.callout</xbutton>
```

```
<create-textbox-button>
```

```
<xbutton name="freetext-textbox" @tooltip tooltip-title="toolbar.tooltip.textbox.title"
@controller="states:CreateTextboxController" icon-class="fv__icon-toolbar-
textbox" >toolbar.create.textbox</xbutton>
```

```
<create-area-highlight-button>
```

```
<xbutton name="create-area-highlight" @tooltip tooltip="toolbar.tooltip.areaHighlight.title"
@controller="states:CreateAreaHighlightController" icon-class="fv__icon-toolbar-area-
highlight">toolbar.create.areahighlight</xbutton>
```

```
<create-pencil-button>
```

```
<xbutton name="pencil-tool" @tooltip tooltip-title="toolbar.tooltip.pencil.title"
@controller="states:CreatePencilController" icon-class="fv__icon-toolbar-
pencil">toolbar.buttons.pencil</xbutton>
```

```
<eraser-button>
```

```
<xbutton name="eraser-tool" @tooltip tooltip-title="toolbar.tooltip.eraser.title"
@controller="states:EraserController" icon-class="fv__icon-toolbar-eraser">toolbar.buttons.eraser</xbutton>
```

```
<stamp-dropdown>
```

It's a dropdown component exhibits all stamp icons and a button for creating custom stamp.

```
<!-- internal implementation -->
```

```
<create-measurement-dropdown>
```

```
<dropdown name="create-measurement-button-list" class="fv__ui-dropdown-hide-text" @cannotBeDisabled
selected="0">
  <xbutton @tooltip tooltip-title="toolbar.buttons.tooltip.distance" name="create-distance-btn" icon-
class="fv__icon-toolbar-distance"
@controller="distance:CreateDistanceController">toolbar.buttons.distance</xbutton>
  <xbutton @hide-on-sr @tooltip tooltip-title="toolbar.buttons.tooltip.perimeter" name="create-perimeter-btn"
icon-class="fv__icon-toolbar-perimeter"
@controller="distance:CreatePerimeterController">toolbar.buttons.perimeter</xbutton>
  <xbutton @hide-on-sr @tooltip tooltip-title="toolbar.buttons.tooltip.area" name="create-area-btn" icon-
class="fv__icon-toolbar-area" @controller="distance:CreateAreaController">toolbar.buttons.area</xbutton>
  <xbutton @hide-on-sr @tooltip tooltip-title="toolbar.buttons.tooltip.circleArea" name="create-circle-area-btn"
icon-class="fv__icon-toolbar-areacircle"
@controller="distance:CreateCircleAreaController">toolbar.buttons.area</xbutton>
</dropdown>
```

```
<create-attachment-button>
```

```
<xbutton name="create-fileattachment" @tooltip tooltip-title="toolbar.tooltip.fileattachment.title"
@controller="states:CreateFileAttachmentController" icon-class="fv__icon-toolbar-
attachment">toolbar.create.fileattachment</xbutton>
```

```
<create-image-button>
```

```
<xbutton name="create-image" @tooltip tooltip-title="toolbar.tooltip.image.title"
@controller="states:CreateImageController" icon-class="fv__icon-toolbar-
image">toolbar.create.image</xbutton>
```

```
<create-link-button>
```

```
<xbutton name="create-link" @tooltip tooltip-title="toolbar.tooltip.link.title"
@controller="states:CreateLinkController" icon-class="fv__icon-toolbar-link" >toolbar.create.link</xbutton>
```

```
<multi-media:multi-media-button>
```

This component is defined in the `multi-media` addon. Before using it, you should add the `multi-media` addon into `addons:[]`. For more details about addons, please refer to Introduction to addons.

Graphics object components

Components defined in `edit-graphics` addon

1. `<edit-graphics:contextmenu-item-properties>`:

A `<contextmenu-item>` component IS used to show graphic object's properties dialog.

2. `<edit-graphics:contextmenu-item-delete>`:

A `<contextmenu-item>` component IS used to delete a graphic object.

3. `<edit-graphics:image-contextmenu>`:

A `<contextmenu>` component with properties & delete items is used for image graphics object.

Components defined in `path-object` addon

1. `<edit-pageobjects:path-contextmenu>`:

A `<contextmenu>` with properties & delete items is used for path graphic object.

2. `<edit-pageobjects:edit-all-objects-button>`:

A button which is used to switch current state-handler to 'edit-all' state-handler to modify currently supported graphic objects in PDF page.

3. `<edit-pageobjects:path-objects-dropdown>`:

A dropdown component which is used to create different types of path objects.

Components defined in `text-object` addon

1. `<edit-text-object:add-text-button>`:

A button component which is used to switch state-handler into add-text.

original template:

2. `<edit-text-object:text-bold-style-button>`:

A button to toggle the current editing text object to bold/thin style.

3. `<edit-text-object:text-italic-style-button>`:

A button to toggle the current editing text object to italic/normal style.

4. `<edit-text-object:text-underline-button>`:

A button to toggle the underline of current editing text object.

5. `<edit-text-object:font-color-picker>`:

A color-picker component which is used to set text color of current editing text object.

6. `<edit-text-object:font-style-dropdown>`:

A dropdown component which is used to set font style & size of the current editing text object.

PDF form components

The following components are defined in `import-form` and `export-form` addons.

1. `<import-form-module:import-form-button>`

A button which is used to select a form file to import form data.

2. `<export-form-module:export-form-dropdown>`

A button which is used to export form data as a xml file.

3. `<create-text-field-button>`

A button which is used to switch the current state-handler to `createTextField` to create text field.

4. `<create-signature-field-button>`

A button which is used to switch the current state-handler to `createSignatureField` to create signature field.

`<ink-sign-dropdown>` and `<create-ink-sign-dialog>`

These components are used to display all ink-signatures and create custom ink-signatures. They should be defined in the layout-template at the same time.

```
<ink-sign-dropdown></ink-sign-dropdown>
<template>
  <create-ink-sign-dialog></create-ink-sign-dialog>
</template>
```

`<password-protect:password-protect-button>`

This button is used to display the password protection dialog to encrypt current PDF document.

Usage

```
<password-protect:password-protect-button></password-protect:password-protect-button>
```

`<password-protect:remove-protect-button>`

This button is used to remove security of the current PDF document.

Usage

```
<password-protect:remove-protect-button></password-protect:remove-protect-button>
```

Redaction components

1. `<redaction:create-redactions-dropdown>`

A button to switch current state-handler to 'create-redaction-state'.

2. `<redaction:apply-redactions-button>`

A button to apply all redactions in PDF document.

3. `<redaction:redaction-search-button>`

A button to toggle search sidebar panel.

sidebar components

1. `<bookmark-sidebar-panel>`
2. `<commentlist-sidebar-panel>`
3. `<thumbnail-sidebar-panel>`
4. `<layer-sidebar-panel>`
5. `<search-sidebar-panel>`
6. `<attachment-sidebar-panel>`

`<distance:ruler-container>` and `<distance:measurement-popup>`

These two components show ruler and measurement information when creating distance annotation. The `<viewer>` component should be wrapped in `<distance:ruler-container>` with `<slot>` and `<distance:measurement-popup>` should be wrapped in a `<template>` tag.

```
<distance:ruler-container name="pdf-viewer-container-with-ruler">
  <slot>
    <viewer></viewer>
  </slot>
</distance:ruler-container>
<template>
  <distance:measurement-popup></distance:measurement-popup>
</template>
```

Right-click menu components

`<page-contextmenu>`

```
<contextmenu name="fv--page-contextmenu">
  <full-screen:contextmenu-item-fullscreen></full-screen:contextmenu-item-fullscreen>
  <contextmenu-separator @require-modules="full-screen" @hide-on-device="ios"></contextmenu-separator>
  <contextmenu-item-select-tool></contextmenu-item-select-tool>
  <contextmenu-item-hand-tool></contextmenu-item-hand-tool>
  <contextmenu-item-marquee-zoom></contextmenu-item-marquee-zoom>
  <contextmenu-separator></contextmenu-separator>

  <contextmenu-item-zoom-actual-size></contextmenu-item-zoom-actual-size>
  <contextmenu-item-zoom-fitpage></contextmenu-item-zoom-fitpage>
  <contextmenu-item-zoom-fitwidth></contextmenu-item-zoom-fitwidth>

  <contextmenu-separator></contextmenu-separator>
  <contextmenu-item-rotate-right></contextmenu-item-rotate-right>
  <contextmenu-item-rotate-left></contextmenu-item-rotate-left>
  <contextmenu-separator></contextmenu-separator>
  <print:contextmenu-item-print></print:contextmenu-item-print>
  <contextmenu-item-search></contextmenu-item-search>
  <contextmenu-separator @require-modules="fpmodule"></contextmenu-separator>
  <fpmodule:contextmenu-item-file-property></fpmodule:contextmenu-item-file-property>
</contextmenu>
```

`<default-annot-contextmenu>`

If the contextmenu for the annotation cannot be found, and that annotation is not a markup, this will be used by default.

```
<contextmenu name="fv--default-annot-contextmenu">
  <contextmenu-item-reply></contextmenu-item-reply>
  <contextmenu-separator></contextmenu-separator>
  <contextmenu-item-delete-annot></contextmenu-item-delete-annot>
  <contextmenu-separator></contextmenu-separator>
  <contextmenu-item-properties></contextmenu-item-properties>
</contextmenu>
```

```
<markup-contextmenu>
```

If the contextmenu for the annotation cannot be found, and that annotation is a markup, this will be used by default.

Original template:

```
<contextmenu name='fv--markup-contextmenu'>
  <contextmenu-item-reply></contextmenu-item-reply>
  <contextmenu-item-cut></contextmenu-item-cut>
  <contextmenu-separator></contextmenu-separator>
  <contextmenu-item-delete-annot></contextmenu-item-delete-annot>
  <contextmenu-separator></contextmenu-separator>
  <contextmenu-item-properties></contextmenu-item-properties>
</contextmenu>
```

Using

```
<markup-contextmenu name="fv--line-contextmenu"></markup-contextmenu>
<markup-contextmenu name="fv--lineararrow-contextmenu"></markup-contextmenu>
<markup-contextmenu name="fv--linedimension-contextmenu"></markup-contextmenu>
<markup-contextmenu name="fv--polylinedimention-contextmenu"></markup-contextmenu>
<markup-contextmenu name="fv--polygondimension-contextmenu"></markup-contextmenu>
<markup-contextmenu name="fv--circle-contextmenu"></markup-contextmenu>
<markup-contextmenu name="fv--square-contextmenu"></markup-contextmenu>
<markup-contextmenu name="fv--polyline-contextmenu"></markup-contextmenu>
<markup-contextmenu name="fv--polygon-contextmenu"></markup-contextmenu>
<markup-contextmenu name="fv--polygoncloud-contextmenu"></markup-contextmenu>
<markup-contextmenu name="fv--ink-contextmenu"></markup-contextmenu>
<markup-contextmenu name="fv--stamp-contextmenu"></markup-contextmenu>
<markup-contextmenu name="fv--text-contextmenu"></markup-contextmenu>
```

```
<caret-contextmenu>
```

Original template:

```
<contextmenu>
  <contextmenu-item-reply></contextmenu-item-reply>
  <contextmenu-separator></contextmenu-separator>
```



```
<contextmenu-item-delete-annot></contextmenu-item-delete-annot>
<contextmenu-separator></contextmenu-separator>
<contextmenu-item-properties></contextmenu-item-properties>
</contextmenu>
```

Usage:

```
<caret-contextmenu name="fv--areahighlight-contextmenu"></caret-contextmenu>
<caret-contextmenu name="fv--replace-contextmenu"></caret-contextmenu>
<caret-contextmenu name="fv--caret-contextmenu"></caret-contextmenu>
```

```
<textmarkup-contextmenu>
```

Original template:

```
<contextmenu name='fv--textmarkup-contextmenu'>
  <contextmenu-item-reply></contextmenu-item-reply>
  <contextmenu-separator></contextmenu-separator>
  <contextmenu-item-copy-text></contextmenu-item-copy-text>
  <contextmenu-item-delete-annot></contextmenu-item-delete-annot>
  <contextmenu-separator></contextmenu-separator>
  <contextmenu-item-properties></contextmenu-item-properties>
</contextmenu>
```

Usage:

```
<textmarkup-contextmenu name="fv--highlight-contextmenu"></textmarkup-contextmenu>
<textmarkup-contextmenu name="fv--strikeout-contextmenu"></textmarkup-contextmenu>
<textmarkup-contextmenu name="fv--underline-contextmenu"></textmarkup-contextmenu>
<textmarkup-contextmenu name="fv--squiggly-contextmenu"></textmarkup-contextmenu>
```

```
<freetext-contextmenu>
```

Original template:

```
<contextmenu name='fv--freetext-contextmenu'>
  <contextmenu-item-cut></contextmenu-item-cut>
  <contextmenu-item-delete-annot></contextmenu-item-delete-annot>
  <contextmenu-separator></contextmenu-separator>
  <contextmenu-item-properties></contextmenu-item-properties>
</contextmenu>
```

Usage:

```
<freetext-contextmenu name="fv--typewriter-contextmenu"></freetext-contextmenu>
<freetext-contextmenu name="fv--callout-contextmenu"></freetext-contextmenu>
<freetext-contextmenu name="fv--textbox-contextmenu"></freetext-contextmenu>
```

```
<action-annot-contextmenu>
```

Original template:

```
<contextmenu name="fv--action-annot-contextmenu">
  <contextmenu-item-delete-annot></contextmenu-item-delete-annot>
  <contextmenu-separator></contextmenu-separator>
  <contextmenu-item-annot-actions></contextmenu-item-annot-actions>
  <contextmenu-separator></contextmenu-separator>
  <contextmenu-item-properties></contextmenu-item-properties>
</contextmenu>
```

Usage:

```
<action-annot-contextmenu name="fv-image-contextmenu"></action-annot-contextmenu>
<action-annot-contextmenu name="fv-link-contextmenu"></action-annot-contextmenu>
```

```
<fileattachment-contextmenu>
```

Original template:

```
<contextmenu name="fv--fileattachment-contextmenu">
  <contextmenu-item-reply></contextmenu-item-reply>
  <contextmenu-separator></contextmenu-separator>
  <contextmenu-item-delete-annot></contextmenu-item-delete-annot>
  <contextmenu-separator></contextmenu-separator>
  <contextmenu-item-properties></contextmenu-item-properties>
</contextmenu>
```

```
<media-contextmenu>
```

Original template:

```
<contextmenu name="fv--media-contextmenu">
  <contextmenu-item-media-download></contextmenu-item-media-download>
  <contextmenu-separator></contextmenu-separator>
  <contextmenu-item-delete-annot></contextmenu-item-delete-annot>
  <contextmenu-separator></contextmenu-separator>
  <contextmenu-item-properties></contextmenu-item-properties>
  <contextmenu-separator></contextmenu-separator>
  <contextmenu-item-media-play></contextmenu-item-media-play>
</contextmenu>
```

```
<sound-contextmenu>
```

Original template:

```
<contextmenu name="fv--sound-contextmenu">
  <contextmenu-item-media-download></contextmenu-item-media-download>
  <contextmenu-separator></contextmenu-separator>
  <contextmenu-item-delete-annot></contextmenu-item-delete-annot>
  <contextmenu-separator></contextmenu-separator>
  <contextmenu-item-media-play></contextmenu-item-media-play>
</contextmenu>
```

```
<redact-contextmenu>
```

Original template:

```
<contextmenu name="fv--redact-contextmenu">
  <contextmenu-item-reply></contextmenu-item-reply>
  <contextmenu-separator></contextmenu-separator>
  <contextmenu-item-delete-annot></contextmenu-item-delete-annot>
  <contextmenu-separator></contextmenu-separator>
  <contextmenu-item-apply></contextmenu-item-apply>
  <contextmenu-item-apply-all></contextmenu-item-apply-all>
  <contextmenu-item-place></contextmenu-item-place>
  <contextmenu-separator></contextmenu-separator>
  <contextmenu-item-properties></contextmenu-item-properties>
</contextmenu>
```

```
<edit-graphics:image-contextmenu>
```

Original template:

```
<contextmenu name="fv--image-graphics-object-contextmenu">
  <edit-graphics:contextmenu-item-properties name="fv--contextmenu-item-image-graphics-object-
properties"></edit-graphics:contextmenu-item-properties>
  <edit-graphics:contextmenu-item-delete name="fv--contextmenu-item-image-graphics-object-delete"></edit-
graphics:contextmenu-item-delete>
</contextmenu>
```

```
<edit-pageobjects:path-contextmenu>
```

Original template:

```
<contextmenu name="fv--path-graphics-object-contextmenu">
  <edit-graphics:contextmenu-item-properties name="fv--contextmenu-item-path-graphics-object-
properties"></edit-graphics:contextmenu-item-properties>
  <edit-graphics:contextmenu-item-delete name="fv--contextmenu-item-path-graphics-object-delete"></edit-
graphics:contextmenu-item-delete>
</contextmenu>
```

```
<text-field-contextmenu>
```

Original template:

```
<contextmenu name='fv--text-field-contextmenu'>
  <contextmenu-item name="fv--contextmenu-item-form-sign" @controller="annot-
opr:SignController">certifySign.sign</contextmenu-item>
  <contextmenu-item-form-properties></contextmenu-item-form-properties>
  <contextmenu-separator></contextmenu-separator>
  <contextmenu-item-delete-annot></contextmenu-item-delete-annot>
</contextmenu>
```

Note: The name of the contextmenu is used internally to obtain the menu components and cannot be changed!

`<text-sel:text-selection-tooltip>`

A floating component that appears when you select text in a PDF page.

`<annottext>`

A floating component that appears when mouse over an annotation.

Directives

@controller

The `@controller` directive attaches a controller class to the component. The controller class must be registered in a module.

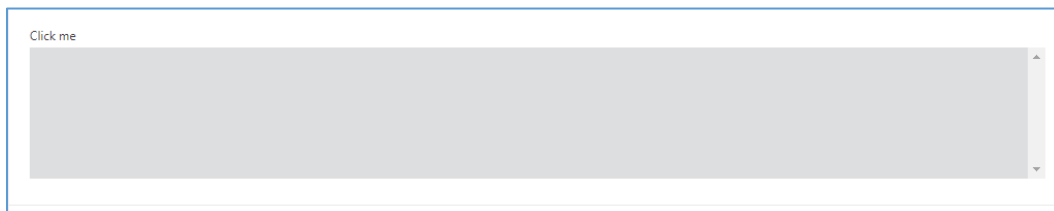
Usage

Syntax

```
<component-name @controller="module-name:ControllerName">
```

Example

This example demonstrates the directive syntax.



```
<html>
  <template id="layout-template">
    <webpdf>
      <div>
        <xbutton @controller="custom:MyController">Click me</xbutton>
      </div>
      <div class="fv__ui-body">
        <viewer></viewer>
      </div>
    </webpdf>
  </template>
</html>
<script>
  UIExtension.PDFUI.module('custom', [])
    .controller('MyController', {
      handle: function() {
        alert('Button click!');
      }
    });
</script>
```

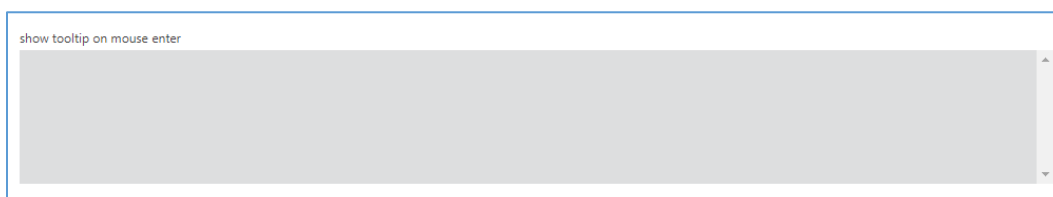
```
var CustomAppearance = UIExtension.appearances.Appearance.extend({
  getLayoutTemplate: function() {
    return document.getElementById('layout-template').innerHTML;
  },
  disableAll: function(){}
});
var libPath = window.top.location.origin + '/lib';
var pdfui = new UIExtension.PDFUI({
  viewerOptions: {
    libPath: libPath,
    jr: {
      licenseSN: licenseSN,
      licenseKey: licenseKey
    }
  },
  renderTo: document.body,
  appearance: CustomAppearance,
  addons: []
});
</script>
```

@tooltip

This is a simple text pop-up tip which is shown on mouse enter and hidden on mouse leave. This tooltip doesn't support complex text and operations.

Example

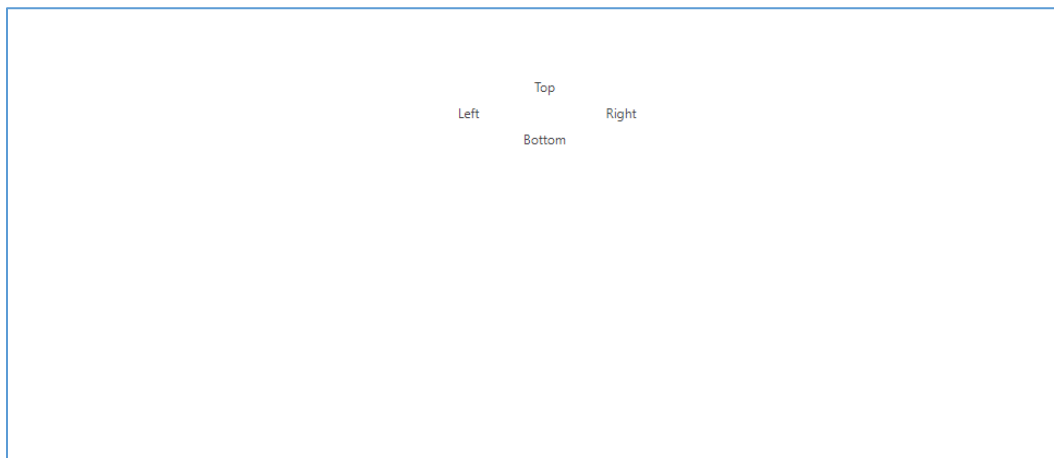
Show tooltip on mouse enter



```
<html>
  <template id="layout-template">
    <webpdf>
      <div>
        <xbutton @tooltip tooltip-title="Tooltip title">show tooltip on mouse enter</xbutton>
      </div>
      <div class="fv__ui-body">
        <viewer></viewer>
      </div>
    </webpdf>
  </template>
</html>
```

```
<script>
var CustomAppearance = UIExtension.appearances.Appearance.extend({
  getLayoutTemplate: function() {
    return document.getElementById('layout-template').innerHTML;
  },
  disableAll: function(){}
});
var libPath = window.top.location.origin + '/lib';
var pdfui = new UIExtension.PDFUI({
  viewerOptions: {
    libPath: libPath,
    jr: {
      licenseSN: licenseSN,
      licenseKey: licenseKey
    }
  },
  renderTo: document.body,
  appearance: CustomAppearance,
  addons: []
});
</script>
```

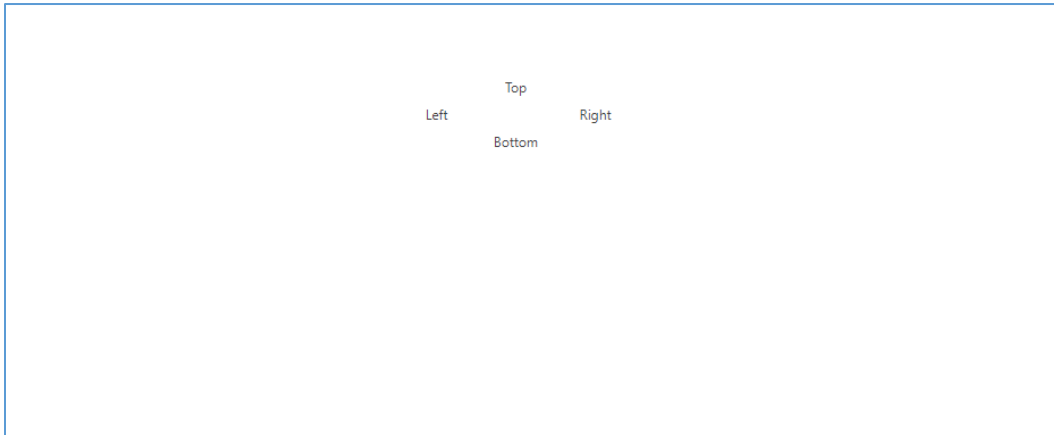
Tooltip placement



```
<html>
<template id="layout-template">
  <webpdf>
    <div class="vertical">
      <xbutton @tooltip tooltip-title="pop text" tooltip-placement="top">Top</xbutton>
      <div class="left-right">
        <xbutton @tooltip tooltip-title="pop text" tooltip-placement="left">Left</xbutton>
        <xbutton @tooltip tooltip-title="pop text" tooltip-placement="right">Right</xbutton>
      </div>
      <xbutton @tooltip tooltip-title="pop text" tooltip-placement="bottom">Bottom</xbutton>
    </div>
    <div class="fv__ui-body">
```

```
        </viewer></viewer>
    </div>
</webpdf>
</template>
</html>
<style>
    .vertical {
        display: flex;
        flex-direction: column;
        align-items: center;
        width: 200px;
        margin: 50px auto;
    }
    .vertical .fv__ui-button {
        width: 60px;
        justify-content: center;
    }
    .left-right {
        display: flex;
        flex-direction: row;
        justify-content: space-between;
        width: 100%;
    }
    .fv__ui-body {
        display: none;
    }
</style>
<script>
    var CustomAppearance = UIExtension.appearances.Appearance.extend({
        getLayoutTemplate: function() {
            return document.getElementById('layout-template').innerHTML;
        },
        disableAll: function(){}
    });
    var libPath = window.top.location.origin + '/lib';
    var pdfui = new UIExtension.PDFUI({
        viewerOptions: {
            libPath: libPath,
            jr: {
                licenseSN: licenseSN,
                licenseKey: licenseKey
            }
        },
        renderTo: document.body,
        appearance: CustomAppearance,
        addons: []
    });
</script>
```

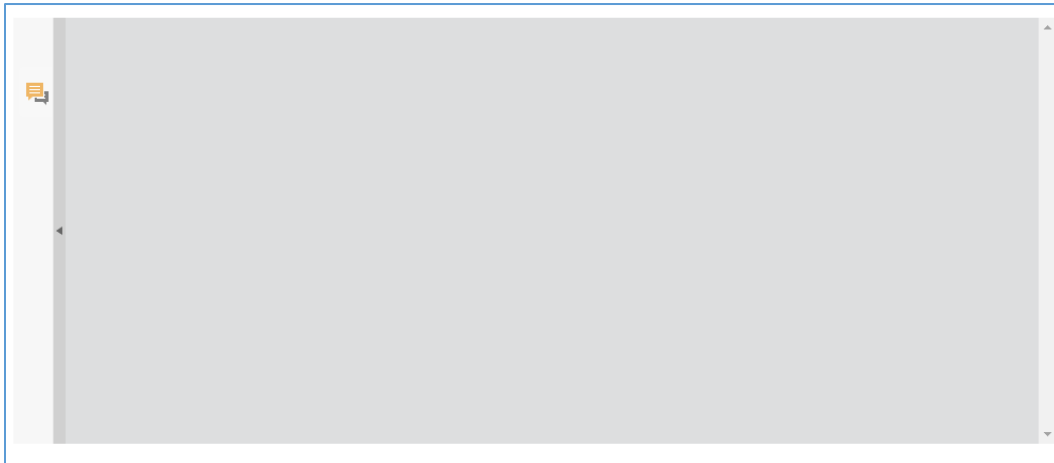
Fragment configuration



```
<html>
  <template id="layout-template">
    <webpdf>
      <div class="vertical">
        <xbutton name="top-button" @tooltip tooltip-title="pop text" tooltip-placement="top">Top</xbutton>
        <div class="left-right">
          <xbutton name="left-button" @tooltip tooltip-title="pop text" tooltip-
placement="left">Left</xbutton>
          <xbutton name="right-button" @tooltip tooltip-title="pop text" tooltip-
placement="right">Right</xbutton>
        </div>
        <xbutton name="bottom-button" @tooltip tooltip-title="pop text" tooltip-
placement="bottom">Bottom</xbutton>
      </div>
      <div class="fv__ui-body">
        <viewer></viewer>
      </div>
    </webpdf>
  </template>
</html>
<style>
  .vertical {
    display: flex;
    flex-direction: column;
    align-items: center;
    width: 200px;
    margin: 50px auto;
  }
  .vertical .fv__ui-button {
    width: 60px;
    justify-content: center;
  }
  .left-right {
    display: flex;
    flex-direction: row;
    justify-content: space-between;
  }
```

```
width: 100%;
}
.fv__ui-body {
display: none;
}
</style>
<script>
var CustomAppearance = UIExtension.appearances.Appearance.extend({
  getLayoutTemplate: function() {
    return document.getElementById('layout-template').innerHTML;
  },
  getDefaultFragments: function() {
    return [{
      target: 'left-button',
      config: {
        tooltip: {
          title: 'fragment config'
        }
      }
    }]
  },
  disableAll: function(){}
});
var libPath = window.top.location.origin + '/lib';
var pdfui = new UIExtension.PDFUI({
  viewerOptions: {
    libPath: libPath,
    jr: {
      licenseSN: licenseSN,
      licenseKey: licenseKey
    }
  },
  renderTo: document.body,
  appearance: CustomAppearance,
  addons: []
});
</script>
```

Tooltip on sidebar



```
<html>
  <template id="layout-template">
    <webpdf>
      <div class="fv__ui-body">
        <sidebar>
          <!-- tooltip-anchor: specify an element to display tip -->
          <sidebar-panel
            @tooltip
            tooltip-title="Tooltip text"
            tooltip-placement="right"
            tooltip-anchor=".fv__ui-sidebar-nav-ctrl"
            icon-class="fv__icon-sidebar-comment-list"
          ></sidebar-panel>
        </sidebar>
        <viewer></viewer>
      </div>
    </webpdf>
  </template>
</html>
<script>
  var CustomAppearance = UIExtension.appearances.Appearance.extend({
    getLayoutTemplate: function() {
      return document.getElementById('layout-template').innerHTML;
    },
    disableAll: function(){}
  });
  var libPath = window.top.location.origin + '/lib';
  var pdfui = new UIExtension.PDFUI({
    viewerOptions: {
      libPath: libPath,
      jr: {
        licenseSN: licenseSN,
        licenseKey: licenseKey
      }
    },
    renderTo: document.body,
```

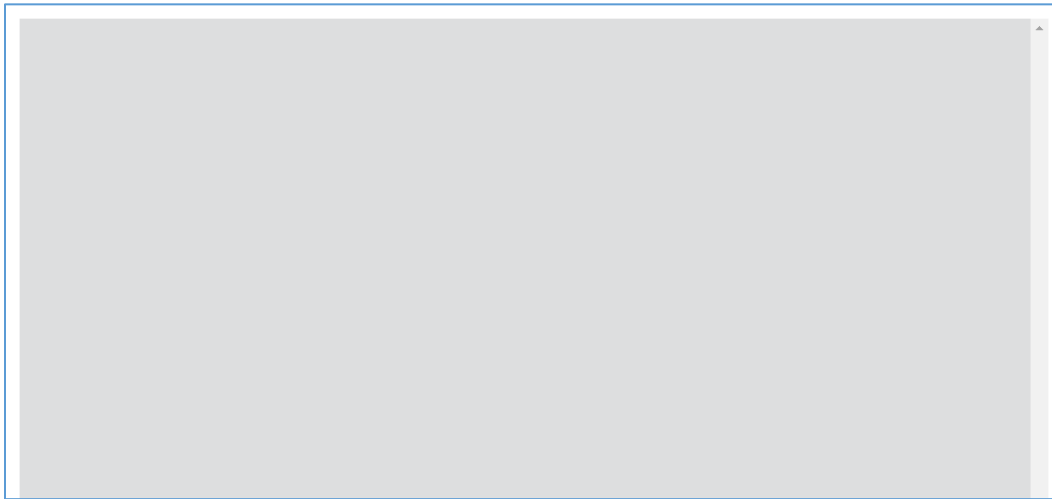
```
    appearance: CustomAppearance,  
    addons: []  
  });  
</script>
```

@draggable

The draggable directive implements a common drag-and-drop function which is commonly used in dialog components.

Example

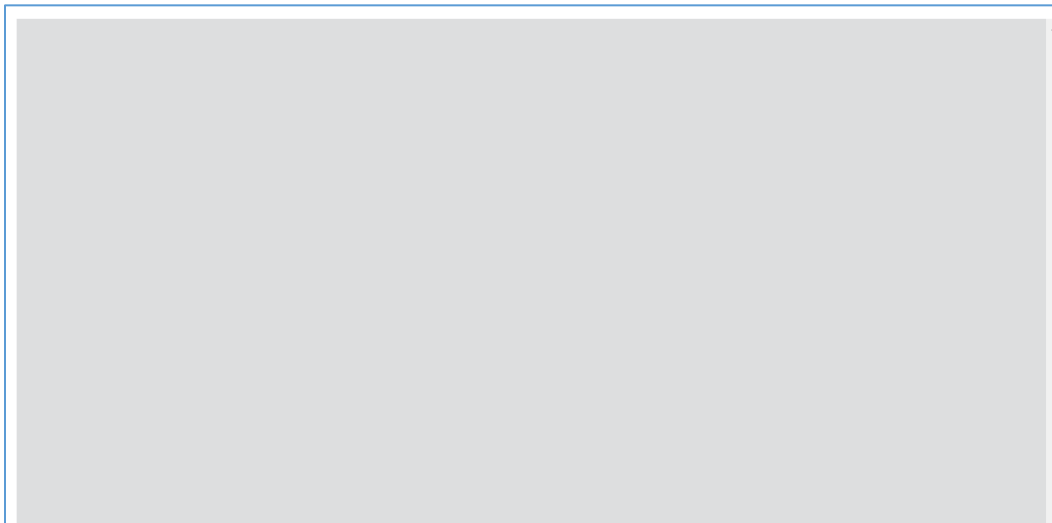
Draggable dialog



```
<html>  
  <template id="layout-template">  
    <webpdf>  
      <div class="fv__ui-body">  
        <viewer></viewer>  
      </div>  
    </template>  
    <layer name="my-layer2" class="center my-layer" @draggable visible>  
      <layer-header title="Click anywhere on the box to drag" icon-class="fv__icon-toolbar-print"></layer-  
header>  
      </layer>  
    </template>  
  </webpdf>  
</template>  
</html>  
<style>  
  .my-layer {  
    width: 400px;  
    height: 300px;
```

```
}  
.flex-container {  
  display: flex;  
  justify-content: space-between;  
}  
</style>  
<script>  
  var CustomAppearance = UIExtension.appearances.Appearance.extend({  
    getLayoutTemplate: function() {  
      return document.getElementById('layout-template').innerHTML;  
    },  
    disableAll: function(){}  
  });  
  var libPath = window.top.location.origin + '/lib';  
  var pdfui = new UIExtension.PDFUI({  
    viewerOptions: {  
      libPath: libPath,  
      jr: {  
        licenseSN: licenseSN,  
        licenseKey: licenseKey  
      }  
    },  
    renderTo: document.body,  
    appearance: CustomAppearance,  
    addons: []  
  });  
</script>
```

Draggable dialog header



```
<html>  
  <template id="layout-template">  
    <webpdf>  
      <div class="fv__ui-body">
```

```

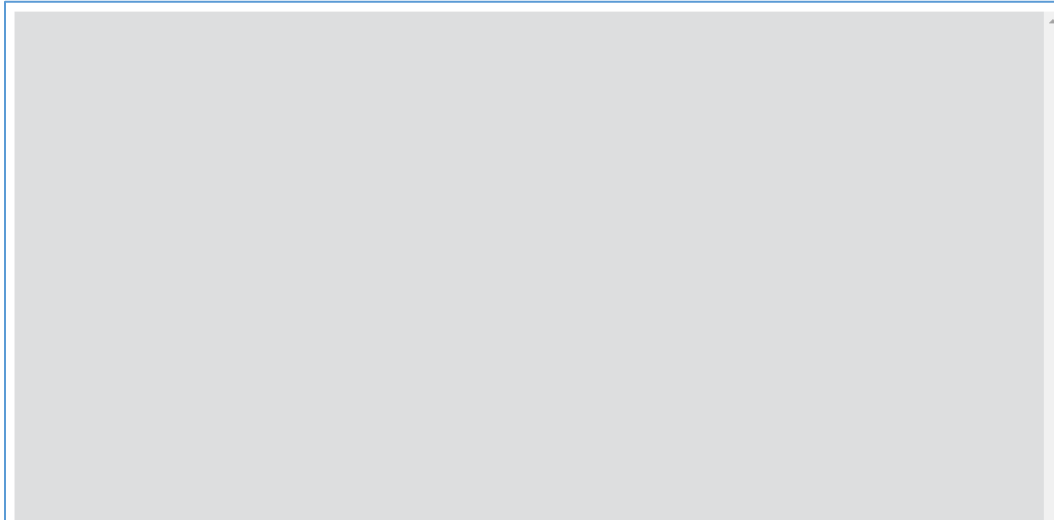
        <viewer></viewer>
    </div>
    <template>
        <layer name="my-layer1" class="center my-layer" visible>
            <layer-header @draggable="{type:'parent'}" title="Click header area to drag" icon-class="fv__icon-
toolbar-print"></layer-header>
        </layer>
    </template>
</webpdf>
</template>
</html>
<style>
    .my-layer {
        width: 400px;
        height: 300px;
    }
    .flex-container {
        display: flex;
        justify-content: space-between;
    }
</style>
<script>
    var CustomAppearance = UIExtension.appearances.Appearance.extend({
        getLayoutTemplate: function() {
            return document.getElementById('layout-template').innerHTML;
        },
        disableAll: function(){}
    });
    var libPath = window.top.location.origin + '/lib';
    var pdfui = new UIExtension.PDFUI({
        viewerOptions: {
            libPath: libPath,
            jr: {
                licenseSN: licenseSN,
                licenseKey: licenseKey
            }
        },
        renderTo: document.body,
        appearance: CustomAppearance,
        addons: []
    });
</script>

```

Non-draggable area

Sometimes, a draggable dialog box may contain components that have their own drag function (e.g., slider). At this point, if the drag function does not stop dragging events from propagating to the outer layer, the overall interaction will be affected. To solve this problem, you can mark the

component with `@stop-drag` to prevent the inner component drag function. Here is the code example:



```
<html>
  <template id="layout-template">
    <webpdf>
      <div class="fv__ui-body">
        <viewer></viewer>
      </div>
    </template>
    <layer name="my-layer1" class="center my-layer" @draggable visible>
      <layer-header title="Drag everywhere" icon-class="fv__icon-toolbar-print"></layer-header>
      <div>
        <label>
          without @stop-drag:
          <input type="range" min="0" max="100" step="0.1">
        </label>
        <label>
          marked with @stop-drag:
          <input @stop-drag type="range" min="0" max="100" step="0.1">
        </label>
      </div>
    </layer>
  </template>
</webpdf>
</template>
</html>
<style>
  .my-layer {
    width: 400px;
    height: 300px;
  }
  .flex-container {
```

```
    display: flex;
    justify-content: space-between;
  }
</style>
<script>
  var CustomAppearance = UIExtension.appearances.Appearance.extend({
    getLayoutTemplate: function() {
      return document.getElementById('layout-template').innerHTML;
    },
    disableAll: function(){}
  });
  var libPath = window.top.location.origin + '/lib';
  var pdfui = new UIExtension.PDFUI({
    viewerOptions: {
      libPath: libPath,
      jr: {
        licenseSN: licenseSN,
        licenseKey: licenseKey
      }
    },
    renderTo: document.body,
    appearance: CustomAppearance,
    addons: []
  });
</script>
```

@device

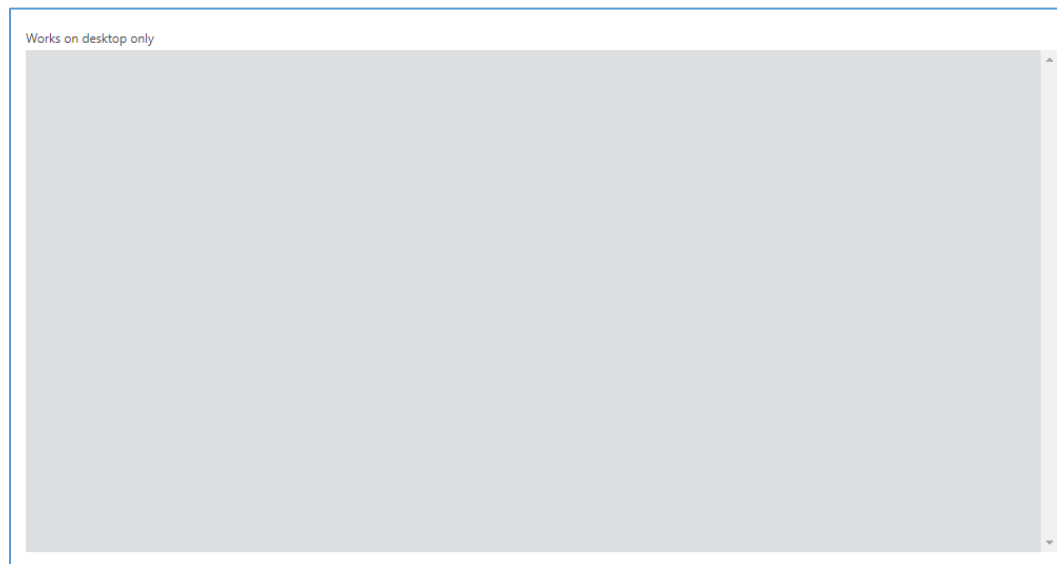
The `@device` directive is used to specify a list of device types and to restrict components to run only on the marked (specified) devices. If the currently running device type doesn't match with any of the marked devices, the component with the device markers won't be resolved and calling `getComponentByName()` or `querySelector` will fail to retrieve the components

Device Type

Name	Description
mobile	mobile devices
tablet	tablet devices
desktop	desktop
touch	touchable screen devices
android	android devices

Name	Description
iphone	iphones
ios	devices with IOS OS
ipad	ipad

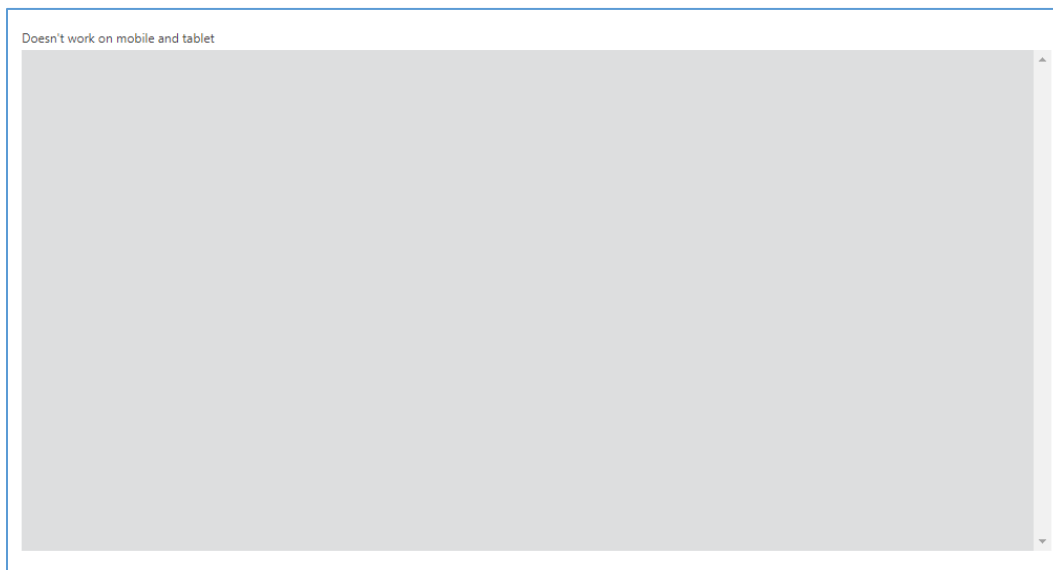
Example



```
<html>
  <template id="layout-template">
    <webpdf>
      <div>
        <xbutton name="desktop-button" @device="desktop">Works on desktop only</xbutton>
        <xbutton name="mobile-tablet-button" @device="mobile,tablet">Works on mobile and
tablet</xbutton>
      </div>
      <div class="fv__ui-body">
        <viewer></viewer>
      </div>
    </webpdf>
  </template>
</html>
<script>
  var CustomAppearance = UIExtension.appearances.Appearance.extend({
    getLayoutTemplate: function() {
      return document.getElementById('layout-template').innerHTML;
    },
    disableAll: function(){}
  });
```

```
afterMounted: function(rootComponent) {
    // In addition to the desktop, other devices will return null
    var desktopButton = rootComponent.getComponentByName('desktop-button');
    // in addition to mobile and tablet device, other devices will return null;
    var mobileTabletButton = rootComponent.getComponentByName('mobile-tablet-button');
    console.info(desktopButton, mobileTabletButton);
}
});
var libPath = window.top.location.origin + '/lib';
var pdfui = new UIExtension.PDFUI({
    viewerOptions: {
        libPath: libPath,
        jr: {
            licenseSN: licenseSN,
            licenseKey: licenseKey
        }
    },
    renderTo: document.body,
    appearance: CustomAppearance,
    addons: []
});
</script>
```

@device.invert example



```
<html>
  <template id="layout-template">
    <webpdf>
      <div>
        <xbutton name="desktop-button" @device.invert="desktop">Doesn't work on desktop</xbutton>
        <xbutton name="mobile-tablet-button" @device.invert="mobile,tablet">Doesn't work on mobile and
tablet</xbutton>
      </div>
```

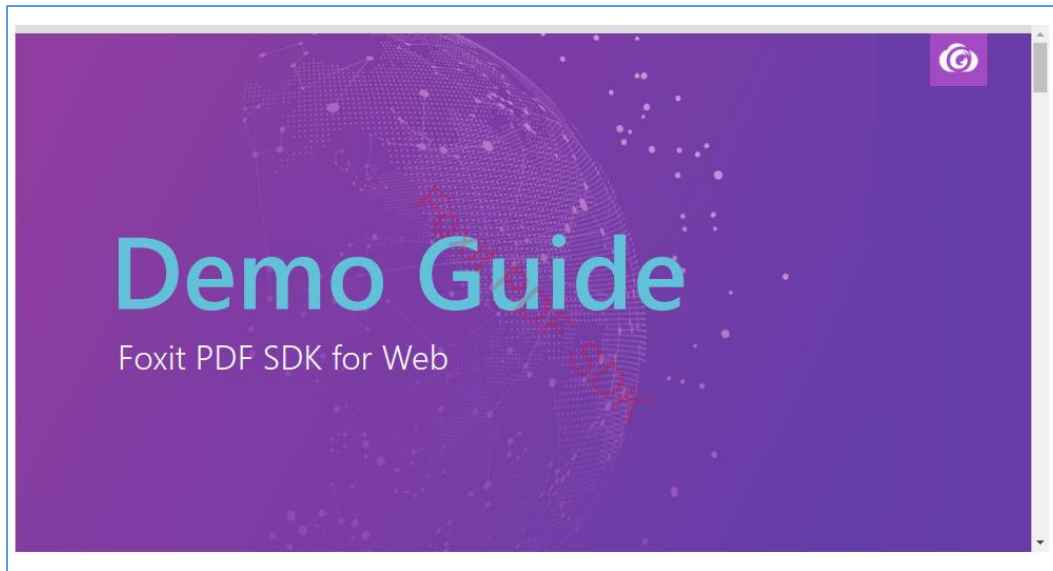
```
<div class="fv__ui-body">
  <viewer></viewer>
</div>
</webpdf>
</template>
</html>
<script>
  var CustomAppearance = UIExtension.appearances.Appearance.extend({
    getLayoutTemplate: function() {
      return document.getElementById('layout-template').innerHTML;
    },
    disableAll: function() {},
    afterMounted: function(rootComponent) {
      // In desktop, this will return null
      var desktopButton = rootComponent.getComponentByName('desktop-button');
      // in mobile and tablet device will return null
      var mobileTabletButton = rootComponent.getComponentByName('mobile-tablet-button');
      console.info(desktopButton, mobileTabletButton);
    }
  });
  var libPath = window.top.location.origin + '/lib';
  var pdfui = new UIExtension.PDFUI({
    viewerOptions: {
      libPath: libPath,
      jr: {
        licenseSN: licenseSN,
        licenseKey: licenseKey
      }
    },
    renderTo: document.body,
    appearance: CustomAppearance,
    addons: []
  });
</script>
```

@require-modules

The `@require-modules` directive is used to determine if a [module](#) exists, and if it does not, the component marked with this directive won't be resolved.

Code example

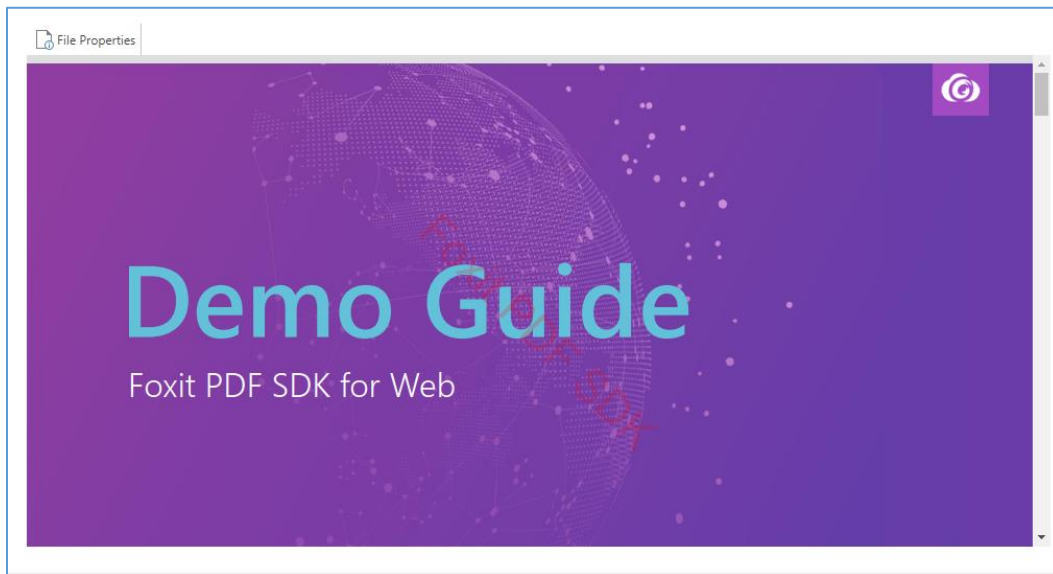
When you run the following example, you won't see any buttons because the `file-Property` addon isn't loaded.



```
<html>
  <template id="layout-template">
    <webpdf>
      <group-list>
        <!-- 'fpmodule' is a module defined in the 'file-property' addon which is not declared in the
        `addons:[]`, this group component will not be rendered -->
        <group name="file-property" @require-modules="fpmodule">
          <fpmodule:file-property-button></fpmodule:file-property-button>
        </group>
      </group-list>
      <div class="fv__ui-body">
        <viewer></viewer>
      </div>
      <template>
        <fpmodule:file-property-dialog></fpmodule:file-property-dialog>
      </template>
    </webpdf>
  </template>
</html>
<script>
  var CustomAppearance = UIExtension.appearances.Appearance.extend({
    getLayoutTemplate: function() {
      return document.getElementById('layout-template').innerHTML;
    }
  });
  var libPath = window.top.location.origin + '/lib';
  var pdfui = new UIExtension.PDFUI({
    viewerOptions: {
      libPath: libPath,
      jr: {
        licenseSN: licenseSN,
        licenseKey: licenseKey
```

```
    }  
  },  
  renderTo: document.body,  
  appearance: CustomAppearance,  
  addons: [] // No addon is loaded  
});  
var origin = window.top.location.origin;  
var url = origin + window.top.location.href.slice(origin.length).replace(/((V.*)?VdocsV).*/,  
'$1FoxitPDFSDKforWeb_DemoGuide.pdf');  
pdfui.openPDFByHttpRequest({  
  range: {  
    url: url,  
  }  
}, { fileName: 'FoxitPDFSDKforWeb_DemoGuide.pdf' })  
  
window.addEventListener(UIExtension.PDFViewCtrl.DeviceInfo.isDesktop ? 'resize' : 'orientationchange',  
function(e) {  
  pdfui.redraw().catch(function(err) {console.log(err)});  
});  
</script>
```

When you run the following example, you will see the `File Property` because the `file-Property` addon is loaded.



```
<html>  
  <template id="layout-template">  
    <webpdf>  
      <group-list>  
        <group name="file-property" @require-modules="fpmodule">  
          <fpmodule:file-property-button></fpmodule:file-property-button>  
        </group>  
      </group-list>
```

```
<div class="fv__ui-body">
  <viewer></viewer>
</div>
<template>
  <fpmodule:file-property-dialog></fpmodule:file-property-dialog>
</template>
</webpdf>
</template>
</html>
<script>
var CustomAppearance = UIExtension.appearances.Appearance.extend({
  getLayoutTemplate: function() {
    return document.getElementById('layout-template').innerHTML;
  }
});
var libPath = window.top.location.origin + '/lib';
var pdfui = new UIExtension.PDFUI({
  viewerOptions: {
    libPath: libPath,
    jr: {
      licenseSN: licenseSN,
      licenseKey: licenseKey
    }
  },
  renderTo: document.body,
  appearance: CustomAppearance,
  addons: [
    libPath + '/uix-addons/file-property'
  ] // the `file-property` addon will be loaded
});
var origin = window.top.location.origin;
var url = origin + window.top.location.href.slice(origin.length).replace(/((V.*)?VdocsV).*/,
'$1FoxitPDFSDKforWeb_DemoGuide.pdf');
pdfui.openPDFByHttpRequest({
  range: {
    url: url,
  }
}, { fileName: 'FoxitPDFSDKforWeb_DemoGuide.pdf' })

window.addEventListener(UIExtension.PDFViewCtrl.DeviceInfo.isDesktop ? 'resize' : 'orientationchange',
function(e) {
  pdfui.redraw().catch(function(err) {console.log(err)});
});
</script>
```


Addons

Introduction to addons

In the `/lib/uix-addons` directory, Foxit PDF SDK for Web provides a set of rich addons that can be freely combined. The following structure lists the currently supported addons. Each addon can be loaded individually or in combination as needed.

```
uix-addons
├──edit-graphics      ----- Edit page objects
├──text-object       ----- Edit text objects
├──export-form       ----- Export form
├──file-property     ----- File information
├──full-screen      ----- Full screen
├──h-continuous     ----- Horizontal continuous page mode
├──h-facing         ----- Horizontal facing mode for the cover
├──h-single         ----- Horizontal single page mode
├──import-form      ----- Import form
├──multi-media      ----- Multimedia comment
├──password-protect ----- Password protection
├──path-objects     ----- Edit path objects
├──print            ----- Print
├──redaction        ----- Redaction
├──undo-redo        ----- undo-redo
└──allInOne.js      ----- All the add-ons collection
```

The following lists all available addons, and each addon can be got by the static method `getName()`.

```
'edit-graphics'
'exportForm'
'file-property'
'full-screen'
'h-continuous'
'h-facing'
'h-single'
'importForm'
'multi-media'
'password-protect'
'edit-pageobjects'
'print'
'redaction'
'editTextObject'
'undo-redo'
```


Load Addons

Load addons individually

You can load all available addons or a few specific addons individually.

Code Example:

```
<script src="path/to/UIExtension.full.js"></script>
<script>
  var pdfui = new UIExtension.PDFUI({...
    addons:[
      "path/to/customized-addon/addon.info.json",
      "path/to/lib/multi-media/addon.info.json",
      ...
    ],
    ...
  })
</script>
```

The file `/examples/UIExtension/complete_webViewer/index.html` also provides an example to show how load all addons individually.

Load addons in combination

In the network environment, downloading too many addons would increase the HTTP requests. To minimize the number of HTTP requests, you may prefer loading all addons in a single file. We provide a script file `allInOne.js` which is a combination of all addons. Besides, you can use our [merge addons tools](#) to tailor addons.

The addon's structure

The entry file -- addon.info.json

The `addon.info.json` is the addon entry file, which includes the addon's library name, i18n sources and css files.

Example:

```
{
  "library": "ExampleUIAddon",
  "i18n": {
    "ns": "example",
    "sources": {
      "en-US": "./locales/en-US.json",
      "zh-CN": "./locales/zh-CN.json"
    }
  }
}
```

```
},  
"css": [  
  "./index.css"  
]  
}
```

The "i18n" sources

This item is used to configure localization. "ns" specifies namespace. "sources" specifies files.

After configuration, you can use `[i18n-namespace]:[i18n-key]` to implement localization.

In the case below, i18n namespace is "example", "i18n-key" could be "toolbar.title", "dialog.title" or "buttons.addText" (Refer to zh-CN.json for details).

```
<h6>example:dialog.title</h6>
```

will be translated to

```
<h6>Dialog title</h6>
```

And

```
<h6>对话框标题 Dialog title</h6>
```

The "css" field

This item specifies style sheets ("index.css" is the output of style-loader). Currently only CSS is supported.

allInOne.js

The `allInOne.js` is a script file that contains all currently supported addons. It is provided for your convenience to load addons in a combination way. This file is only for PC. If you want it for mobile, you should rebuild it to cut out the feature `editTextObject` which is unavailable in mobile.

You have two methods to build a custom addons:

- Use our [merge addons tools](#) to merge desired addons, and output a new all-in-one addons' file.
- Use the `UIXAddons.filter` to dynamically detach an addon from `allInOne.js` and load it. Check out [Loading custom addInOne.js](#).

Load allInOne.js

```
<script src="path/to/UIExtension.full.js"></script>  
<script src="path/to/allInOne.js"></script>  
<script>
```

```
var pdfui = new UIExtension.PDFUI({...
  addons:UIXAddons, // UIXAddons is the library name in allInOne.js
  ...
})
</script>
```

Loading custom addInOne.js

The default `allInOne.js` combines all currently supported addons into a single script. You can detach the unwanted addon by `UIXAddons.filter` and then load them to web viewer.

```
<script src="path/to/UIExtension.full.js"></script>
<script src="path/to/allInOne.js"></script>
<script>
  UIXAddons = UIXAddons.filter(addon=>addon.getName() != 'editTextObject')
  var pdfui = new UIExtension.PDFUI({...
    addons:UIXAddons,
    ...
  })
</script>
```

Merge addons

If you want to merge addons by yourself or rebuild `allInOne.js` to merge your selected addons, you can use our merge addon tools `addon loader` and `gulp plugin`. Check out the links below for details.

- [addon-loader for webpack](#)
- [addon merge tool for gulp](#)

You can also refer to `/examples/UIExtension/use-merged-addon` for usage.

Develop custom addons

The `/examples/UIExtension/scaffoldDemo/` is a scaffold project that contains an open source addon example. You may refer to the [/examples/UIExtension/scaffoldDemo/readme.md](#) file in that directory to start developing your own addons.

Customization

From version 7.0, Foxit PDF SDK for Web comes with built-in UI design including the feature modules UI, which are implemented using Foxit PDF SDK for Web and are shipped in the UIExtension.js. Furthermore, customizing UI is straightforward. Foxit PDF SDK for Web provides a rich set of APIs for developers to customize and style the appearance of your web viewer.

The user interface of UIExtension consists of two parts: [template](#) and [fragments](#). Template is equivalent to the extension of HTML, the components in the template should be declared by tags. Template is used to customize the UI layout (css style, icon, text, and so on) without interactions. Fragments are a set of UI snippets, which can be used to customize the configuration items and interaction logic of the components in the template. Each snippet has an operation type "action" that specifies the action mode (append, prepend, before, after, ext, replace, insert, and remove, the default is ext.) of the snippets. Through these action modes, you can insert, delete, replace and modify the components in the template.

For all the built-in components used in the following examples, please refer to [Understanding the built-in components](#).

Customize the UI

Customize the UI layout using template

Template is mainly used to customize the UI layout of the components. Following will list some examples to demonstrate the usage of template. Please note that all the examples are based on the [Integrate the complete web viewer into your project](#) in chapter Integration.

Create a simple template

A simplest template is as follows:

```
var pdfui = new UIExtension.PDFUI({
  viewerOptions: {
    libPath: './lib', // the library path of web sdk.
    jr: {
      licenseSN: licenseSN,
      licenseKey: licenseKey
    }
  }
})
```

```
},
renderTo: '#pdf-ui', // the div (id="pdf-ui").
appearance: UIExtension.appearances.Appearance.extend({
  getLayoutTemplate: function() {
    return [
      '<webpdf>',
      '  <viewer></viewer>',
      '</webpdf>'
    ].join("");
  }
})
});
```

- <webpdf> tag listens the opening/closing document events, and then trigger the **enableAll** or **disableAll** function of the **Appearance** object to enable or disable the related components in the UI.
- <viewer> tag is where the PDF contents are rendered. Each template must have a <viewer> tag. It can be placed anywhere you want, please refer to the examples in the following sections.

Refresh your browser (<http://127.0.0.1:8080/index.html>), and then you can see a simple web PDF viewer without any UI component.

Add a new toolbar button

Use <toolbar> tag to add a new toolbar button.

```
var pdfui = new UIExtension.PDFUI({
  viewerOptions: {
    libPath: './lib', // the library path of web sdk.
    jr: {
      licenseSN: licenseSN,
      licenseKey: licenseKey
    }
  },
  renderTo: '#pdf-ui', // the div (id="pdf-ui").
  appearance: UIExtension.appearances.Appearance.extend({
    getLayoutTemplate: function() {
      return [
        '<webpdf>',
        '  <toolbar>',
        '    <open-file-dropdown></open-file-dropdown>',
        '  </toolbar>',
        '  <viewer></viewer>',
        '</webpdf>'
      ].join("")
    }
  })
});
```

```
});
```

Refresh your browser (<http://127.0.0.1:8080/index.html>), and then you can see the new toolbar button.

Add a new tab page

Use <tabs> and <tab> tags to add a new tab page.

```
var CustomAppearance = UIExtension.appearances.Appearance.extend({
  getLayoutTemplate: function() {
    return [
      '<webpdf>',
      '  <toolbar>',
      '    <gtab group="custom-tabs" text="Home" body="home-tab-body">',
      '    </gtab>',
      '    <gtab group="custom-tabs" text="Comment" body="comment-tab-body">',
      '    </gtab>',
      '  </toolbar>',
      '  <div class="tab-bodies">',
      '    <div name="home-tab-body">',
      '      <open-file-dropdown></open-file-dropdown>',
      '    </div>',
      '    <div name="comment-tab-body" class="flex-row">',
      '      <create-strikeout-button></create-strikeout-button>',
      '      <create-underline-button></create-underline-button>',
      '      <create-squiggly-button></create-squiggly-button>',
      '      <create-replace-button></create-replace-button>',
      '      <create-caret-button></create-caret-button>',
      '      <create-note-button></create-note-button>',
      '    </div>',
      '  </div>',
      '  <viewer></viewer>',
      '</webpdf>'
    ].join("")
  }
})
var pdfui = new UIExtension.PDFUI({
  viewerOptions: {
    libPath: './lib', // the library path of web sdk.
    jr: {
      licenseSN: licenseSN,
      licenseKey: licenseKey
    }
  },
  renderTo: '#pdf-ui', // the div (id="pdf-ui").
  appearance: CustomAppearance
});
```

Refresh your browser (<http://127.0.0.1:8080/index.html>), and then you can see the new tab page.

Add a sidebar button

Use <sidebar> tag to add a sidebar button.

```
var CustomAppearance = UIExtension.appearances.Appearance.extend({
  getLayoutTemplate: function() {
    return [
      '<webpdf>',
      '  <toolbar>',
      '    <gtab group="custom-tabs" text="Home" body="home-tab-body">',
      '    </gtab>',
      '    <gtab group="custom-tabs" text="Comment" body="comment-tab-body">',
      '    </gtab>',
      '  </toolbar>',
      '  <div class="tab-bodies">',
      '    <div name="home-tab-body">',
      '      <open-file-dropdown></open-file-dropdown>',
      '    </div>',
      '    <div name="comment-tab-body" class="flex-row">',
      '      <create-strikeout-button></create-strikeout-button>',
      '      <create-underline-button></create-underline-button>',
      '      <create-squiggly-button></create-squiggly-button>',
      '      <create-replace-button></create-replace-button>',
      '      <create-caret-button></create-caret-button>',
      '      <create-note-button></create-note-button>',
      '    </div>',
      '  </div>',
      '  <div class="flex-row">',
      '    <sidebar>',
      '      <bookmark-sidebar-panel></bookmark-sidebar-panel>',
      '    </sidebar>',
      '    <viewer></viewer>',
      '  </div>',
      '</webpdf>'
    ].join("")
  }
});

var pdfui = new UIExtension.PDFUI({
  viewerOptions: {
    libPath: './lib', // the library path of web sdk.
    jr: {
      licenseSN: licenseSN,
      licenseKey: licenseKey
    }
  },
  renderTo: '#pdf-ui', // the div (id="pdf-ui").
  appearance: CustomAppearance
});
```

Refresh your browser (<http://127.0.0.1:8080/index.html>), and then you can see the bookmark sidebar.

Built-in layout template

The built-in layout template is placed in the `examples\UIExtension\layout_templates` directory. For desktop, reference the **built-in-pc-layout-template.tpl** file, and for mobile, reference the **built-in-mobile-layout-template.tpl** file. You can modify the template directly to customize the UI layout as desired.

In the "examples\UIExtension\custom_appearance" directory, Foxit PDF SDK for Web provides two custom template examples. "**adaptive-to-the-device.html**" is adaptive to the device (desktop and mobile), and "**not-adaptive-to-the-device.html**" is not adaptive to the device, which only supports desktop.

Customize the UI using fragments

Fragments are a set of UI snippets. It can be used to customize the configuration items and interaction logic of the components in the template.

Create a new drop-down menu item

The sample code below creates a new drop-down menu including two drop-down buttons, and append it to the end of the list of children of the group component with the name of "home-tab-group-hand".

To get the name of a target component (only for widget), you can right-click the component in the browser, choose **Inspect**, and then find the value of the `component-name` attribute in the corresponding `<a>` tag. For a container component, for example `target: 'home-tab-group-hand'`, you can right-click one of the subcomponents in the browser, choose "**Inspect**", and then find the value of the "`component-name`" attribute in the related `<div>` tag.

```
const customModule = UIExtension.PDFUI.module('custom', []);
customModule.controller('CustomController', {
  mounted: function () {
    console.info(this.component, 'mounted');
  },
  handle: function (selectedFile) {
    alert(selectedFile.name);
  }
});
var CustomController = customModule.getControllerClass('CustomController');

var CustomAppearance = UIExtension.appearances.RibbonAppearance.extend({
  getDefaultFragments: function() {
    return [{
      // Add a component to the end of the list of children of a specified target component.
```



```
        action: UIExtension.UIConsts.FRAGMENT_ACTION.APPEND,
        // Specify the name of the target component that the new components defined in the above template
        // will be appended to. All the target names of fragments are defined in the layout template.
        target: 'home-tab-group-hand',
        // Define the properties of the added component, such as icon, text, and css style.
        template: [
            '<dropdown icon-class="fv__icon-toolbar-stamp">',
            '  <dropdown-button name="show-hello-button" icon-class="fv__icon-toolbar-hand">say
hello</dropdown-button>',
            '  <dropdown-button name="select-pdf-file-button" accept=".pdf" file-selector icon-class="fv__icon-
toolbar-open">open</dropdown-button>',
            '</dropdown>'
        ].join(""),
        // Define the interaction logic of the added component.
        config: [{
            // specify the component in the above template that the configuration will be applied to.
            // For example, the configuration will be applied to the component with the name of "show-hello-
            button".
            target: 'show-hello-button',
            callback: function () {
                alert('hello');
            }
        },
        {
            // The configuration will be applied to the component with the name of "select-pdf-file-button" which is
            // defined in the above template of fragments.
            target: 'select-pdf-file-button',
            // Extend Controller, and implement the handle function.
            callback: CustomController
        }
    ]
  }
})

var pdfui = new UIExtension.PDFUI({
  viewerOptions: {
    libPath: './lib', // the library path of web sdk.
    jr: {
      licenseSN: licenseSN,
      licenseKey: licenseKey
    }
  },
  renderTo: '#pdf-ui', // the div (id="pdf-ui").
  appearance: CustomAppearance
});
```

Refresh your browser (<http://127.0.0.1:8080/index.html>), and then you can see a new created drop-down menu.

Delete a toolbar button

To delete a toolbar button through fragment is fairly simple. For example, to delete the Hand tool, you only need to add a new object to the fragment. Based on the above example, add the code below:

```
{
  target: 'hand-tool',
  action: UIExtension.UIConsts.FRAGMENT_ACTION.REMOVE
}
```

Refresh your browser (<http://127.0.0.1:8080/index.html>), and then you can see the Hand tool has been removed from the group component.

Modify a toolbar button

To modify a toolbar button through fragment is also fairly simple. Just like [Delete a toolbar button](#), you only need to add a new object to the fragment.

Change icon of a button

For example, to change the icon of the Hand tool, just add the code below:

```
{
  target: 'hand-tool',
  config: {
    iconCls: 'fv__icon-toolbar-note' // your custom icon.
  }
}
```

Refresh your browser (<http://127.0.0.1:8080/index.html>), and then you can see the icon of the Hand tool has been changed.

Change the tooltip of a button

For example, to change the tooltip of the Hand tool, just add the code below:

```
{
  target: 'hand-tool',
  config: {
    tooltip: {
      title: 'your custom tooltip'
    }
  }
}
```

Refresh your browser (<http://127.0.0.1:8080/index.html>), and then you can see the tooltip of the Hand tool has been changed to "your custom tooltip".

Change the event of a button

To change the event of a button, there are two ways:

- 1) Overwrite the built-in event. For example:

```
{
  target: 'hand-tool',
  config: {
    callback: function() {
      alert('your click event handler');
    }
  }
}
```

- 2) Add custom behavior based on the built-in event (original logic). You can configure the button using the method as follows:

```
{
  target: 'hand-tool',
  config: {
    callback: {
      before: function (handleMethodArguments) {
        console.info('called before handle callback with arguments: ', handleMethodArguments);
      },
      after: function (value, handleMethodArguments) {
        console.info('called after handle callback with returning value and arguments: ', value,
handleMethodArguments);
      }
    }
  }
}
```

More examples

For more sample codes, please refer to the examples in the `examples\UIExtension\fragment_usage` folder.

Modularization

In order to differentiate the built-in components and user-defined components to avoid conflicts, UIExtension provides modularization feature, which allows you to register the components in different modules separately. Then, you only need to add the prefix of the module name when you declare the components in the template.

Create your own custom module

If you want to define and use a custom component which has the same name with the built-in component, you can create a custom module and then register your custom component in the custom module.

For example, in the built-in component, there is already an existing component with the name of **"dropdown"**, if you also want to define a custom component called **"dropdown"**, you can refer to the simple code below:

Create a new module "my-widgets", and registers a user-defined component in this module:

```
// Create a new module. Please note that the second parameter must be an array if you create a new module.
var module = UIExtension.modular.module('my-widgets', []);

function UserDefinedDropdownComponent() {
  UIExtension.Component.apply(this, arguments);
}
UserDefinedDropdownComponent.getName = function() {
  return 'dropdown'; // Declare the tag name of the component. There is already an existing component with
the same name of 'dropdown' in the built-in component.
}
UserDefinedDropdownComponent.prototype.constructor = UIExtension.Component;
UserDefinedDropdownComponent.prototype.render = function() {
  UIExtension.Component.prototype.render.call(this);
  this.element.innerText = 'User-defined dropdown component';
}
module.registerComponent(UserDefinedDropdownComponent);
```

Then, the built-in dropdown and user-defined dropdown can be differentiated in the following way:

```
<!-- built-in dropdown -->
<dropdown></dropdown>
<!-- user-defined dropdown -->
<my-widgets:dropdown></my-widgets:dropdown>
```

For example, use the user-defined dropdown component:

```
var pdfui = new UIExtension.PDFUI({
  // Omit other parameters.
  appearance: UIExtension.appearances.RibbonAppearance.extend({
    getDefaultFragments: function() {
      return [{
        action: UIExtension.UIConsts.FRAGMENT_ACTION.APPEND,
        target: 'home-tab-group-hand',
        template: '<my-widgets:dropdown></my-widgets:dropdown>' // use a colon to separate the module
name and component name in the template.
      }];
    }
  });
```

```
}  
}  
});
```

Understanding the built-in components

In simple terms, a tag defined in the template is a component. Container and Widget also belong to the component. The different is that container can hold subcomponents, but widget cannot. All of the native HTML tags are defined as container components, so you can freely nest them.

Foxit PDF SDK for Web has already wrapped a rich set of built-in components for developers to quickly build a web viewer, and customize it as desired. This section will list and introduce the basic components and business components.

Basic components

Type	Name	Description
container	toolbar	Toolbar components, like an empty div (just has bottom border)
	tabs	Tab components, which are the parent components of tab and used to manage the tab pages. Note: <i>Tab must exist as the subcomponents of the tabs component.</i>
	tab	Tab pages.
	group-list	Group list, which is used to manage group components.
	group	Group components, which must exist as the subcomponents of group-list.
	sidebar	Sidebar, which managers the sidebar panels.
	sidebar-panel	Sidebar panel, which allow you to specify the icon, title and content of the panel. For example, the <i>sidebar-search</i> , and <i>sidebar-bookmark</i> in the demo page.
	layer	Floating box component. It supports translucent backdrop and modal box, which can be used to implement dialogs.
	layer-header	A component for the head of a dialog, which can be set with a title and a close button.
	layer-toolbar	Equivalent to a div with 1.5em height and flex layout. It is usually used to store buttons or other components.
	layer-view	Equivalent to a div with <i>fv-ui-layer-panel</i> . It is usually used to store the contents of dialogs.
	dropdown	Dropdown component.

Type	Name	Description
	dropdown-item	Dropdown items.
	contextmenu	Context menu component. The subcomponents must be contextmenu-item or contextmenu-separator. To show context menu component, you need to call showAt method.
widget	dropdown-button	A type of dropdown items. It will trigger the controller's handle method and click event after clicking. If set to file selector mode, it will trigger the change event after clicking to select a file, and also pass the selected file to the controller's handle method.
container	accordion	Accordion component.
	accordion-card	The card of accordion component. It can be set with a title, and the content can be shrunk. It must be present as the subcomponent of the accordion.
	webpdf	The root component of the default template. It will listen the document open and close event of pdfviewer object, and control to disable/enable all the components depending on the document state.
widget	viewer	A component that renders PDF files. Each template file must be configured with a viewer component.
	text	It will be parsed into a text node in the DOM tree. The content of the text component will be localized by i18next library.
	xbutton	Button component. It supports setting the icon and text. After clicking, it will trigger the click event and call the controller's handle method.
	file-selector	File selection component. It supports setting icon and text. After clicking and selecting a file, it will trigger the change event, pass the selected file to the controller's handle method and call it.
	number	Number input box. You can set the number range and step size. The default range is infinite, and it has no step size by default. After setting the step size, if the minimum value is not equal to infinitely small, then the value of the input box must be (the minimum value + step size* integer).
	contextmenu-item	Context menu item. It must be as a subcomponent of contextmenu. Other configurations and usages are the same as xbutton.
	contextmenu-separator	The separator line of context menu. It has no other special effects.

Type	Name	Description
slot	slot	<p>Slot can be used to implement the requirement for inserting subcomponents to the different locations of the complex container components.</p> <p>Note: If "for="slotName", slotName" is not supported, it will report the "slot does not exist" error. In this case, it depends on how the corresponding component of the slot's parent node handle the value, please refer to the Component#appendSlot method.</p>

Business components

Type	Name	Inherited Component	Description
widget	hand-button	xbutton	Hand tool button
widget	selection-button	xbutton	Text/Annotation selection tool button
widget	snapshot-button	xbutton	Snapshot tool button
widget	open-file-dropdown	dropdown	Dropdown menu for opening document
widget	download-file-button	xbutton	Download file button
widget	goto-prev-page-button	xbutton	Go to the previous page
widget	goto-next-page-button	xbutton	Go to the next page
widget	goto-page-input	number	The input box for the page number that you want to jump to.
widget	zoom-out-button	xbutton	Zoom out page button
widget	zoom-in-button	xbutton	Zoom in page button
widget	editable-zoom-dropdown	dropdown	The editable zoom dropdown menu
widget	zoom-dropdown	dropdown	Zoom dropdown menu
widget	continuous-page-button	xbutton	A button used to enable continuous page mode which views pages continuously with scrolling enabled
widget	continuous-facing-page-button	xbutton	A button used to enable continuous-facing page mode which views pages side-by-side with continuous scrolling enabled
widget	loupe-tool-button	xbutton	Loupe tool button

Type	Name	Inherited Component	Description
widget	marquee-tool-button	xbutton	Marquee tool button
widget	create-text-highlight-button	xbutton	Text highlight tool button
widget	create-strikeout-button	xbutton	Text strikeout tool button
widget	create-underline-button	xbutton	Text underline tool button
widget	create-squiggly-button	xbutton	Text squiggly tool button
widget	create-replace-button	xbutton	Text replace tool button
widget	create-caret-button	xbutton	Caret tool button
widget	create-note-button	xbutton	Note tool button
widget	create-typewriter-button	xbutton	Typewriter tool button
widget	create-callout-button	xbutton	Callout tool button
widget	create-textbox-button	xbutton	Textbox tool button
widget	create-drawings-dropdown	dropdown	Shapes tool dropdown
widget	create-pencil-button	xbutton	Pencil tool button
widget	eraser-button	xbutton	Eraser tool button
widget	create-area-highlight-button	xbutton	Area highlight tool button
widget	create-distance-button	xbutton	Distance tool button
widget	create-attachment-button	xbutton	Attachment tool button
widget	create-image-button	xbutton	Image tool button
widget	create-link-button	xbutton	Link tool button
widget	stamp-dropdown	dropdown	Stamp tool dropdown list
widget	add-image-button	xbutton	Add image tool button
widget	ink-sign-dropdown	dropdown	Ink signature dropdown list
widget	create-stamp-dialog	layer	Create stamp dialog

Type	Name	Inherited Component	Description
widget	loupe-tool-dialog	layer	Loupe tool dialog
widget	create-ink-sign-dialog	layer	Create ink signature dialog
widget	bookmark-sidebar-panel	sidebar-panel	Bookmark panel in the sidebar
widget	commentlist-sidebar-panel	sidebar-panel	Comment list panel in the sidebar
widget	thumbnail-sidebar-panel	sidebar-panel	Thumbnail panel in the sidebar
widget	layer-sidebar-panel	sidebar-panel	PDF layers panel in the sidebar
widget	search-sidebar-panel	sidebar-panel	Search panel in the sidebar
widget	attachment-sidebar-panel	sidebar-panel	Attachments panel in the sidebar
container	page-contextmenu	contextmenu	Page context menu
container	annot-contextmenu	contextmenu	Annotation context menu
container	action-annot-contextmenu	contextmenu	Image and link annotation context menu
container	default-markup-contextmenu	contextmenu	Markup annotation context menu
container	fileattachment-contextmenu	contextmenu	Attachment annotation context menu
container	media-contextmenu	contextmenu	video/audio annotation context menu
container	redact-contextmenu	contextmenu	Redaction context menu
container	textmarkup-contextmenu	contextmenu	Caret/highlight/strikeout/underline/squiggly/typewriter/callout/textbox context menu
widget	contextmenu-item-annot-actions	contextmenu-item	Show action settings dialog
widget	contextmenu-item-apply-all	contextmenu-item	Apply all redaction annotations
widget	contextmenu-item-apply	contextmenu-item	Apply current redaction annotation

Type	Name	Inherited Component	Description
widget	contextmenu-item-copy-text	contextmenu-item	Copy the content data of the current annotation
widget	contextmenu-item-delete-annot	contextmenu-item	Delete current annotation
widget	contextmenu-item-hand-tool	contextmenu-item	Switch to hand tool
widget	contextmenu-item-marquee-zoom	contextmenu-item	Switch to loupe tool
widget	contextmenu-item-media-download	contextmenu-item	Download current multimedia data streams
widget	contextmenu-item-place	contextmenu-item	Apply current redaction annotation to other page
widget	contextmenu-item-properties	contextmenu-item	Show the properties dialog of current annotation
widget	contextmenu-item-reply	contextmenu-item	Reply current annotation
widget	contextmenu-item-search	contextmenu-item	Show search sidebar
widget	contextmenu-item-select-tool	contextmenu-item	Switch to selection tool
widget	contextmenu-item-zoom-actual-size	contextmenu-item	Zoom to 100%
widget	contextmenu-item-fitpage	contextmenu-item	Fit page (resize the page to fit entirely in the document pane)
widget	contextmenu-item-fitwidth	contextmenu-item	Fit width (resize the page to fit the width of the window. Part of the page may be out of view)
widget	contextmenu-item-zoomin	contextmenu-item	Zoom in
widget	contextmenu-item-zoomout	contextmenu-item	Zoom out

Type	Name	Inherited Component	Description
container	text-sel:text-selection-tooltip	layer	The floating box appears when selecting texts
container	freetext:freetext-tooltip	layer	The floating box appears when selecting freetext annotation
widget	comment-list:toggle-commentlist-group-button	xbutton	Collapse/expand the comment group components (annotations in one PDF page are organized into a same group) in commentlist panel

Business components in the add-on modules

The business components in the add-on modules are valid only when your project has already integrated the related add-on modules.

Type	Module:Name	Inherited Component	Add-on Module	Description
widget	print:print-button	xbutton	print	Open the print dialog
widget	print:print-dialog	layer	print	Print dialog
widget	import-form-module:import-form-button	xbutton	import-form	Import form data
widget	export-form-module:export-form-dropdown	xbutton	export-form	Export form data
widget	fpmodule:file-property-button	xbutton	file-property	Show current document information dialog
widget	fpmodule:file-property-dialog	layer	file-property	Document information dialog
widget	multi-media:multi-media-button	xbutton	multi-media	Switch to the multimedia Annotation creation tool
widget	edit-pageobjects:edit-all-objects-button	xbutton	path-objects	Switch to the page objects (image/path/text) editing tool
container	edit-pageobjects:page-objects-dropdown	dropdown	path-objects	Path objects creation tool dropdown list

Type	Module:Name	Inherited Component	Add-on Module	Description
widget	edit-text-object:add-text-button	xbutton	text-object	Text editing
widget	edit-text-object:text-bold-style-button	xbutton	text-object	Make text bold
widget	edit-text-object:text-italic-style-button	xbutton	text-object	Italicize text
container	edit-text-object:font-color-picker	dropdown	text-object	Set the color of text
container	edit-text-object:font-style-dropdown	dropdown	text-object	Set font and font size of text
widget	password-protect:password-protect-button	xbutton	password-protect	Open the password protect dialog
widget	password-protect:remove-protect-button	xbutton	password-protect	Remove password protect
container	redaction:create-redactions-dropdown	dropdown	redaction	Redaction options dropdown list
widget	redaction:apply-redactions-button	xbutton	redaction	Apply redaction
widget	redaction:redaction-search-button	xbutton	redaction	Open the left search panel
container	redaction:redaction-page-dialog	layer	redaction	Mark page to redact dialog
widget	h-continuous:h-continuous-button	xbutton	h-continuous	Switch to horizontal continuous page mode

Customize annotation right-click menu

This guide will walk you through the following topics:

- Customizing right-click menu for supported annotations
- Customizing right-click menu for unsupported annotations

- Hiding a right click menu or menu items
- Showing a customized right-click menu

Customize right-click menu for supported annotations

Web Viewer supports most of standard annotation types. Each type can have its own right-click menu. Below list the supported annotation and their corresponding XML element name.

```
{
  "line": "fv--line-contextmenu",
  "lineararrow": "fv--lineararrow-contextmenu",
  "linedimension": "fv--linedimension-contextmenu",
  "polylinedimension": "fv--polylinedimension-contextmenu",
  "polygondimension": "fv--polygondimension-contextmenu",
  "circle": "fv--circle-contextmenu",
  "square": "fv--square-contextmenu",
  "polyline": "fv--polyline-contextmenu",
  "polygon": "fv--polygon-contextmenu",
  "polygoncloud": "fv--polygoncloud-contextmenu",
  "fileattachment": "fv--fileattachment-contextmenu",
  "freetexttypewriter": "fv--freetexttypewriter-contextmenu",
  "typewriter": "fv--typewriter-contextmenu",
  "freetextcallout": "fv--freetextcallout-contextmenu",
  "callout": "fv--callout-contextmenu",
  "freetexttextbox": "fv--freetexttextbox-contextmenu",
  "textbox": "fv--textbox-contextmenu",
  "freetext": "fv--freetext-contextmenu",
  "ink": "fv--ink-contextmenu",
  "stamp": "fv--stamp-contextmenu",
  "text": "fv--text-contextmenu",
  "areahighlight": "fv--areahighlight-contextmenu",
  "highlight": "fv--highlight-contextmenu",
  "caret": "fv--caret-contextmenu",
  "replace": "fv--replace-contextmenu",
  "squiggly": "fv--squiggly-contextmenu",
  "strikeout": "fv--strikeout-contextmenu",
  "redact": "fv--redact-contextmenu",
  "underline": "fv--underline-contextmenu",
  "media": "fv--media-contextmenu",
  "image": "fv--image-contextmenu",
  "link": "fv--link-contextmenu",
  "sound": "fv--sound-contextmenu"
}
```

The annotation element name can be accessed by using `super.getAnnotsContextMenuName(owner)` in the `UIExtension.XViewerUI`. You may refer to the section [Customizing viewerUI](#) for a code example.

For the supported annotations, you can use the method

UIExtension.UIConsts.FRAGMENT_ACTION.APPEND action to replace, add and remove menu items.

Replacing menu items

```
new UIExtension.PDFUI({
  appearance: UIExtension.appearances.AdaptiveAppearance.extend({
    getDefaultFragments: function() {
      return [{
        target: 'fv--highlight-contextmenu',
        action: UIExtension.UIConsts.FRAGMENT_ACTION.REPLACE,
        template: `
          <contextmenu name="fv--highlight-contextmenu">
            <contextmenu-item-reply></contextmenu-item-reply>
            <contextmenu-item-delete-annot></contextmenu-item-delete-annot>
            <contextmenu-item-properties></contextmenu-item-properties>
            <contextmenu-itemname="x-user-custom-contextmenu-item">Custom </contextmenu-item>
          </contextmenu>`,
        config: [{
          target: 'x-user-custom-contextmenu-item',
          callback: function() {
            alert("custom contextmenu item clicked!");
          }
        }]
      }];
    }
  });
});
```

Adding menu items

```
new UIExtension.PDFUI({
  appearance: UIExtension.appearances.AdaptiveAppearance.extend({
    getDefaultFragments: function() {
      return [{
        target: 'fv--textbox-contextmenu',
        action: UIExtension.UIConsts.FRAGMENT_ACTION.APPEND,
        template: `
          <contextmenu-item name="x-user-custom-contextmenu-item"></contextmenu-item>
        `,
        config: [{
          target: 'x-user-custom-contextmenu-item',
          callback: function() {
            alert("custom contextmenu item clicked!");
          }
        }]
      }];
    }
  });
});
```

Removing menu items

```
new UIExtension.PDFUI({
  appearance: UIExtension.appearances.AdaptiveAppearance.extend({
    getDefaultFragments: function() {
      return [{
        target: 'fv--media-contextmenu>fv--contextmenu-item-media-download',
        action: UIExtension.UIConsts.FRAGMENT_ACTION.REMOVE
      }]
    }
  })
})
```

Customize right-click menu for unsupported annotations

Unsupported annotations mean annotations that are not numerated in the above supported list but have already been defined in the PDF references. To customize the context menu for an unsupported annotation, you should rewrite `getAnnotsContextMenuName` in `XViewUI` to create a new context menu and then add it in template.

A quick way to check if the current annotation is supported or not in Web Viewer, you may check their corresponding element name, which is by default labeled as `"fv--default-annot-contextmenu"`.

The following example is making assumption that your current PDF file contains a 'trapnet' annotation which is not yet supported in Web Viewer, and you want to customize its right-click menu.

```
new UIExtension.PDFUI({
  viewerOptions: {
    viewerUI: new class extends UIExtension.XViewerUI {
      createContextMenu(owner, anchor, config) {
        if(owner instanceof PDFViewCtrl.AnotComponent) {
          if(owner.annot.getType() === 'trapnet') {
            return 'custom-trapnet-contextmenu-name';
          }
        }
        return super.createContextMenu(owner, anchor, config);
      }
    }(),
  },
  appearance: UIExtension.appearances.AdaptiveAppearance.extend({
    getDefaultFragments: function() {
      return [{
        target: 'template-container',
        action: UIExtension.UIConsts.FRAGMENT_ACTION.APPEND,
        template: `
          <contextmenu name="custom-trapnet-contextmenu-name">
            <contextmenu-item-reply></contextmenu-item-reply>
          </contextmenu>
        `
      }]
    }
  })
})
```

```
        <contextmenu-item-delete-annot></contextmenu-item-delete-annot>
        <contextmenu-item-properties></contextmenu-item-properties>
        <contextmenu-item name="x-user-custom-contextmenu-item">Custom </contextmenu-item>
    </contextmenu>
    ,
    config:[{
        target: 'x-user-custom-contextmenu-item',
        callback: function() {
            alert('custom contextmenu item clicked!');
        }
    ]
}]
}
})
})
```

Hiding the right-click menu or items

You can use one of the following approaches to achieve the hiding.

1) Configuring a class method in fragments to force the hiding action

```
new UIExtension.PDFUI({
    appearance: UIExtension.appearances.AdaptiveAppearance.extend({
        getDefaultFragments: function() {
            // the other options ...
            return [{
                target: 'fv--underline-contextmenu',
                config: {
                    cls: 'fv__ui-force-hide'
                }
            }]
        }
    })
})
```

The effect of this method is that there is no response following the right-clicking on the underline.

2) Customizing viewerUI

```
new UIExtension.PDFUI({
    viewerOptions: {
        viewerUI: new class extends UIExtension.XViewerUI {
            createContextMenu(owner, anchor, config) {
                if(owner instanceof PDFViewCtrl.AnnotComponent) {
                    const contextMenuName = super.getAnnotsContextMenuName(owner)
                    if(contextMenuName === 'fv--underline-contextmenu'){
                        return;
                    }
                }
            }
        }
    }
})
```



```

    }
    return super.createContextMenu(owner, anchor, config);
  }
}
}
});

```

This method will hide the built-in menu when a right-clicking occurs, and present the browser default menu.

3) Overwrite the `showContextMenu` of `AnnotComponent`

```

const pdfui = new UIExtension.PDFUI({
  // ....
});
pdfui.initializePromise.then(function () {
  var annotMap = {};
  pdfui.registerMatchRule(function(annot, AnnotComponentClass) {
    let type = annot.getType();
    var intent = annot.getIntent() && annot.getIntent() || "default";
    // You can add more annotation types
    if(type === 'underline') {
      return AnnotComponentClass;
    }
    if (annotMap[type] && annotMap[type][intent]) {
      return annotMap[type][intent];
    }
    annotMap[type] = annotMap[type] || {};
    return annotMap[type][intent] = (class extends AnnotComponentClass {
      showContextMenu() {
        // Do nothing
      }
    });
  });
});

```

This method will hide the build-in menu and show the browser default menu of a right-clicking

Showing a customized right-click menu

You should overwrite the viewerUI to show your own right-click menu.

```

new UIExtension.PDFUI({
  viewerOptions: {
    viewerUI: new (class extends UIExtension.XViewerUI {
      createContextMenu(owner, anchor, config) {
        if (owner instanceof PDFViewCtrl.AnnotComponent) {
          const contextMenuName = super.getAnnotsContextMenuName(owner);
          if (contextMenuName === "fv--underline-contextmenu") {
            return new (class extends PDFViewCtrl.IContextMenu {

```

```
constructor() {
    super();
    this.initContextmenu();
}
destroy() {
    $(anchor).contextMenu("destroy");
}
showAt(x, y) {
    $(anchor).contextMenu();
}
disable() {
    super.disable();
    $(anchor).contextMenu("destroy");
}
enable() {
    super.enable();
    this.initContextmenu();
}
initContextmenu() {
    // The code example below requires referencing JQuery libraries including contextMenu.min.css,
contextMenu.min.js and min.js.
    $(anchor).contextMenu({
        selector: config.selector,
        items: [
            {
                name: 'show "Hello World"',
                callback: function() {
                    alert("hello world");
                }
            },
            {
                name: 'show "Bye!"',
                callback: function() {
                    alert("Bye!");
                }
            }
        ]
    });
}
})();
}
return super.createContextMenu(owner, anchor, config);
}
})();
}
});
```

Customize page right-click menu

This guide will present you a list of component name of the built-in page right-click menu, show you how to remove one of menu items through fragments and templates, add or insert new user items, and how to hide the menu via the fragments and viewerUI.

Page right-click menu items

The right-click menu is named as fv--page-contextmenu. It contains the following items:

- fv--contextmenu-item-full-screen
- fv--contextmenu-item-select-tool
- fv--contextmenu-item-hand-tool
- fv--contextmenu-item-marquee-zoom
- fv--contextmenu-item-zoom-actual-size
- fv--contextmenu-item-zoom-fitpage
- fv--contextmenu-item-zoom-fitwidth
- fv--contextmenu-item-print
- fv--contextmenu-item-search
- fv--file-property-contextmenu

Removing a menu item

The item in the target field will be deleted when the sample code below is executed.

```
new PDFUI({
  appearance: UIExtension.appearances.AdaptiveAppearance.extend({
    getDefaultFragments: function() {
      // the other options ...
      return [{
        target: "fv--contextmenu-item-zoom-actual-size",
        action: UIExtension.UIConsts.FRAGMENT_ACTION.REMOVE
      }]
    }
  })
});
```

Replacing a menu item

The item in the target field will be replaced when the sample code below is executed.

```
new UIExtension.PDFUI({
  appearance: UIExtension.appearances.AdaptiveAppearance.extend({
```

```
getDefaultFragments: function() {
    // the other options ...
    return [{
        target: "fv--contextmenu-item-zoom-actual-size",
        action: UIExtension.UIConsts.FRAGMENT_ACTION.REPLACE,
        template: `<contextmenu-item name="custom-contextmenu-item">customize contextmenu
item</contextmenu-item>`,
        config: [{
            target: "custom-contextmenu-item",
            callback: function() {
                alert("contextmenu item clicked!");
            }
        }]
    }]
}
});
```

Inserting a new item

A new item defined in the template will be added after the item in the target when the sample code below is executed.

```
new UIExtension.PDFUI({
    appearance: UIExtension.appearances.AdaptiveAppearance.extend({
        getDefaultFragments: function() {
            // the other options ...
            return [{
                target: "fv--contextmenu-item-zoom-actual-size",
                action: UIExtension.UIConsts.FRAGMENT_ACTION.AFTER,
                template: `<contextmenu-item name="custom-contextmenu-item">customize contextmenu
item</contextmenu-item>`,
                config: [
                    {
                        target: "custom-contextmenu-item",
                        callback: function() {
                            alert("contextmenu item clicked!");
                        }
                    }
                ]
            }]
        }
    })
});
```

Hiding the right-click menu or items

You can use one of the following approaches to achieve the hiding.

1) Configuring a class method in fragments to force the hiding action

```
new UIExtension.PDFUI({
  appearance: UIExtension.appearances.AdaptiveAppearance.extend({
    getDefaultFragments: function() {
      // the other options ...
      return [{
        target: "fv--page-contextmenu",
        config: {
          cls: "fv__ui-force-hide"
        }
      }]
    }
  })
});
```

The effect of this method is that there is no response following the right-clicking.

2) Customizing viewUI

```
new PDFUI({
  viewerOptions: {
    viewerUI: new (class extends UIExtension.XViewerUI {
      createContextMenu(owner, anchor, config) {
        switch (owner) {
          case PDFViewCtrl.STATE_HANDLER_NAMES.STATE_HANDLER_HAND:
          case PDFViewCtrl.STATE_HANDLER_NAMES
            .STATE_HANDLER_SELECT_ANNOTATION:
            return;
        }
        return super.createContextMenu(owner, anchor, config);
      }
    })()
  }
});
```

This method will hide the built-in menu when a right-clicking occurs, but present the browser default menu.

Showing a customized right-click menu

You should overwrite the viewerUI to show your own right-click menu.

```
new UIExtension.PDFUI({
  viewerOptions: {
    viewerUI: new (class extends UIExtension.XViewerUI {
      createContextMenu(owner, anchor, config) {
        switch (owner) {
          case PDFViewCtrl.STATE_HANDLER_NAMES.STATE_HANDLER_HAND:
          case PDFViewCtrl.STATE_HANDLER_NAMES
            .STATE_HANDLER_SELECT_ANNOTATION:
            return new (class extends PDFViewCtrl.IContextMenu {
```

```
constructor() {
    super();
    this.initContextmenu();
}
destroy() {
    $(anchor).contextMenu("destroy");
}
showAt(x, y) {
    $(anchor).contextMenu();
}
disable() {
    super.disable();
    $(anchor).contextMenu("destroy");
}
enable() {
    super.enable();
    this.initContextmenu();
}
initContextmenu() {
    //The code example below requires referencing in order JQuery libraries including
contextMenu.min.css, jquery.min.js and contextMenu.min.js.
    $(anchor).contextMenu({
        selector: config.selector,
        items: [
            {
                name: 'show "Hello World"',
                callback: function() {
                    alert("Hello world");
                }
            },
            {
                name: 'show "How do your do!"',
                callback: function() {
                    alert("How do you do!");
                }
            }
        ]
    });
}
})();
return super.createContextMenu(owner, anchor, config);
}
})();
}
});
```

Customize the Floating Text Selection Tooltip

Customizing the floating tooltip involves two steps. First you should create a custom controller to define your own logic for a target tool, and then add or edit the tool on fragments. The following sections will walk you through:

- A sample for creating a custom controller and modifying components by fragments
- The logic processing methods used in floating tooltip
- The component name of the floating tooltip

A sample for creating a custom controller and modifying components by fragmentation

Code sample can be accessed on </examples/UIExtension/custom-text-selection-tool>.

The logic processing methods used in floating tooltip

In some cases, when you modify the tooltip, you will most likely want to create your own controller to handler your tool. The code below shows commonly used methods to create a controller.

```
var tooltipLayer = this.component.getClosestComponentByType("tooltip-layer");
var textSelectionTool = tooltipLayer.getCurrentSelectionTool();
textSelectionTool.getSelectionInfo().then((selectionInfo)=>{
});
textSelectionTool.pageRender // The current rendering page object
```

The `getClosestComponentByType()` is used to get the matched tooltip layer.

The `getCurrentSelectionTool()` is used to get the text selection tool object. And the `getSelectionInfo()` is called to obtain the selected text information and the current rendering page object. The obtained text information includes:

- `page` // PDF page object
- `text` // Text contents
- `rectArray` //Text block (unit: point)

The component name of the floating tooltip

Component Name	Description
fv--text-selection-tooltip	The floating tooltip layer

Component Name	Description
fv--text-selection-tooltip-copy	Copy tool
fv--text-selection-tooltip-create-highlight	Highlight tool
fv--text-selection-tooltip-create-strikeout	Strikeout tool
fv--text-selection-tooltip-create-underline	Underline tool

Customize Internationalization Resources

Assumption

Assume you have an `assets/` in your website root directory, where you will configure the internationalization resources. Let's call this path as `websiteRoot/assets/`.

Configuration

- 1) Copy `lib/locales` inside SDK to `websiteRoot/assets/`.
- 2) Set up the `i18n` path for loading resources.

```
new UIExtension.PDFUI({  
  i18n: {  
    absolutePath: 'websiteRoot/assets/locales'  
  },  
  // the other options...  
});
```

- 3) Add more localization languages. Create a new folder in `websiteRoot/assets/locales`. The folder name should follow the language codes standard, such as `zh-CN` for Chinese, `ru_RU` for Russian.
- 4) Copy the `"ui.json"` file under `websiteRoot/assets/locales/en-US` directory into your created folder in the above step. Translate (Localize) all the entries in the `"ui.json"` file.
- 5) Set up default language.

```
new UIExtension.PDFUI({  
  i18n: {  
    absolutePath: `websiteRoot/assets/locales`,  
    lng: 'zh-CN'  
  },  
  // the other options  
})
```


Verify the configuration in developer environment.

1. Clear your browser caches to ensure the latest i18n resources will be loaded.
2. Refresh your browser, open the Network panel in DevTools, and check if the **ui.json** request url points to your custom language path. If so, it means success.

Make the Web Viewer Adaptive to the Device

Foxit PDF SDK for Web can allow the viewer to be adaptive to the device (desktop or mobile), which means if you access the web viewer in your desktop browser, it will be present using the desktop UI layout, and if you access the web viewer in your mobile browser, it will be present using the mobile UI layout. To make the project adaptive to the device, you may refer to the example **adaptive-to-the-device.html** in `examples/UIExtension/custom_appearance/` folder for detail.

Framework Integration

Foxit PDF SDK for Web Example - Angular.js

This guide shows two examples for angular. One introduces how to quickly run the out-of-the-box sample in Foxit PDF SDK for Web package, and the other presents a way to integrate Foxit PDF SDK for Web into an existing Angular/cli app.

Quickly run the out-of-the-box example for Angular

This example is built for @angular/cli app. It can be accessed at `/integrations/` inside Foxit PDF SDK for Web package.

Prerequisites

- [Nodejs](#) and [npm](#)
- [Foxit PDF SDK for Web](#)

Getting started

Enter `../integrations/angular.js/` inside Foxit PDF SDK for Web, and execute:

```
npm i
```

This step will create a `node_modules` folder and download all dependencies, copy the `lib` folder from the root folder to `/integrations/angular/src*`, and auto rename it as `foxit-lib`.

Running the example

On the shell, execute the following command to start your application:

```
npm start
```

Now you are ready to launch the app. Open your browser, navigate to `<http://localhost:4200>` to load your example.

Integrate Foxit PDF SDK for Web into existing Angular project

This integration assumes you have `@Angular/cli` app installed

Prerequisites

- [Nodejs](#) and [npm](#)
- [@angular/cli](#)
- [Foxit PDF SDK for Web](#)

Basic setup

Let's call the root folder of your exiting project as AngularJS and Foxit PDF SDK for Web as SDK.

Find the **lib** folder inside SDK, duplicate it to **AngularJS/src/** and rename it as **foxit-lib**. Besides, to correctly refer your fonts library, you also need to duplicate the **external** folder inside SDK to **AngularJS/src/foxit-lib/assets**.

Inside AngularJS, implement the following:

- 1) In the **angular.json**, update **architect/build** options of **assets**, **styles**, **extractCss**, and **architect/lint** section.

```
{
  ...
  "build": {
    "assets": [
      ...,
      {
        "glob": "**/*",
        "input": "src/foxit-lib",
        "output": "/foxit-lib",
        "ignore": ["PDFViewCtrl.*", "UIExtension.*"]
      }
    ],
    "styles": [
      "src/foxit-lib/UIExtension.css",
      "src/styles.css"
    ],
    "extractCss": true,
    ...
  }
}
"lint": {
  "builder": "...",
  "options": {
    "tsConfig": [
      //existing configuration can remain as they are
    ],
    "exclude": [
      // other dependencies you may have
      "src/foxit-lib/**/*.*"
    ]
  }
}
```

```
},
```

- 2) Update `tsconfig.app.json` to exclude the `"src/foxit-lib/**/*.ts"`.

```
{  
  ...,  
  "exclude": [  
    ...  
    ...,  
    ...,  
    "src/foxit-lib/**/*.ts"  
  ]  
}
```

Creating components

- 1) In AngularJS, run

```
ng generate component PDFViewer
```

This step will create `pdfviewer` folder and related component files under `AngularJS/src/app`. Now, you need to implement the followings in `AngularJS/src/app/`.

- 2) Place the `license-key.js` into `../pdfviewer/`. You can find the license information at `SDK/examples/`.
- 3) Update `../pdfviewer/component.ts`. For configuration details, refer to the counterpart file inside SDK.
- 4) Update `../component.html` to pass a DOM element for placing web viewer.

```
<div>  
  <app-foxitpdfviewer  
    #pdfviewer  
    class="foxit-pdf-viewer-container"  
  ></app-foxitpdfviewer>  
</div>
```

- 5) Update `component.css` to make it look as what you preferred.

```
.foxit-pdf-viewer-container {  
  display: block;  
  margin: 0 auto;  
  width: 1280px;  
  height: 1024px;  
}
```

Referencing Addons

If you are integrating Foxit PDF SDK for Web into your existing Angular project, you should read this section before continuing. You may want to check out [Addons](#) for detailed introductions.

Here we introduce three ways to reference SDK addons for Angular project, you may choose one of them based on your needs. This [Comparison](#) will help you to better understand the difference of the three ways and make a choice.

1. Reference fragmented addons

This method was used by default in past versions before version 7.2. You should open `pdfviewer.component.ts`, write the addons under `ngOnInit()` as shown below:

```
ngOnInit(){
  this.pdfui = new UIExtension.PDFUI({
    addons: [
      the_path_to_foxit_lib + '/uix-addons/file-property/addon.info.json',
      the_path_to_foxit_lib + '/uix-addons/full-screen/addon.info.json',
      // .etc
    ],
    // other options
  });
}
```

Where `the_path_to_foxit_lib` is the SDK lib folder, the path depends on the assets configuration of `angular.json`. For details, check out [Basic Setup](#).

2. Import modular addons

This method was used by default in the out-of-the-box example for Angular.

1) Install.

```
npm install -D gulp @foxitsoftware/gulp-merge-addon
```

2) Refer to `/integrations/angular/gulpfile.js` for merging addons with `gulp.task`.

3) Update the scripts section in `package.json`:

```
"scripts": {
  "postinstall": "npm run update-sdk",
  "update-sdk": "node bin/setup.js",
  "merge-addons": "gulp merge-addons",
  "start": "npm run merge-addons && ng serve",
  "build": "npm run merge-addons && ng build",
  "test": "npm run merge-addons && ng test",
  "lint": "set NODE_OPTIONS=--max-old-space-size=8192 && ng lint",
  "e2e": "ng e2e"
},
```

This way will automatically merge addons once `npm start` is successfully executed.

- 4) The import method can be seen at

```
/integrations/angular/src/app/pdfviewer/pdfviewer.component.ts.
```

3. Reference allInOne.js

The allInOne.js already combines all addons, that locates in `foxit-lib/uix-addons/`. To reference this file, open `pdfviewer.component.ts`, and update the code as follows:

```
// ...
import * as UIExtension from 'path/to/foxit-lib/UIExtension.full.js';
import * as Addons from 'path/to/foxit-lib/uix-addons/allInOne.js';
// ...
```

Under the `ngOnInit()`, pass Addons to PDFUI:

```
this.pdfui = new UIExtension.PDFUI({
  addons: Addons,
  // other options
});
```

Comparisons of loading methods

Loading methods	Configuration	HTTP Requests	Editable (i.e edit localization resources, and addon.info.json)
Fragmentized	No	n+	Yes
Modularized	Configure gulp	0	Yes, but should re-merge the addons after modification
allInOne.js	No	1	No

Note: You can rebuild allInOne.js by using our [Addons merge tools](#).

Running your Application

On the Shell, run

```
npm start
```

Awesome, all are made ready. In your browser, go to <http://localhost:4200> to load your application.

Foxit PDF SDK for Web Example - React.js

This guide shows two examples. One introduces how to quickly run the out-of-the-box sample for react.js in Foxit PDF SDK for Web package, and the other presents a way to integrate Foxit PDF SDK for Web into an existing React app created with WebPack and Babel.

Quickly run the out-of-the-box sample for react.js

Note: The root folder of Foxit PDF SDK for Web is referred as `root` in the following.

Foxit PDF SDK for Web provides a boilerplate project for React app which was created with WebPack and Babel. This example can be found at `root/integrations/` inside Foxit PDF SDK for Web package.

Overview the project structure

```
├─app/
│  ├─components/
│  │   └─PDFViewer/
│  ├─containers/
│  │   └─App/
│  ├─foxit-lib/
│  │   └─...
│  ├─app.js
│  ├─index.html
│  ├─preload.js
│  └─license-key.js
├─development/
│  ├─webpack/
│  │   └─...
│  └─setup.js
├─package.json
└─babel.config.js
```

Key directory and files descriptions:

File/Folder	Description
app/	Contains all JS and CSS files for the app.
app/components/PDFViewer/	Contains the initialization plugins for Foxit PDF SDK for Web.
app/preload.js	This entry point used to preload SDK core assets.

File/Folder	Description
app/app.js	The entry point for application.
development/	Contains automated scripts for packaging in dev mode, application initialization and etc.
package.json	Lists dependencies, version build information and etc.

Prerequisites

- [Nodejs](#) and [npm](#)
- [Foxit PDF SDK for Web](#)

Getting started

Navigate to `root/integrations/react.js/`, and execute:

```
npm run setup
```

This setup will implement the followings:

- `npm install` to install dependencies.
- `npm run update-sdk`
 - Copy `lib` folder from the root folder to the `root/integrations/react.js/app/`, and auto rename it to `foxit-lib`.
 - Copy `root/examples/license-key.js` to the `root/integrations/react.js/app/`.

Referencing Addons

If you are integrating Foxit PDF SDK for Web into your existing React project, you should read this section before continuing. You may want to check out [Addons](#) for detailed introductions.

Here we introduce three ways to reference SDK addons for React project, you may choose one of them based on your needs. This [Comparison](#) will help you to better understand the difference of the three ways and make a choice.

1. Reference fragmented addons

This method was used by default in past versions before version 7.2. You should open `components/PDFViewer/index.js`, and refer addons as shown below:

```
this.pdfui = new UIExtension.PDFUI({  
  addons: [  
    // ...  
  ]  
});
```



```
    the_path_to_foxit_lib + '/uix-addons/file-property/addon.info.json',
    the_path_to_foxit_lib + '/uix-addons/full-screen/addon.info.json',
    // .etc
  ],
  // other options
});
```

Where `the_path_to_foxit_lib` is the SDK lib folder.

2. Import modular addons

- 1) Install.

```
npm i -D @foxitsoftware/addon-loader
```

- 2) update `development/webpack/webpack.base.js` configuration:

```
{
  test: /addon\.info\.json$/,
  include: /foxit-lib/,
  use: [{
    loader: 'babel-loader',
    options: options.babelLoaderOptions
  }, '@foxitsoftware/addon-loader'],
  type: 'javascript/auto'
}
```

- 3) In `components/PDFViewer/index.js`, import `addon.info.json` for each addon:

```
import * as UIExtension from '../foxit-lib/UIExtension.full.js'
import filePropertyAddon from '../foxit-lib/uix-addons/file-property/addon.info.json';
import multiMediaAddon from '../foxit-lib/uix-addons/multi-media/addon.info.json';
import passwordProtectAddon from '../foxit-lib/uix-addons/password-protect/addon.info.json';
import redactionAddon from '../foxit-lib/uix-addons/redaction/addon.info.json';
import pathObjectsAddon from '../foxit-lib/uix-addons/path-objects/addon.info.json';
import printAddon from '../foxit-lib/uix-addons/print/addon.info.json';
import fullScreenAddon from '../foxit-lib/uix-addons/full-screen/addon.info.json';
import importFormAddon from '../foxit-lib/uix-addons/import-form/addon.info.json';
import exportFormAddon from '../foxit-lib/uix-addons/export-form/addon.info.json';
import undoRedoAddon from '../foxit-lib/uix-addons/undo-redo/addon.info.json';
import textObjectAddon from '../foxit-lib/uix-addons/text-object/addon.info.json';
```

And pass addons to the PDFUI constructor:

```
const libPath = '/foxit-lib/';
this.pdfui = new UIExtension.PDFUI({
  addons: [
    filePropertyAddon,
    multiMediaAddon,
    passwordProtectAddon,
    redactionAddon,
    pathObjectsAddon,
    printAddon,
    fullScreenAddon,
```

```
importFormAddon,
exportFormAddon,
undoRedoAddon
].concat(
  // text-object addon is disabled on mobile platform
  UIExtension.PDFViewCtrl.DeviceInfo.isMobile
    ? []
    : textObjectAddon
),
// other options
});
```

3. Reference allInOne.js

The allInOne.js already combines all addons, which locates in `foxit-lib/uix-addons/`. To refer this file, open `components/PDFViewer/index.js`, and update the code as follows:

```
// ...
import * as UIExtension from '../foxit-lib/UIExtension.full.js';
import * as Addons from '../foxit-lib/uix-addons/allInOne.js';
// ...
```

And pass parameters to the PDFUI constructor:

```
this.pdfui = new UIExtension.PDFUI({
  addons: UIExtension.PDFViewCtrl.DeviceInfo.isMobile
    ? Addons.filter(it => it.getName() !== 'textEditObject')
    : Addons,
  // other options
});
```

Comparisons of loading methods

Loading methods	Configuration	HTTP Requests	Editable (i.e edit localization resources, and addon.info.json)
Fragmentized	No	n+	Yes
Modularized	Configure gulp	0	Yes, but should re-merge the addons after modification
allInOne.js	No	1	No

Note: You can rebuild allInOne.js by using our [Addons merge tools](#).

Running the example

On the shell, execute the following command to start the development service:

```
npm start
```

Now you are ready to launch the app. Open your browser, navigate to `<http://127.0.0.1:9102/>` to load your example.

Building

```
npm run build
```

The production will be placed into `root/integrations/react.js/dist`.

Testing

```
cd ./dist && http-server -p 8080
```

Integrate Foxit PDF SDK for Web into existing Angular project

This integration assumes you have React app created with Webpack and Babel.

Prerequisites

- [Nodejs](#) and [npm](#)
- [React.js created with WebPack and Babel](#)
- [Foxit PDF SDK for Web](#)

Webpack configuration

Let's call the root folder of your existing React project and Foxit PDF SDK for Web as ReactJS and SDK.

- 1) Create and configure the following 3 files in the `ReactJS/development/webpack` folder:

- `webpack.base.js`
- `webpack.dev.js`
- `webpack.prod.js`

For the configuration details, refer to the counterpart files in `SDK/integrations/react.js/development/webpack/`. You can also directly duplicate the files to `ReactJS/development/webpack`.

- 2) Configure npm script in `package.json`

```
"script": {  
  "start": "webpack-dev-server --config development/webpack/webpack.dev.js",  
  "build": "webpack --config development/webpack/webpack.prod.js"  
}
```

Adding dependencies and entry point files

- 1) In SDK, copy the `lib` and `SDK/examples/license-key.js` to `ReactJS/app`, and change the lib name to `foxi-lib`. Besides, to correctly refer your fonts library, you also need to duplicate the `external` folder inside SDK to `ReactJS/app/foxi-lib/assets`.
- 2) Create and configure the following files in ReactJS:
 - the `babel.config.js`
 - the `../app/components/PDFViewer/index.js`
 - the `../app/containers/App/index.js`
 - the `index.html`, `app.js` and `preload.js` inside `../app/`

For the configuration details, refer to the corresponding files in SDK. You can also directly duplicate those files into the counterpart folders in ReactJS.

Running your application

```
npm run start
```

Now everything is set up. Open your browser, navigate to <http://127.0.0.1:9102/> to launch your application.

Foxit PDF SDK for Web Example - React.js created by "create-react-app"

This guide shows two examples. One introduces how to quickly run the out-of-the-box sample for react.js created by "create-react-app" in Foxit PDF SDK for Web package, and the other presents a way to integrate Foxit PDF SDK for Web into React app created by "create-react-app".

Quickly run the out-of-the-box sample for create-react-app

Note: The root folder of Foxit PDF SDK for Web is referred as `root` in the following.

Foxit PDF SDK for Web provides a boilerplate project for React app which was created by "create-react-app". This example can be found at `root/integrations/` inside Foxit PDF SDK for Web package.

Overview the project structure

```
app
├── public
│   └── index.html
├── src/
│   ├── components/
│   │   └── PDFViewer/
```

```
| |—foxit-lib/
| | |—...
| |—App.css
| |—App.js
| |—index.css
| |—index.js
| |—license-key.js
| |—preload.js
|—eslintignore
|—config-overrides.js
|—package.json
```

Key directory and files descriptions:

File/Folder	Description
src/	Contains all JS and CSS files for the app.
src/components/PDFViewer/	Contains the initialization plugins for Foxit PDF SDK for Web.
src/preload.js	This entry point used to preload SDK core assets.
src/license-key.js	The license-key.
src/app.js	The entry point for application.
config-overrides.js	Adjusts the Webpack configuration.
package.json	Lists dependencies, version build information and etc.

Prerequisites

- [Nodejs](#) and [npm](#)
- [Foxit PDF SDK for Web](#)

Getting started

- Copy root/examples/license-key.js to the root/integrations/create-react-app/app/src.
- Navigate to `root/integrations/create-react-app/app/`, and execute:

```
npm install
npm install -S @foxitsoftware/foxit-pdf-sdk-for-web-library
npm install -D copy-webpack-plugin customize-cra react-app-rewired
npm run start
```

Now everything is set up. Open your browser, navigate to <http://localhost:3000/> to launch this application.

Integrate Web SDK to react app created by "create-react-app"

Prerequisites

- [Nodejs](#) and [npm](#)
- [React.js created by create-react-app](#)
- [Foxit PDF SDK for Web](#)

Getting started

1. Create the React app with "create-react-app":

```
`npx create-react-app app`
```

2. In `app` folder, update `package.json`:

```
"scripts": {  
  "start": "react-app-rewired --max_old_space_size=8192 start",  
  "build": "react-app-rewired --max_old_space_size=8192 build",  
  "test": "react-app-rewired --max_old_space_size=8192 test --env=jsdom",  
  "eject": "react-app-rewired --max_old_space_size=8192 eject"  
},
```

3. In `app` folder, add `config-overrides.js`:

```
const CopyWebpackPlugin = require("copy-webpack-plugin");  
const { override, addWebpackPlugin, addWebpackExternals } = require('customize-cra');  
const path = require("path")  
  
const libraryModulePath = path.resolve('node_modules/@foxitsoftware/foxit-pdf-sdk-for-web-library');  
const libPath = path.resolve(libraryModulePath, 'lib');  
  
module.exports = override(  
  addWebpackPlugin(  
    new CopyWebpackPlugin({  
      patterns: [{  
        from: libPath,  
        to: 'foxit-lib',  
        force: true  
      }]  
    })  
  ),  
  addWebpackExternals(  
    'UIExtension',  
    'PDFViewCtrl'  
  )  
)
```

```
)
```

4. In `app` folder, add `.eslintignore`:

```
license-key.js
```

5. In `src` folder, add `preload.js`:

```
import preloadJrWorker from '@foxitsoftware/foxit-pdf-sdk-for-web-library/lib/preload-jr-worker';
import { licenseKey, licenseSN } from './license-key';

const libPath = "/foxit-lib/"

window.readyWorker = preloadJrWorker({
  workerPath: libPath,
  enginePath: libPath+'jr-engine/gsdk',
  fontPath: 'http://webpdf.foxitsoftware.com/webfonts/',
  licenseSN,
  licenseKey,
});
```

6. In `src/index.js` file, import `preload.js`:

```
import './preload.js'
```

7. In `src` folder, add `components/PDFViewer/index.js`:

```
import React from "react";
import * as UIExtension from '@foxitsoftware/foxit-pdf-sdk-for-web-library/lib/UIExtension.full.js';
import "@foxitsoftware/foxit-pdf-sdk-for-web-library/lib/UIExtension.css";

export default class PDFViewer extends React.Component {
  constructor() {
    super();
    this.elementRef = React.createRef();
  }

  render() {
    return <div className="foxit-PDF" ref={this.elementRef} />;
  }

  componentDidMount() {
    const element = this.elementRef.current;
    const libPath = "/foxit-lib/";
    this.pdfui = new UIExtension.PDFUI({
      viewerOptions: {
        libPath,
        jr: {
          readyWorker: window.readyWorker
        }
      },
      renderTo: element,
    });
  }
}
```

```
    appearance: UIExtension.appearances.adaptive,  
    addons: UIExtension.PDFViewCtrl.DeviceInfo.isMobile ?  
    libPath+'uix-addons/allInOne.mobile.js':  
    libPath+'uix-addons/allInOne.js'  
  });  
}  
componentWillUnmount() {  
  this.pdfui.destroy();  
}  
}
```

8. Update App.js:

```
import './App.css';  
import PDFViewer from './components/PDFViewer';  
function App() {  
  return (  
    <div className="App">  
      <PDFViewer></PDFViewer>  
    </div>  
  );  
}  
  
export default App;
```

9. Install your node_modules and run:

```
cd app  
npm install  
npm install -S @foxitsoftware/foxit-pdf-sdk-for-web-library  
npm install -D copy-webpack-plugin customize-cra react-app-rewired  
npm run start
```

10. Now everything is set up. Open your browser, navigate to <http://localhost:3000/> to launch your application.

Foxit PDF SDK for Web Example - Vue.js

This guide shows two examples. One introduces how to quickly run the out-of-the-box sample for Vue.js in Foxit PDF SDK for Web package, and the other presents a way to integrate Foxit PDF SDK for Web into an existing Vue app.

Quickly run the out-of-the-box example for Vue.js

Foxit PDF SDK for Web provides a boilerplate project for [@vue/cli](#) app. This example can be found at `/integrations/` inside Foxit PDF SDK for Web package.

Prerequisites

- [Nodejs](#) and [npm](#)
- [Foxit PDF SDK for Web](#)

Getting started

Enter `../integrations/vue.js/` inside Foxit PDF SDK for Web, and run:

```
npm run setup
```

This setup will implement the followings:

- `npm install` - install dependencies.
- `npm run update-sdk`
 - Copy `lib` folder from the root folder to the `../integrations/vue.js/src/`, and auto rename it to `foxit-lib`.
 - Copy `../examples/license-key.js` to the `../integrations/react.js/src/`.

Running the example

```
npm run serve
```

Now you are ready to launch the app. Open your browser, navigate to `<http://127.0.0.1:9103/>` to load your example.

Integrate Foxit PDF SDK for Web into existing Vue.js project

This integration assumes you have [@vue/cli](#) app installed with all default settings.

Prerequisites

- [Nodejs](#) and [npm](#)
- [@vue/cli](#)
- [Foxit PDF SDK for Web](#)

Setup

Let's call the root folder of your exiting project as VueJS and Foxit PDF SDK for Web as SDK.

- 1) In SDK, duplicate the following folder and files to `VueJS/src`, and change the lib name to `foxit-lib`.
 - The `lib` folder.
 - The `../examples/license-key.js` file.
 - The `/integrations/vue.js/src/preload.js` file

Besides, to correctly refer your fonts library, you also need to duplicate the `external` folder inside SDK to `VueJS/src/foxit-lib/assets`.

- 2) Run `npm i -D cross-env` to install `cross-env`, and add the following segments to `serve` and `build` in `VueJS/package.json`.

```
cross-env NODE_OPTIONS=--max_old_space_size=8192 NODE_ENV=development vue-cli-  
service serve  
cross-env NODE_OPTIONS=--max_old_space_size=8192 NODE_ENV=production vue-cli-  
service build
```

The purpose of this step is to avoid memory leak error.

Configuration

Add configurations onto the following files in your VueJS:

- `vue.config.js`
- `.eslintignore`

For the configuration details, refer to the counterpart files in SDK.

Component

Create `src/components/PDFViewer.vue` in your VueJS, and reference it in `src/App.vue`.

For the configuration details, refer to the counterpart files in SDK.

Referencing Addons

If you are integrating Foxit PDF SDK for Web into your existing Vue.js project, you should read this section before continuing. You may want to check out [Addons](#) for detailed introductions.

Here we introduce three ways to reference SDK addons for Vue.js project, you may choose one of them based on your needs. This [Comparison](#) will help you to better understand the difference of the three ways and make a choice.

1. Reference fragmented addons

This method was used by default in past versions before version 7.2. You should open `PDFViewer.vue`, and reference addons as shown below:

```
this.pdfui = new UIExtension.PDFUI({  
  addons: [  
    the_path_to_foxit_lib + '/uix-addons/file-property/addon.info.json',  
    the_path_to_foxit_lib + '/uix-addons/full-screen/addon.info.json',  
    // .etc
```

```
},  
  // other options  
});
```

Where `the_path_to_foxit_lib` is the SDK lib folder.

2. Import modular addons

a) Install.

```
npm i -D @foxitsoftware/addon-loader
```

b) Update `vue.config.js`, you may refer to `/integrations/vue.js/vue.config.js`.

c) In `PDFViewer.vue`, import `addon.info.json` for each addon:

```
import * as UIExtension from './foxit-lib/UIExtension.full.js'  
import filePropertyAddon from './foxit-lib/uix-addons/file-property/addon.info.json';  
import multiMediaAddon from './foxit-lib/uix-addons/multi-media/addon.info.json';  
import passwordProtectAddon from './foxit-lib/uix-addons/password-protect/addon.info.json';  
import redactionAddon from './foxit-lib/uix-addons/redaction/addon.info.json';  
import pathObjectsAddon from './foxit-lib/uix-addons/path-objects/addon.info.json';  
import printAddon from './foxit-lib/uix-addons/print/addon.info.json';  
import fullScreenAddon from './foxit-lib/uix-addons/full-screen/addon.info.json';  
import importFormAddon from './foxit-lib/uix-addons/import-form/addon.info.json';  
import exportFormAddon from './foxit-lib/uix-addons/export-form/addon.info.json';  
import undoRedoAddon from './foxit-lib/uix-addons/undo-redo/addon.info.json';  
import textObjectAddon from './foxit-lib/uix-addons/text-object/addon.info.json';
```

And pass addons to the PDFUI constructor:

```
const libPath = './foxit-lib/';  
this.pdfui = new UIExtension.PDFUI({  
  addons: [  
    filePropertyAddon,  
    multiMediaAddon,  
    passwordProtectAddon,  
    redactionAddon,  
    pathObjectsAddon,  
    printAddon,  
    fullScreenAddon,  
    importFormAddon,  
    exportFormAddon,  
    undoRedoAddon  
  ].concat(  
    // text-object addon is disabled on mobile platform  
    UIExtension.PDFViewCtrl.DeviceInfo.isMobile  
      ? []  
      : textObjectAddon  
  ),  
  // other options
```

```
});
```

3. Reference allInOne.js

The allInOne.js already combines all addons, which locates in `foxit-lib/uix-addons/`. To reference this file, open `PDFViewer.vue`, and update the code as follows:

```
// ...
import * as UIExtension from '../foxit-lib/UIExtension.full.js';
import * as Addons from '../foxit-lib/uix-addons/allInOne.js';
// ...
```

And pass parameters to the PDFUI constructor:

```
this.pdfui = new UIExtension.PDFUI({
  addons: UIExtension.PDFViewCtrl.DeviceInfo.isMobile
    ? Addons.filter(it => it.getName() !== 'textEditObject')
    : Addons,
  // other options
});
```

Comparisons of loading methods

Loading methods	Configuration	HTTP Requests	Editable (i.e edit localization resources, and addon.info.json)
Fragmentized	No	n+	Yes
Modularized	Configure gulp	0	Yes, but should re-merge the addons after modification
allInOne.js	No	1	No

Note: You can rebuild allInOne.js by using our [Addons merge tools](#).

Running your application

```
npm run serve
```

Now everything is set up. Open your browser, input the address based on you set up to launch your application.