3-Heights™
PDF Web Viewer TypeScript API

Version 6.6.0
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
<td>4</td>
</tr>
<tr>
<td>1.1</td>
<td>Description</td>
<td>4</td>
</tr>
<tr>
<td>1.2</td>
<td>API Features</td>
<td>4</td>
</tr>
<tr>
<td>1.3</td>
<td>Viewer Features</td>
<td>4</td>
</tr>
<tr>
<td>1.4</td>
<td>Supported Input and Output Formats</td>
<td>5</td>
</tr>
<tr>
<td>1.5</td>
<td>Conformance</td>
<td>5</td>
</tr>
<tr>
<td>1.6</td>
<td>Supported Browsers</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>Installation and Deployment</td>
<td>6</td>
</tr>
<tr>
<td>2.1</td>
<td>Installing Dependencies and Launching a Test Server</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>Converting the Code to JavaScript</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>License Management</td>
<td>7</td>
</tr>
<tr>
<td>3.1</td>
<td>License Installation and Management</td>
<td>7</td>
</tr>
<tr>
<td>3.2</td>
<td>Troubleshooting</td>
<td>7</td>
</tr>
<tr>
<td>3.2.1</td>
<td>License key cannot be set</td>
<td>7</td>
</tr>
<tr>
<td>3.2.2</td>
<td>The current license does not permit running the product in this environment (e.g. domain)</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>Interface Reference</td>
<td>8</td>
</tr>
<tr>
<td>4.1</td>
<td>PDF Viewer Application Program Interface</td>
<td>8</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Enumerations</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>PdfPageLayoutMode Enumeration</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>PdfFitMode Enumeration</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>PdfItemCategory Enumeration</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>PdfDestinationType Enumeration</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>AnnotationBorderStyle Enumeration</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>StampAnnotationColor Enumeration</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>PdfItemType Enumeration</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>PdfActionType Enumeration</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>SearchResultType Enumeration</td>
<td>11</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Methods</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>PdfViewerApi</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>open</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>openFDF</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>openBlob</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>openFDFBlob</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>openUri</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>openFDFUri</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>saveFile</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>close</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>forceFlushCache</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>getMaxCacheSize</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>setMaxCacheSize</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>getPageCount</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>getPageNo</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>setPageNo</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>getRotation</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>setRotation</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>getZoom</td>
<td>15</td>
</tr>
</tbody>
</table>
setZoom ................................................................. 15
getScrollMaxPosition ........................................ 16
getScrollPosition ................................................ 16
setScrollPosition ................................................. 16
setSlidingWindowSize ......................................... 16
getSlidingWindowSize ........................................ 16
goTo ............................................................... 16
isOpen ............................................................... 16
setPageLayoutMode ............................................. 17
getPageLayoutMode ............................................. 17
setFitMode .......................................................... 17
getFitMode .......................................................... 17
getBorderSize ..................................................... 17
resumeDrawing ................................................... 17
suspendDrawing ................................................. 17
renderCanvas ...................................................... 18
renderPage ........................................................ 18
setIgnoringPreferences ....................................... 18
getIgnoringPreferences ........................................ 18
getProductVersion ............................................. 18
getOutlines ........................................................ 18
getTextFragmentOnPoint ...................................... 18
getTextSelection ............................................... 19
ggetTextFromSelection ....................................... 19
ggetTextAnnotationOnPoint .................................. 19
ggetTextPopups ................................................... 19
ggetTextOpenPopups ......................................... 19
ggetTextItemsFromPage ..................................... 19
createItem .......................................................... 19
ggetItem ............................................................. 20
ggetTextItems ...................................................... 20
updateItem ........................................................ 20
deleteItem ........................................................ 20
verifyStampImage ............................................... 20
getStampInfo ...................................................... 20
hasChanges ........................................................ 20
search ................................................................. 21
setLicenseKey .................................................... 21
transformScreenPointToPdfPoint ............................. 21
transformScreenRectToPdfRect .............................. 21
transformPdfPageRectToScreenRect ....................... 22
transformPdfLengthToDeviceLength ....................... 22
ggetPageScreenRect .......................................... 22
addEventListener ............................................... 22

4.1.3 Events ............................................................. 22
firstVisiblePage .................................................. 22
lastVisiblePage .................................................... 22
rotation ............................................................. 22
zoom ................................................................. 23
fitMode ............................................................ 23
pageLayoutMode ............................................... 23
busyState .......................................................... 23
1 Introduction

1.1 Description

The 3-Heights™ PDF Web Viewer TypeScript API is a compact, high-performance, high-quality PDF viewer API with an elegant viewer implementation on top. The GUI and API are written in TypeScript and can be used as is or completely changed and customized to one's needs. The core code is written in C++ and transpiled to WebAssembly for a near native performance of all computationally intensive operations and with the use of WebWorkers a responsive GUI is guaranteed.

The API offers a multitude of navigational and display options for displaying documents. Get complete access to the low level API and adjust the viewer to your needs. When using the TypeScript code the possibilities are near endless. Change small things or completely reinvent the viewer based on the API.

If only small changes are made, you can rely on a well tested viewer GUI with intuitive controls that allow for a smooth PDF viewing experience.

The PDF Web Viewer TypeScript API has been optimized for displaying PDF/A files. For best viewing experience, it is recommended to convert PDF files using the 3-Heights™ PDF to PDF/A Converter before viewing.

1.2 API Features

- Well defined TypeScript API with interfaces and enums for all the common PDF objects such as annotations, outline items (bookmarks) or destinations
- Provides synchronous functions where possible and promises for asynchronous calls which resolve or reject always giving informative feedback to the developer
- Support of high resolution retina displays
- Cache management, control the memory consumption
- Simple transformation functions to map from screen points to PDF points
- i18n: Easily add new languages and translations to the API making the viewer usable in any language you want.
- API functions to create, change or delete annotations, including:
  - Text annotations
  - Ink annotations
  - Stamp annotations (Draft, Approved, etc.)
  - Freetext annotations
  - Highlight annotations (highlight, strike out, underline, squiggly)

1.3 Viewer Features

- Navigate manually (user action) or programmatically through a document
- Select between different fit modes: actual size, fit to width, fit to height
- Rotate and display the page
- Render thumbnails and use them for navigation
- Obtain outlines (Bookmarks)
- i18n: Easily add new languages and translations to the API making the viewer usable in any language you want.
- API functions to create, change or delete annotations, including:
  - Text annotations
  - Ink annotations
  - Stamp annotations (Draft, Approved, etc.)
Freetext annotations
Highlight annotations (highlight, strike out, underline, squiggly)
The viewer implements touch handling for mobile devices
Opening of password protected documents
Handling of unsaved changes when closing a document or opening a new one
Open files from blobs or array buffers
FDF: Separate annotations from the PDF document. Save annotations separately into and FDF or open an FDF file with a corresponding PDF file and merge the annotations into the PDF for displaying and editing.

1.4 Supported Input and Output Formats
- PDF 1.x (PDF 1.0, . . . , PDF 1.7)
- PDF 2.0
- PDF/A-1, PDF/A-2, PDF/A-3
- FDF

1.5 Conformance
Standards:
- ISO 32000-1 (PDF 1.7)
- ISO 32000-2 (PDF 2.0)
- ISO 19005-1 (PDF/A-1)
- ISO 19005-2 (PDF/A-2)
- ISO 19005-2 (PDF/A-3)

1.6 Supported Browsers
- Chrome 63+
- Firefox 55+
- Edge 41+
- Safari 11.0.3+
2 Installation and Deployment

To setup the development environment node.js with the npm package manager are needed.

2.1 Installing Dependencies and Launching a Test Server

- Unpack the contents of the zip file to a target folder
- Open a command line in the target folder
- Run `npm install`
- Enter the license key in the `src/examples/pdf-web-viewer/index.ts`
- `npm run test` will launch a local web server and load the web viewer

2.2 Converting the Code to JavaScript

To compile the TypeScript and the SCSS files run

```
npm run build
```

A script is provided that compiles the TypeScript code into deployable JavaScript code. In addition to compiling TypeScript into JavaScript it also uses the rollup module to merge all the JavaScript output files from the compilation into one single file which allows for easy deployment.
3 License Management

3.1 License Installation and Management

The license key has to be passed to the viewer constructor as the second argument (see index.html).

3.2 Troubleshooting

3.2.1 License key cannot be set

The license key cannot be set in the viewer application. The error message is: "Invalid license format."

Possible causes:
- There might be a typo in the key - or the formatting of the key is incorrect.

Solution

3.2.2 The current license does not permit running the product in this environment (e.g. domain)

Possible causes:
- The license key is not meant to be used with the current domain.

Solution
Make sure that the domain on which the viewer is running matches the domain to which the key is registered.
4 Interface Reference

The PDF Web Viewer TypeScript API offers various functionalities. The possible functionalities are listed below.

4.1 PDF Viewer Application Program Interface

In this section there is a detailed interface description of the PDF Web Viewer TypeScript interface. The methods are defined in the PdfViewerApi.ts file.

4.1.1 Enumerations

PdfPageLayoutMode Enumeration

<table>
<thead>
<tr>
<th>PdfPageLayoutMode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SINGLE_PAGE</td>
<td>Shows one single page, scrolling when reaching the top/bottom of the page will jump to the next/previous page.</td>
</tr>
<tr>
<td>ONE_COLUMN</td>
<td>Shows one column of pages, that can be scrolled through.</td>
</tr>
<tr>
<td>TWO_COLUMN_LEFT</td>
<td>Shows a column of pairs of pages, that can be scrolled through. The first page is placed left.</td>
</tr>
<tr>
<td>TWO_COLUMN_RIGHT</td>
<td>Shows a column of pairs of pages, that can be scrolled through. The first page is placed as a separate title page at the top.</td>
</tr>
<tr>
<td>TWO_PAGE_LEFT</td>
<td>Shows a pair of pages, scrolling when reaching the top/bottom of the page will jump to the next/previous page pair. The first page is placed left.</td>
</tr>
<tr>
<td>TWO_PAGE_RIGHT</td>
<td>Shows a pair of pages, scrolling when reaching the top/bottom of the page will jump to the next/previous page pair. The first page is placed as a separate title page at the top.</td>
</tr>
</tbody>
</table>

PdfFitMode Enumeration

<table>
<thead>
<tr>
<th>PdfFitMode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIT_PAGE</td>
<td>The window is zoomed to fit the whole page into the viewport.</td>
</tr>
<tr>
<td>FIT_WIDTH</td>
<td>The window is zoomed to fit the page's width into the viewport. If there are multiple columns of pages shown, the viewport will fit to that width.</td>
</tr>
<tr>
<td>ACTUAL_SIZE</td>
<td>The window is zoomed to reflect the true size of the page.</td>
</tr>
</tbody>
</table>

PdfItemCategory Enumeration
PdfItemCategory

<table>
<thead>
<tr>
<th>PdfItemCategory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANNOTATION</td>
<td>Enum representing annotation types</td>
</tr>
<tr>
<td>TEXT_FRAGMENT</td>
<td>representing text fragments</td>
</tr>
<tr>
<td>OUTLINE</td>
<td>Enum representing outline objects</td>
</tr>
<tr>
<td>DESTINATION</td>
<td>Enum representing all PDF destination types see also aboutenum:pdfdestination</td>
</tr>
<tr>
<td>CONTENT_ELEMENT</td>
<td>Enum representing content element types</td>
</tr>
</tbody>
</table>

PdfDestinationType Enumeration

<table>
<thead>
<tr>
<th>PdfDestinationType</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIT</td>
<td>Fits the viewport to the destination page</td>
</tr>
<tr>
<td>FITH</td>
<td>Fits the viewport to the horizontal dimension of the destination page</td>
</tr>
<tr>
<td>FITV</td>
<td>Fits the viewport to the vertical dimension of the destination page</td>
</tr>
<tr>
<td>FITR</td>
<td>Fits the viewport to the destination rectangle</td>
</tr>
<tr>
<td>FITB</td>
<td>Fits the viewport to the bounding box of the destination page</td>
</tr>
<tr>
<td>FITBH</td>
<td>Fits the viewport to the horizontal dimension of the destination page bounding box</td>
</tr>
<tr>
<td>FITBV</td>
<td>Fits the viewport to the vertical dimension of the destination page bounding box</td>
</tr>
<tr>
<td>XYZ</td>
<td>Fits the viewport to be at a specific location with a specific zoom level</td>
</tr>
</tbody>
</table>

AnnotationBorderStyle Enumeration

<table>
<thead>
<tr>
<th>AnnotationBorderStyle</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOLID</td>
<td>A solid rectangle surrounding the annotation</td>
</tr>
<tr>
<td>DASHED</td>
<td>A dashed rectangle surrounding the annotation</td>
</tr>
<tr>
<td>BEVELED</td>
<td>A simulated embossed rectangle that appears to be raised above the surface of the page</td>
</tr>
<tr>
<td>INSET</td>
<td>A simulated engraved rectangle that appears to be recessed below the surface of the page</td>
</tr>
<tr>
<td>UNDERLINE</td>
<td>A single line along the bottom of the annotation rectangle</td>
</tr>
</tbody>
</table>

StampAnnotationColor Enumeration
### Stamp Color

<table>
<thead>
<tr>
<th>Colors</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GREEN</td>
<td>Colors the background of the stamp in green</td>
</tr>
<tr>
<td>RED</td>
<td>Colors the background of the stamp in red</td>
</tr>
<tr>
<td>BLUE</td>
<td>Colors the background of the stamp in blue</td>
</tr>
</tbody>
</table>

### PdfItemType Enumeration

<table>
<thead>
<tr>
<th>Item Types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNKNOWN</td>
<td>Unknown item type</td>
</tr>
<tr>
<td>TEXT</td>
<td>Text annotation (Sticky Note)</td>
</tr>
<tr>
<td>LINK</td>
<td>Link annotation</td>
</tr>
<tr>
<td>FREE_TEXT</td>
<td>Freetext annotation</td>
</tr>
<tr>
<td>LINE</td>
<td>Line annotation (not supported)</td>
</tr>
<tr>
<td>SQUARE</td>
<td>Square annotation (not supported)</td>
</tr>
<tr>
<td>CIRCLE</td>
<td>Circle annotation (not supported)</td>
</tr>
<tr>
<td>POLYGON</td>
<td>Polygon annotation (not supported)</td>
</tr>
<tr>
<td>POLY_LINE</td>
<td>Poly line annotation (not supported)</td>
</tr>
<tr>
<td>HIGHLIGHT</td>
<td>Highlight annotation</td>
</tr>
<tr>
<td>UNDERLINE</td>
<td>Underline annotation</td>
</tr>
<tr>
<td>SQUIGGLY</td>
<td>Squiggy annotation</td>
</tr>
<tr>
<td>STRIKE_OUT</td>
<td>Strike-out annotation</td>
</tr>
<tr>
<td>STAMP</td>
<td>Stamp annotation</td>
</tr>
<tr>
<td>CARET</td>
<td>Caret annotation (not supported)</td>
</tr>
<tr>
<td>INK</td>
<td>Ink annotation</td>
</tr>
<tr>
<td>POPUP</td>
<td>Popup annotation</td>
</tr>
<tr>
<td>FILE_ATTACHMENT</td>
<td>File attachment annotation (not supported)</td>
</tr>
<tr>
<td>SOUND</td>
<td>Sound annotation (not supported)</td>
</tr>
<tr>
<td>MOVIE</td>
<td>Movie annotation (not supported)</td>
</tr>
</tbody>
</table>
### Item Types

<table>
<thead>
<tr>
<th>Item Types</th>
<th>Annotation Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIDET</td>
<td>Widet annotation (not supported)</td>
</tr>
<tr>
<td>SCREEN</td>
<td>Screen annotation (not supported)</td>
</tr>
<tr>
<td>PRINTER_MARK</td>
<td>Printer mark annotation (not supported)</td>
</tr>
<tr>
<td>TRAP_NET</td>
<td>Trap net annotation (not supported)</td>
</tr>
<tr>
<td>WATERMARK</td>
<td>Watermark annotation (not supported)</td>
</tr>
<tr>
<td>THREED</td>
<td>3D annotation (not supported)</td>
</tr>
</tbody>
</table>

### PdfActionType Enumeration

#### Action Types

<table>
<thead>
<tr>
<th>Action Types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNKNOWN</td>
<td>Unknown action type</td>
</tr>
<tr>
<td>GO_TO</td>
<td>Go to destination in the current document</td>
</tr>
<tr>
<td>URI</td>
<td>Resolve a uniform resource identifier</td>
</tr>
</tbody>
</table>

### SearchResultType Enumeration

#### Search Result Types

<table>
<thead>
<tr>
<th>Result Types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
<td>When the search was successful and a hit was found</td>
</tr>
<tr>
<td>NO_RESULT</td>
<td>When the search was successful but no hit was found</td>
</tr>
<tr>
<td>END</td>
<td>The end of the search (when wrapping is disabled)</td>
</tr>
</tbody>
</table>

### 4.1.2 Methods

#### PdfViewerApi

**Method:** PdfViewerApi()

Constructor of Interfacing class PdfViewerApi, instantiates an empty viewer, which can now be configured and used to open files. The constructor listens to an event called memorywarning which is emitted by the Cordova plugin cordova-plugin-memory-warning. The event will call forceFlushCache.

**open**

**Method:** Promise<void> open(Uint8Array buffer, string password?)

Constructor of Interfacing class PdfViewerApi, instantiates an empty viewer, which can now be configured and used to open files. The constructor listens to an event called memorywarning which is emitted by the Cordova plugin cordova-plugin-memory-warning. The event will call forceFlushCache.
Open a new PDF file.

**Parameters:**

*buffer*  [Uint8Array]  A buffer holding the PDF in memory

*password?*  [string]  The password needed to decrypt de PDF

**Note:** Any previously opened file must be closed first.

### openFDF

**Method:** Promise<void> openFDF(Uint8Array pdfBuffer, Uint8Array fdfBuffer, string password?)

Open a PDF file with an FDF file. The two files are opened to build a merged file such that the annotations defined in the FDF file are then contained in the PDF file.

**Parameters:**

*pdfBuffer*  [Uint8Array]  A buffer holding the PDF in memory

*fdfBuffer*  [Uint8Array]  A buffer holding the FDF in memory

*password?*  [string]  The password needed to decrypt the PDF

**Note:** Any previously opened file must be closed first.

### openBlob

**Method:** Promise<void> openBlob(Blob blob, string password?)

Open a PDF file from a blob.

**Parameters:**

*blob*  [Blob]  The blob of the file.

*password?*  [string]  The password needed to decrypt the PDF.

### openFDFBlob

**Method:** Promise<void> openFDFBlob(Blob pdfBlob, Blob fdfBlob, string password?)
Parameters:


password? [string] The password needed to decrypt the PDF.

Returns:

A promise returning void if resolved or returning an error message if rejected.

Open a PDF file with an FDF file from blobs. The two files are opened to build a merged file such that the annotations defined in the FDF file are then contained in the PDF file.

openUri

Method: Promise<void> openUri(string pdfUri, string password?, string pdfAuthorization?)

Parameters:

pdfUri [string] URI string where the PDF is hosted

password? [string] The password needed to decrypt the PDF.

pdfAuthorization? [string] Authentication token of the PDF file for HTML authentication of the form "<type> <credentials>".

Returns:

A promise returning void if resolved or returning an error message if rejected.

Open a PDF file via URI directly from a host.

openFDFUri

Method: Promise<void> openFDFUri(string pdfUri, string fdfUri, string password?, string pdfAuthorization?, string fdfAuthorization?)

Parameters:

pdfUri [string] URI string where the PDF is hosted

fdfUri [string] URI string where the associated FDF is hosted

password? [string] The password needed to decrypt the PDF.

pdfAuthorization? [string] Authentication token of the PDF file for HTML authentication of the form "<type> <credentials>".
**fdfAuthorization?**  [string]  Authentication token of the FDF file for HTML authentication of the form "<type> <credentials>".

**Returns:**
A promise returning void if resolved or returning an error message if rejected

Open a PDF file with an FDF file from URIs. As with blobs the viewer will merge the two documents into one such that the annotations are then contained in the PDF in memory. Currently this means that the complete PDF and FDF have to be loaded to memory.

**saveFile**

**Method:**  Promise<Uint8Array>  saveFile(bool  asFdf)

**Parameter:**

**asFdf**  [bool]  Whether the file should be saved as a PDF containing the document and the annotations, or a FDF containing **ONLY** the annotations.

Save the currently opened PDF file.

**close**

**Method:**  Promise<void>  close()

**Returns:**
A promise returning nothing if resolved and an error message if rejected.

Close the currently opened document.

**forceFlushCache**

**Method:**  forceFlushCache()

Clear the rendered pages cache to free memory.

**getMaxCacheSize**

**Method:**  getMaxCacheSize()

Get the current maximum size of the rendered pages cache.

**setMaxCacheSize**

**Method:**  setMaxCacheSize(number  cacheSize)
The viewer caches bitmaps of rendered pages. When the cache reaches the size limit it will remove pages that are not in the visible range.

Changing zoom invalidates the bitmaps in the cache.

Setting a custom sliding window size has priority over cache size. This means that the cache size can be bigger than the set maximum cache size because it will keep rendered pages within the sliding window in the cache.

**getPageCount**

<table>
<thead>
<tr>
<th>Method:</th>
<th>getPageCount()</th>
</tr>
</thead>
</table>

Returns the total number of pages in the opened document.

**getPageNo**

<table>
<thead>
<tr>
<th>Method:</th>
<th>getPageNo()</th>
</tr>
</thead>
</table>

Returns the page number of the first visible page.

**setPageNo**

<table>
<thead>
<tr>
<th>Method:</th>
<th>setPageNo(number pageNo)</th>
</tr>
</thead>
</table>

Moves the viewport to show the provided page number as first visible page on the viewport.

**getRotation**

<table>
<thead>
<tr>
<th>Method:</th>
<th>getRotation()</th>
</tr>
</thead>
</table>

Returns the rotation of the viewport.

**setRotation**

<table>
<thead>
<tr>
<th>Method:</th>
<th>setRotation(number rotation)</th>
</tr>
</thead>
</table>

Rotate the viewport by the provided amount of degrees.

**getZoom**

<table>
<thead>
<tr>
<th>Method:</th>
<th>getZoom()</th>
</tr>
</thead>
</table>

Returns the current zoom factor of the viewport.

**setZoom**

<table>
<thead>
<tr>
<th>Method:</th>
<th>setZoom(number zoom)</th>
</tr>
</thead>
</table>
Set the zoom for the viewport.

**getScrollMaxPosition**

<table>
<thead>
<tr>
<th>Method:</th>
<th>getScrollMaxPosition()</th>
</tr>
</thead>
</table>

Returns the maximum possible scroll position of the scrollbar.

**getScrollPosition**

<table>
<thead>
<tr>
<th>Method:</th>
<th>getScrollPosition()</th>
</tr>
</thead>
</table>

Get the current scroll position of the viewport with regard to the document/page.

**setScrollPosition**

<table>
<thead>
<tr>
<th>Method:</th>
<th>setScrollPosition()</th>
</tr>
</thead>
</table>

Set the scroll position of the viewport. Will be clamped to 0 and getScrollMaxPosition.

**setSlidingWindowSize**

<table>
<thead>
<tr>
<th>Method:</th>
<th>setSlidingWindowSize(number slidingWindowSize)</th>
</tr>
</thead>
</table>

By default the window size is 1. This means one page before the first visible page and one page after the last visible page will be rendered. Changing this value will influence cache behaviour as sliding window parameter take priority over the cache size.

**Note:** iOS 12 has memory restrictions on the canvases. Canvases cannot take up more than 256 MB of memory otherwise the program will crash. This means the sliding window can not be too big.

**getSlidingWindowSize**

<table>
<thead>
<tr>
<th>Method:</th>
<th>getSlidingWindowSize(number cacheSize)</th>
</tr>
</thead>
</table>

**goTo**

<table>
<thead>
<tr>
<th>Method:</th>
<th>goTo(PdfDestination destination)</th>
</tr>
</thead>
</table>

Navigate to the provided destination in the document.

**isOpen**

<table>
<thead>
<tr>
<th>Method:</th>
<th>isOpen()</th>
</tr>
</thead>
</table>
Returns `true` if a document is opened. A file counts as open as soon as the open operation has completed successfully.

**setPageLayoutMode**

**Method:** `setPageLayoutMode(PdfPageLayoutMode mode)`

Set the page layout mode.

**getPageLayoutMode**

**Method:** `getPageLayoutMode()`

Get the currently set page layout mode.

**setFitMode**

**Method:** `setFitMode(PdfFitMode fitMode)`

Set the fit mode of the viewer.

**getFitMode**

**Method:** `getFitMode()`

Get the currently set fit mode.

**getBorderSize**

**Method:** `getBorderSize()`

Get the size of the border allocated around each page. Distance between 2 pages is effectively twice the border size.

**resumeDrawing**

**Method:** `resumeDrawing()`

Resumes drawing into the canvas.

**suspendDrawing**

**Method:** `suspendDrawing()`

Stop the viewer from updating the canvas. Navigating is still possible but the viewport will not update until drawing has been resumed `resumeDrawing`. 
**renderCanvas**

**Method:** `renderCanvas(CanvasRenderingContext2D ctx)`

Renders the current viewport into the provided canvas. In the sample implementation this call is tied to the callback provided to the requestAnimationFrame function.

**renderPage**

**Method:** `Promise<PageImage> renderPage(number pageNo, number width, number height)`

Explicitly render a page with the provided dimensions. The width and height will be adjusted to the aspect ratio of the page, fitting to the smaller of the two values.

**setIgnoringPreferences**

**Method:** `setIgnoringPreferences(boolean ignore)`

Set whether when opening a new PDF file the in the file embedded preferences for the startup viewport are ignored (first shown page, fitmode and zoom).

**getIgnoringPreferences**

**Method:** `getIgnoringPreferences()`

Get whether embedded preferences in the PDF file are ignored during opening a new file or not.

**getProductVersion**

**Method:** `getProductVersion()`

Get the product version of the viewer.

**getOutlines**

**Method:** `Promise<OutlineItem[]> getOutlines(OutlineItem parent)`

Load the child outlines of a given parent. Can be used to call recursively on children which have descendants. If parent is equal to `null` the root of the outlines will be loaded.

**getTextFragmentOnPoint**

**Method:** `TextFragment getTextFragmentOnPoint(PdfPoint point)`

Obtain the text fragment which contains the point.
**getTextSelection**

Method: `TextSelectionRect[] getTextSelection(PdfPoint startPoint, PdfPoint endPoint)`

Get the rectangles of the text contained between the two points.

**getTextFromSelection**

Method: `string getTextFromSelection(PdfRect[] selection)`

Get the text contained in the provided rectangles.

**getAnnotationOnPoint**

Method: `Annotation getAnnotationOnPoint(PdfPoint point, boolean onlySelectable)`

Returns the topmost annotation at the provided PdfPoint. If onlySelectable is `true` only markup annotations are returned otherwise the function returns `null`.

**getPopups**

Method: `Annotation[] getPopups(number firstPage, number lastPage)`

Obtain all annotations with popups from first page to last page.

**getOpenPopups**

Method: `Annotation[] getOpenPopups(number firstPage, number lastPage)`

Obtain all annotations which have opened popups.

**getItemsFromPage**

Method: `Promise<PdfItemsOnPage> getItemsFromPage(number page, PdfItemCategory category)`

Loads the items of PdfItemCategory for a given page.

**createItem**

Method: `createItem(AnnotationArgs item)`

Create an item. <TODO: Finalize API>
**getItem**

**Method:** `ReturnObject getItem(number id)`

Obtain the item with the given ID. If the retrieval was successful, the item is contained in the `value` property of the `ReturnObject` object.

**getItems**

**Method:** `getItems(number[] ids)`

Obtain a list of all the items of the provided IDs. Each entry of the list is a `ReturnObject`.

**updateItem**

**Method:** `Promise<PdfItem> updateItem(PdfItem item)`

Update the editable properties of an item (e.g. annotation)

**deleteItem**

**Method:** `Promise<void> deleteItem(PdfItem item)`

Delete this item.

**verifyStampImage**

**Method:** `verifyStampImage(string name, Uint8Array image)`

Verify that the image can be added to the viewer as a stamp.

**getStampInfo**

**Method:** `getStampInfo(StampInfoArgs stampArgs)`

Get information about a particular stamp. Used for text stamps to retrieve the aspect ratio of the stamp.

**hasChanges**

**Method:** `boolean hasChanges()`

Returns true if there are unsaved changes in the document (modified, added or deleted annotations). A resolved save operation resets the state to `false` even if the returned buffer has been discarded.
search

**Method:** Promise/SearchResult> search(string toSearch, number startPage, number startIndex, boolean reverse, boolean caseSensitive, boolean wrappingSearch, boolean regex)

**Parameters:**
- **toSearch** [string]  String to be searched for
- **startPage** [number]  Page on which to start the search
- **startIndex** [number]  Where the search starts on a given page
- **reverse** [boolean]  If true the search will start on the startPage and go backwards from there
- **caseSensitive** [boolean]  Enables case sensitive search
- **wrappingSearch** [boolean]  When wrapping is enabled once hitting the last page it will wrap around to the first page and continue from there
- **regex** [boolean]  When enabled regex expressions can be used for the search

**Returns:**
TODO add about reference

setLicenseKey

**Method:** Promise<void> setLicenseKey(string licenseKey)

Set the license key for the viewer

transformScreenPointToPdfPoint

**Method:** ScreenPointToPdfPointResult transformScreenPointToPdfPoint(Point point, number page?, boolean guaranteePointIsOnPage)

Transforms a point from screen coordinates to PDF page coordinates. This is useful when either creating something to provide exact coordinates or testing whether a pdf object (e.g. annotation) can be found at a certain position.

transformScreenRectToPdfRect

**Method:** PdfRect transformScreenRectToPdfRect(Rect rect, number page)

Convert a rectangle from device coordinates to PDF coordinates. This function assumes that the screen rectangle is contained within one page.
transformPdfPageRectToScreenRect

**Method:** Rect transformPdfPageRectToScreenRect(PdfRect pdfRect)

Transforms a PDF rectangle into device coordinates.

**transformPdfLengthToDeviceLength**

**Method:** number transformPdfLengthToDeviceLength(number pdfLength)

Transforms a PDF length into device length.

ggetPageScreenRect

**Method:** getPageScreenRect(number pageNo)

Get the screen rectangle of a given page to know the page rectangle in device coordinates.

addEventListener

**Method:** addEventListener(string eventName, function callback)

Adds a call back to an event. Whenever the event is fired, the callback will be executed. See [Events](#) for events that can be subscribed to.

### 4.1.3 Events

Subscribe to events via the `addEventListener`. Following events can be subscribed to:

**firstVisiblePage**

**Event:** number firstVisiblePage

Fires when the first visible page number changes.

**lastVisiblePage**

**Event:** number lastVisiblePage

Fires when the last visible page number changes.

**rotation**

**Event:** number rotation

Fires when the view rotation changes.
zoom

**Event:** `number zoom`

Fires when the magnification level of the viewport changes.

fitMode

**Event:** `PdfFitMode fitMode`

Fires when the fit mode changes.

pageLayoutMode

**Event:** `PdfPageLayoutMode pageLayoutMode`

Fires when the page layout mode changes.

busyState

**Event:** `boolean busyState`

Fires whenever the busy state of the viewer changes. The viewer is considered busy when it's rendering (both implicit rendering calls and explicit rendering calls from `renderPage`) or saving. `true`

canvasInvalidated

**Event:** `boolean canvasInvalidated`

Fires whenever a viewport changing method (implicit or explicit) has been called. Even if the viewport ultimately didn't change.

error

**Event:** `Error error`

This event fires on two occasions:
- Viewer crashed and cannot recover
- `getItemsFromPage` has been called with an invalid category. The viewer cannot reject any promise since it does not know which promise to reject thus firing an error.

itemCreated

**Event:** `PdfItem itemCreated`

Fires when an item has been created.
itemUpdated

**Event:** PdfItem itemUpdated

Fires when an item has been updated.

itemDeleted

**Event:** DeletedItem itemDeleted

Fires when an item has been deleted.

pageChanged

**Event:** number pageChanged

Fires when the content of a page has been changed (e.g. an annotation was added, edited or deleted).

outlinesLoaded

**Event:** OutlineItem[] outlinesLoaded

Fires when the outlines

### 4.2 Annotation Flags

Annotations possess a flag attribute which controls the annotation’s behaviour in different contexts such as printing, viewing, editing.

The GUI for example prohibits a locked annotation from being changed by disabling these editing options. The flags are a bit mask and using following table shows the different values for the different flags.

Currently supported flags are: Locked & ReadOnly

<table>
<thead>
<tr>
<th>Bit position</th>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Invisible</td>
<td>This flag only applies to annotations which do not belong to one of the standard annotation types. If set, the annotation will not be rendered nor printed even if the Print flag is set.</td>
</tr>
<tr>
<td>2</td>
<td>Hidden</td>
<td>If set, do not render the annotation or allow it to interact with the user, regardless of its annotation type or whether an annotation handler is available.</td>
</tr>
<tr>
<td>3</td>
<td>Print</td>
<td>If set, print the annotation when the page is printed unless the Hidden flag is also set. If clear, never print the annotation, regardless of whether it’s rendered on the screen.</td>
</tr>
</tbody>
</table>
Annotation Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>NoZoom</td>
</tr>
<tr>
<td>5</td>
<td>NoRotate</td>
</tr>
<tr>
<td>6</td>
<td>NoView</td>
</tr>
<tr>
<td>7</td>
<td>ReadOnly</td>
</tr>
<tr>
<td>8</td>
<td>Locked</td>
</tr>
<tr>
<td>9</td>
<td>ToggleNoView</td>
</tr>
<tr>
<td>10</td>
<td>LockedContents</td>
</tr>
</tbody>
</table>

4.3 Limitations of the 3-Heights™ PDF Web Viewer TypeScript API

Due to the restrictions given in the web environment, there are some limitations to the capabilities of the PDF Web Viewer TypeScript API, as listed below.

4.3.1 Color Management

The PDF Web Viewer TypeScript API uses an RGB colorspace for rendering and converts all resources into this colorspace for rendering. When viewing the rendered output on a display screen this is not noticeable, as such screens can only display additive color (RGB colors) anyway. However there may be some noticeable differences when printing out the rendered output, due to printed pages using subtractive color (reflected light, e.g. CMYK colors).

**Effect:** The colors of non-rgb content (e.g. CMYK images) can differ slightly from the original when printing rendered output to physical printer.

4.3.2 System Fonts

Many PDF's rely on not embedding the used fonts in the pdf but instead referring to a common font, which is assumed to be installed on the local computer. Due to Browsers protection mechanisms a downloaded script like the PDF Web Viewer TypeScript API is not allowed to directly access these fonts. Thus the PDF Web Viewer TypeScript API analyzes the referenced fonts and tries to replace them with a standard font, which is packaged with the PDF Web
Viewer TypeScript API. However these standard fonts are limited and fonts which don’t use a standard encoding or use uncommon glyphs (e.g. foreign alphabets such as cyrillic) cannot always be properly replaced.

**Effect:** Text in non-embedded fonts which uses special glyphs may be rendered using wrong glyphs or not be rendered at all.

**Note:** This issue can be avoided by converting all files to PDF/A before viewing. This ensures that all fonts are embedded in the PDF file.

### 4.3.3 Range Requests

When using `openUri` the viewer tries to open large PDF files in parts to reduce the initial loading time. If the host of the PDF does not support range requests the complete file will be loaded.

There are certain flags that have to be set to allow range requests:

<table>
<thead>
<tr>
<th><strong>Range Request Settings</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key</strong></td>
</tr>
<tr>
<td>Access-Control-Expose-Headers</td>
</tr>
<tr>
<td>Access-Control-Allow-Headers</td>
</tr>
</tbody>
</table>
5 Users’s Guide

5.1 JavaScript Promises

To provide a response API the PDF Web Viewer TypeScript API works with asynchronous calls where needed. In these instances JavaScript promises are used where the caller can use the promise and define behaviours in case of success (the promise resolves) or failure (the promise is rejected).

The promises are handled by the PdfWebViewerAPICallbackHandler class. After an asynchronous call has finished the PdfWebViewerAPICallbackHandler will either resolve or reject the promise.

5.2 Rich Text for Free Text Annotations

Free text annotations can either have plain text content or rich text content. This section specifies the format of rich text content that the 3-Heights™ PDF Web Viewer TypeScript API supports.

Rich text content strings in PDF have an XHTML format with a restricted set of CSS2 styles. The following describes all allowed XML elements (in blue) and their allowed XML attributes (in green).

body The unique root element. Should contain one paragraph (p).

   xmlns (Required) Value "http://www.w3.org/1999/xhtml"
   xmlns:xfa (Required) Value "http://www.xfa.org/schema/xfa-data/1.0/"
   xfa:APIVersion (Required) Value "Acrobat:18.11.0"
   xfa:spec (Required) Value "2.0.2"
   style (Optional) Value see Supported CSS2 Styles.

p A paragraph element. In contrast to web-browser-style rendering of XHTML, a paragraph does not induce a line break. Can contain spans (span).

   style (Optional) Value see Supported CSS2 Styles.

span An element that defines a common style for its content. Can contain spans (span).

   style (Optional) Value see Supported CSS2 Styles.

b The enclosed text behaves as if style="font-weight:bold".

i The enclosed text behaves as if style="italic".

br A line break. This element must always occur in the self-closed form (<br/>) and is not allowed to have any attributes.

5.2.1 Supported CSS2 Styles

Rich text strings in PDF support a restricted set of CSS2 (Cascading Style Sheet) styles. All styles must be specified in a style attribute as follows:

   style="<key>:<value>;<key>:<value>;;...

The following describes the supported <key>s and their possible <value>s:
**text-align** Specifies the text alignment. Possible values are:
- left
- right
- center
- justify

Example: `style="text-align:left"

**text-decoration** Specifies strike-through and underline text styles. Styles are combined by specifying multiple of the following values separated spaces:
- line-through: Draws a horizontal line through the words.
- underline or word: Underlines words either in a continuous line or as interrupted lines, one for each word, respectively.
- double: Draws the underline (continuous or word-wise) as a double line.

Example: `style="text-decoration:underline line-through"

**color** Specifies the color for the text and text decoration (strike-through and underline). The value must have the following form: `#rrggbb`, where `rr`, `gg`, and `bb` are 2-digit hexadecimal numbers ranging from 00 to ff that indicate the red, green, and blue value of the color in an RGB color-space.

Example: `style="color:#a2358c"

**font-family** Specifies the font to be used. Currently the following values are supported:
- Helvetica
- Times
- Courier
- Symbol
- ZapfDingbats

Example: `style="font-family:Helvetica"

**font-size** Specifies the size of the font in the format `<number>pt`, where `<number>` is a floating point number.

Example: `style="font-size:12pt"

**font-style** The only supported values are italic and normal.

Example: `style="font-style:italic"

**font-weight** The only supported values are bold and normal.

Example: `style="font-weight:bold"

**font** This is an abbreviated form for setting font-style, font-weight, font-size, and font-family in a single style value. The values for the above four styles can be specified in any order, separated by spaces.

Example: `style="Helvetica 14pt bold"

The default style settings are as follows:
- text-align:left
- font-family:Helvetica
- font-size:12pt
- font-style:normal
- font-weight:normal
- color:#000000
6 Version History

Some of the documented changes below may be preceded by a marker that specifies the interface technologies the change applies to. E.g. [C, Java] applies to the C and the Java interface.

6.1 Changes in Version 6

- **New** method `openUri` for opening a PDF from a given URI. When the file is larger than 256 KB the viewer tries to load the PDF piecewise.
- **New** method `openFDFUri` for opening a PDF with an associated FDF given their URIs. This method circumvents the piecewise loading.
- `openUri` and `openFDFUri` accept now also authentication tokens for PDFs protected by authentication mechanisms.

6.2 Changes in Version 5

- **New** Option: `promptOnUnsavedChanges`. When set to true a warning will show up when the user tries to navigate away from the viewer (also closing the tab or window) while the currently opened document has unsaved changes.
- **New** additional supported operating system: Windows Server 2019.
- **New** API functions to customize the sliding window size. The sliding window pre-renders pages outside the visible area for the current zoom level in order to improve the viewing experience by reducing the time to wait until a page has been rendered.
- **New**: annotation flags exposed at API level. For possible flags please refer to the manual.

6.3 Changes in Version 4.12

- **New** HTTP proxy setting in the GUI license manager.

6.4 Changes in Version 4.11

- **New** support for reading PDF 2.0 documents.

6.5 Changes in Version 4.10

- **Improved** robustness against corrupt input PDF documents.
- **Improved** annotation appearance generation for polyline, squiggly, and stamp annotations.
- [C] **Clarified** Error handling of `TPdfStreamDescriptor` functions.

6.6 Changes in Version 4.9

- **Improved** support for and robustness against corrupt input PDF documents.
- **Improved** repair of embedded font programs that are corrupt.
- **New** support for OpenType font collections in installed font collection.
- **Improved** metadata generation for standard PDF properties.
- [C] **Changed** return value `pfGetLength` of `TPDFStreamDescriptor` to `pos_t`.

---

1 This has no effect on neither the .NET, Java, nor COM API
6.7 Changes in Version 4.8

- **Improved** creation of annotation appearances to use less memory and processing time.
- **Added** repair functionality for TrueType font programs whose glyphs are not ordered correctly.
7 Licensing, Copyright, and Contact

PDF Tools AG is a world leader in PDF (Portable Document Format) software, delivering reliable PDF products to international customers in all market segments.

PDF Tools AG provides server-based software products designed specifically for developers, integrators, consultants, customizing specialists and IT-departments. Thousands of companies worldwide use our products directly and hundreds of thousands of users benefit from the technology indirectly via a global network of OEM partners. The tools can be easily embedded into application programs and are available for a multitude of operating system platforms.

Licensing and Copyright The 3-Heights™ PDF Web Viewer TypeScript API is copyrighted. This user’s manual is also copyright protected; it may be copied and given away provided that it remains unchanged including the copyright notice.

Contact
PDF Tools AG
Kasernenstrasse 1
8184 Buchenbülach
Switzerland
http://www.pdf-tools.com
pdfsales@pdf-tools.com