User Manual

3-Heights™ PDF Viewer API R2

Version 4.9.11.0

PDF-TOOLS.COM
Premium PDF Technology

Contents

1 Introduction

1.1 Description

The 3-Heights™PDF Viewer API R2 is a compact, high-performance, high-quality PDF viewer. It offers a multitude of navigational and display options for displaying documents. This viewer is mainly characterized by its increased performance and its uniform and simple interface.

1.2 Features

1.2.1 Viewing Features

- Display PDF file in single page or multi-page mode
- Enter password to decrypt PDF documents
- Query the number of pages in a document
- Set scalability (page size, page width, actual size, any size)
- Reproduce documents with Chinese, Japanese and Korean fonts (CJK)
- Read document from file or memory
- Double Buffering
- Set number of pages per window
- Rotate and display the page
- Support for "FDF" file format

1.2.2 Navigation Features

- Search PDF documents for a specific string of text or for a regular expression
- Show and hide windows for bookmarks and page navigation
- Jump to a bookmark's location
- Move windows vertically and horizontally using the mouse or keyboard
- Zoom in and out using the mousewheel, a numerical zoomfactor or a zooming rectangle
- Freely select any page in the document for display
- Highlight rectangle in color

1.2.3 Supported Input Formats

- PDF 1.0 - 1.7, PDF/A

1.2.4 Conformance

Standards:

- ISO 19005-1 (PDF/A-1)
- ISO 19005-2 (PDF/A-2)
- ISO 32000-1 (PDF 1.7),

1.3 Interfaces

There is a Java Swing, a .NET WPF and .NET Windows Forms interface available. The .NET interfaces use .NET framework version 4.0. However the use of .Net framework version 4.5.2 or newer is recommended, as the support lifecycle for older versions has ended.

1.4 Operating Systems

The 3-Heights™ PDF Viewer API R2 is available for the following operating systems:

- Windows 7, 8, 8.1, 10 – 32 and 64 bit
- Windows Server 2008, 2008 R2, 2012, 2012 R2, 2016 – 32 and 64 bit

2 Installation and Deployment

2.1 Windows

The 3-Heights™ PDF Viewer API R2 comes as a ZIP archive containing various files including runtime binary executable code, files required for the developer, documentation and license terms.

1. Download the ZIP archive of the product from your download account at https://www.pdf-tools.com.
2. Unzip the file using a tool like WinZip available from WinZip Computing, Inc. at http://www.winzip.com to a directory on your hard disk where your program files reside (e.g. `C:\Program Files\PDF Tools AG`)
3. Check the appropriate option to preserve file paths (folder names). The unzip process now creates the following subdirectories:

Subdirectory	Description
`bin`	Contains the runtime executable binary code.
`bin\Fonts`	Contains required standard fonts and the font mapping file (see Fonts).
`bin\Icc`	Contains color profiles and links to download additional color profiles
`doc`	Contains documentation files.
`jar`	Contains java archive files for java components.
`lib`	Contains the object file library to include in your C/C++ project.
`samples`	Contains sample programs in various programming languages

4. Optionally register your license key using the License Management.
5. Identify which interface you are using. Perform the specific installation steps for that interface described in chapter Interfaces

2.2 Interfaces

2.2.1 .NET Interface

The two .NET interfaces for WPF and Windows Forms do not require any further installation. Simply reference the DLLs in the project of your application. `bin\PdfViewerAPI.dll` and `bin\PdfViewerCSharpAPI.dll` are required for both .NET interfaces. In addition either `bin\PdfViewerWPF.dll` or `bin\PdfViewerWinForms.dll` is required, depending on which interface is being used.

Note, that referencing the `.xml` documentation files contained in the `bin\` folder can also be useful when using an Integrated Development Environment (IDE). They contain additional documentation information about the interface that will be displayed to the user by the IDE.

Do ensure to use .NET framework version 4.0 or newer for the application that uses the dll's.

2.2.2 Java Interface

The available Java Interface uses Swing and does not require any further installation. Simply reference the java archive file `jar\PdfViewerAWT.jar` as well as the native library `bin\PdfViewerAPI.dll` for building your java project.

Note that in most Integrated Development Environments (IDEs) it is possible to link the provided javadoc in `doc\javadoc\*` to the java archive file `jar\PdfViewerAWT.jar`. Alternatively one can also view the javadoc directly by opening `doc\javadoc\index.html` in a browser.

Be aware, that the both java as well as the PDF Viewer API R2 are platform specific (32 or 64 bit). This means that the 64 bit version of the PDF Viewer API R2 will only work with 64 bit java installations.

2.3 Uninstall, Install a New Version

If you used the MSI for the installation, go to Start → 3-Heights™ PDF Viewer API R2... → Uninstall ...

If you used the ZIP file: In order to uninstall the product undo all the steps done during installation, e.g. un-register using `regsvr32 -u`, delete all files, etc.

Installing a new version does not require to previously uninstall the old version. The files of the old version can directly be overwritten with the new version. If using the COM interface, the new DLL must be registered, un-registering the old version is not required.

2.4 Color Profiles

For calibrated color spaces (such color spaces with an associated ICC color profile) the color conversion is well defined. For the conversion of uncalibrated device color spaces (DeviceGray, DeviceRGB, DeviceCMYK) however, the 3-Heights™ PDF Viewer API R2 requires apropriate color profiles. Therefore it is important, that the profiles are available and that they describe the colors of the device your input documents are intended for.

If no color profiles are available, default profiles for both RGB and CMYK are generated on the fly by the 3-Heights™ PDF Viewer API R2.

2.4.1 Default Color Profiles

If no particular color profiles are set default profiles are used. For device RGB colors a color profile named `"sRGB Color Space Profile.icm"` and for device CMYK a profile named `"USWebCoatedSWOP.icc"` are searched for in the following directories:

Windows

1. `%SystemRoot%\spool\drivers\color`
2. directory `Icc`, which must be a direct sub-directory of where the `PDFViewerAPI .dll` resides.

2.4.2 Get Other Color Profiles

Most systems have pre-installed color profiles available, for example on Windows at `%SystemRoot%\system32\spool\drivers\color\`. Color profiles can also be downloaded from the links provided in the directory `bin\Icc\` or from the following websites:

- http://www.pdf-tools.com/public/downloads/resources/colorprofiles.zip
- http://www.color.org/srgbprofiles.html
- https://www.adobe.com/support/downloads/iccprofiles/iccprofiles_win.html

2.5 Fonts

2.5.1 Font Cache

A cache of all fonts in all Font Directories is created. If fonts are added or removed from the font directories, the cache is updated automatically.

In order to achieve optimal performance, make sure that the cache directory is writable for the 3-Heights™ PDF Viewer API R2. Otherwise the font cache cannot be updated and the font directories have to be scanned on each program startup.

The font cache is created in the subdirectory `<CacheDirectory>/Installed Fonts` of the Cache Directory.

2.6 Special Directories

2.6.1 Directory for temporary files

This directory for temporary files is used for data specific to one instance of a program. The data is not shared between different invocations and deleted after termination of the program.

The directory is determined as follows. The product checks for the existence of environment variables in the following order and uses the first path found:

Windows

1. The path specified by the `%TMP%` environment variable.
2. The path specified by the `%TEMP%` environment variable.
3. The path specified by the `%USERPROFILE%` environment variable.
4. The Windows directory.

2.6.2 Cache Directory

The cache directory is used for data that is persisted and shared between different invocations of a program. The actual caches are created in subdirectories. The content of this directory can safely be deleted to clean all caches.

This directory should be writable by the application, otherwise caches cannot be created or updated and performance will degrade significantly.

Windows

- If the user has a profile:
  `%LOCAL_APPDATA%\PDF Tools AG\Caches`
- If the user has no profile:
  `<TempDirectory>\PDF Tools AG\Caches`

where `<TempDirectory>` refers to the Directory for temporary files.

2.6.3 Font Directories

The location of the font directories depends on the operating system. Font directories are traversed recursively in the order as specified below.

If two fonts with the same name are found, the latter one takes precedence, i.e. user fonts will always take precedence over system fonts.

Windows

1. `%SystemRoot%\Fonts`
2. directory `Fonts`, which must be a direct sub-directory of where `PDFViewerAPI .dll` resides.

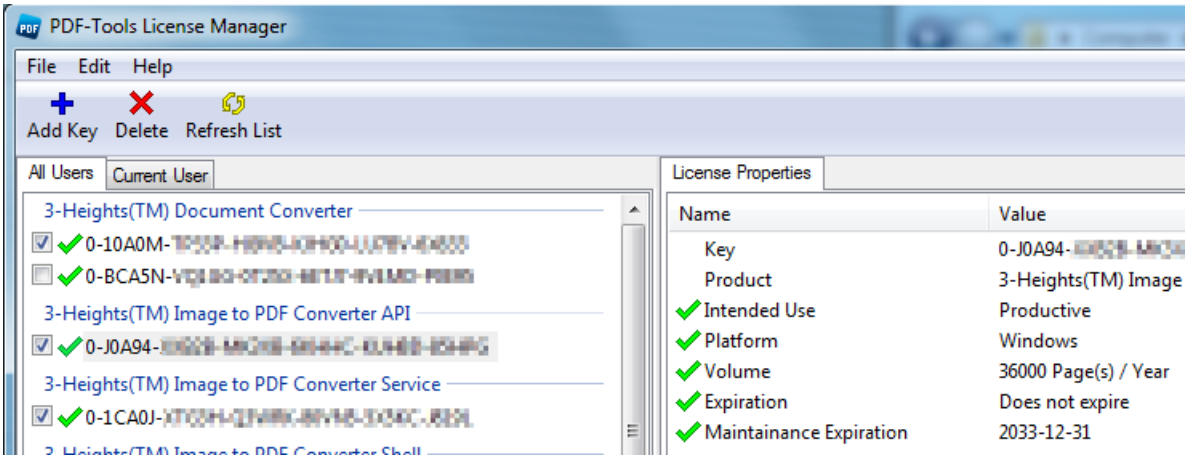
3 License Management

There are three possibilities to pass the license key to the application:

1. The license key is installed using the GUI tool (graphical user interface). This is the easiest way if the licenses are managed manually. It is only available on Windows.
2. The license key is installed using the shell tool. This is the preferred solution for all non-Windows systems and for automated license management.
3. The license key is passed to the application at run-time via the `SetLicenseKey` method. This is the preferred solution for OEM scenarios.

3.1 Graphical License Manager Tool

The GUI tool `LicenseManager.exe` is located in the `bin` directory of the product kit.



3.1.1 List all installed license keys

The license manager always shows a list of all installed license keys in the left pane of the window. This includes licenses of other PDF Tools products. The user can choose between:

- Licenses available for all users. Administrator rights are needed for modifications.
- Licenses available for the current user only.

3.1.2 Add and delete license keys

License keys can be added or deleted with the “Add Key” and “Delete” buttons in the toolbar.

- The “Add key” button installs the license key into the currently selected list.
- The “Delete” button deletes the currently selected license keys.

3.1.3 Display the properties of a license

If a license is selected in the license list, its properties are displayed in the right pane of the window.

3.1.4 Select between different license keys for a single product

More than one license key can be installed for a specific product. The check-box on the left side in the license list marks the currently active license key.

3.2 Command Line License Manager Tool

The command line license manager tool `licmgr` is available in the `bin` directory for all platforms except Windows.

A complete description of all commands and options can be obtained by running the program without parameters:

```
licmgr
```

List all installed license keys:

```
licmgr list
```

The currently active license for a specific product is marked with a star '*' on the left side.

Add and delete license keys:

Install new license key:

```
licmgr store X-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX
```

Delete old license key:

```
licmgr delete X-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX
```

Both commands have the optional argument `-s` that defines the scope of the action:

g For all users

u Current user

Select between different license keys for a single product:

```
licmgr select X-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX
```

3.3 License Key Storage

Depending on the platform the license management system uses different stores for the license keys.

3.3.1 Windows

The license keys are stored in the registry:

- "HKLM\Software\PDF Tools AG" (for all users)
- "HKCU\Software\PDF Tools AG" (for the current user)

4 Programming Interfaces

4.1 .NET

After installing and registering the PDF Viewer API R2, you find .NET sample solutions in the path `samples\CS.NET\`. There are samples provided for .NET WPF as well as .NET Windows Forms. For each interface there is both a minimal example, showing how to interface with the API to achieve basic functionality (i.e. opening a file and showing it) as well as an advanced example, with a fully integrated UI, that can be used like a normal desktop viewer. For each sample, a solution file is provided.

For starting out with the integration of the PDF Viewer API R2, it is recommended to use the minimal sample as a reference to achieve basic functionality. For integrating advanced features later, one may consult the advanced sample, which shows at least one way of integrating each feature.

To run the provided samples, ensure that you have registered your license using the License Manager (see Chapter 3 License Management) and that the platform of the kit you downloaded matches the platform you have configured the IDE to use (64 bit vs. 32 bit).

When implementing the API in your own application, consider that the API uses .NET framework 4.0. This means that the integrating application has to use .NET framework 4.0 or newer.

4.2 Java

After installing and registering the PDF Viewer API R2, you find java samples in the path `samples\java\`. The provided samples are written in swing and AWT. There is a minimal sample, showing how to interface with the API to achieve basic functionality (i.e. opening a file and showing it). The second sample uses all the advanced features that the API offers and has a fully integrated UI, that can be used like a normal desktop viewer.

For each sample a `run.bat` script is provided, which compiles and runs the sample using the java installation as configured by the PATH variable.

For starting out with the integration of the PDF Viewer API R2, it is recommended to use the minimal sample as a reference to achieve basic functionality. For integrating advanced features later, one may consult the advanced sample, which shows at least one way of integrating each feature.

5 User's Guide

Most samples in this guide are written in C#. The call sequence, however, for any other programming language is the same.

5.1 Open a PDF Document

Documents can be opened either from file using the method Open or from memory using the method OpenMem.

Example: Open PDF from File System.

```
private void Open(){
  viewer.onOpenCompleted += OnOpenCompletedEventHandler;
  viewer.Open(Directory.GetCurrentDirectory() + "\input.pdf", "");
}
private void OnOpenCompletedEventHandler(PdfViewerException ex)
{
  if(ex != null)
    MessageBox.Show("Failed to open file: " + ex.getMessage());
  viewer.onOpenCompleted -= OnOpenCompletedEventHandler;
}
```

Viewer is an object of type PdfViewerWPF or PdfViewerWinForms. The sample checks whether opening the input file was successful or not. If not, it shows a message box.

Note that it is required to register an eventHandler, as the opening operation is asynchronous and the **viewer.Open** command immediatly returns. In general it is recommended that you attach the eventHandler once at the beginning and don't detach when it gets triggered. However, if multiple sources use the open command and require their own event handlers, then attaching and detaching as shown in the example might be useful.

OpenMem is usually used when a PDF document is already available in memory, e.g. is read from a data base or is passed in-memory from another application.

The following example shows how a PDF document can be opened form memory by reading it from file and writing it in a byte array and then reading that byte array using OpenMem.

Example: Open PDF from Memory.

```
private void OpenMem_Click()
{
  viewer.onOpenCompleted += OnOpenCompletedEventHandler;
  byte[] fileMem = File.ReadAllBytes
    (Directory.GetCurrentDirectory() + "\input.pdf");
  viewerForm.viewerAPI.OpenMem(fileMem, "");
}
private void OnOpenCompletedEventHandler(PdfViewerException ex)
{
  if(ex != null)
    MessageBox.Show("Failed to open file: " + ex.getMessage());
  viewer.onOpenCompleted -= OnOpenCompletedEventHandler;
```

```
}
```

Using OpenMem is especially useful if the file is already in memory. This can either happen when the application just created the file or if the same file is used multiple times.

5.2 Navigation

There are various ways how the user can navigate in a document with the PDF Viewer API R2. The following features can be used to navigate:

- Use the property PageNo to set page number to be displayed. The page range goes from 1 to PageCount.
- Use the mouse in cursor mode eMouseMoveMode (default) to scroll the page by pressing the left mouse button and moving the mouse up and down. Change the cursor mode using the property MouseMode.
- Use horizontal and vertical scroll bars.
- Use the mouse wheel to scroll vertically or hold the control key to zoom in and out with the mouse wheel.
- Use the property Destination to move to a specific location and zoom level within the document.
- Use the outlines (bookmarks) or thumbnails from the navigation panel at left hand side of the control to access any page or section.
  - The outlines and thumbnails panels can be enabled or disabled using the properties ShowOutlines and ShowThumbnails.
- The Search function can be used to move the viewport to the found match.

5.3 Suspend Layouting

The methods SuspendLayout allows the layouting to be suspended. This can be useful if the implementing application wants to change multiple visual settings, but wishes to avoid having multiple visual updates. The layouting can then later be resumed using ResumeLayout.

The following example shows an advanced application of this principle. It suspends layouting before opening a file and then waits for the file having completed opening, using the OpenCompleted event. It then applies some custom navigation commands and only then resumes layouting. For the user this results in one single visual update, whereas without suspending, he might catch a few glimpses of the intermediate states in between the visual updates.

Example: Suspend layouting to prevent multiple visual updates.

```
private void Open(String fileName)
{
  viewer.SuspendLayout();
  viewer.OpenCompleted += OnOpenCompletedHandler;
  viewer.Open(fileName, "");
}

private void OnOpenCompletedHandler(PdfViewerException exception)
{
  viewer.OpenCompleted -= OnOpenCompletedHandler;
  if(exception == null)
  {
    viewer.PageLayoutMode = TPageLayoutMode.SinglePage;
    viewer.PageNo = 1;
    viewer.Rotate = 90;
```

```
  }
  viewer.ResumeLayout();
}
```

5.4 Using viewer in background

The PDF Viewer API R2 does not require to have a visual parent to be used. This means that one can instantiate the control in the background, open a file and only display it to the user once it is loaded. This is illustrated in the following example, assuming that the **this** object is a WPF control.

Example: Add viewer only to visual control after it has successfully opened the file.

```
private void Open(String fileName)
{
  viewer.OpenCompleted += OnOpenCompletedHandler;
  viewer.Open(fileName, "");
}

private void OnOpenCompletedHandler(PdfViewerException exception)
{
  viewer.OpenCompleted -= OnOpenCompletedHandler;
  if(exception == null)
  {
    this.Children.Add(viewer);
  }
  viewer.ResumeLayout();
}
```

5.5 Listening to Property changes

The API offers a multitude of properties that describe the current state of the viewer (see section Properties). However the application might need to not only know the current state, but also be informed when that state changes. For this purpose, the PDF Viewer API R2 implements the INotifyPropertyChanged Interface and the corresponding `PropertyChanged` event. The `PropertyChanged` event passes the string of the property that just changed along. This is illustrated in the following sample, which will listen for changes of the `FitMode` property and highlight toolstripbuttons as checked accordingly.

Example: Listen to changes of active fitmode.

```
viewer.PropertyChanged += OnFitModeChanged;

private void OnFitModeChanged(object sender, PropertyChangedEventArgs e)
{
  if (!e.PropertyName.Equals("FitMode"))
    return;
  fitPage.Checked = (viewerAPI.FitMode ==
    PdfTools.PdfViewerCSharpAPI.Model.FitMode.FitPage);
  fitWidth.Checked = (viewerAPI.FitMode ==
    PdfTools.PdfViewerCSharpAPI.Model.FitMode.FitWidth);
```

```
  fitActualSize.Checked = (viewerAPI.FitMode ==
    PdfTools.PdfViewerCSharpAPI.Model.FitMode.FitTrueSize);
}
```

This same pattern can be used with almost all readable properties, such as `Zoom`, `PageNo` or `MouseMode`.

When using the WPF API, then most properties are also available as dependency properties, which can be bound to directly in the xaml markup.

6 Programmer's Reference

In the following sections, we document the interface of the PdfViewerAPI, when using C#. Both a WPF and a Windows Forms Interface are available. While they are mostly identical, there are a few key differences, imposed by the used graphical library. For example, WPF allows the use of Dependency Properties, so all public properties in the WPF interface can be used as bindings and thus notify changes of the property value automatically. In Windows Forms, the viewer client has to implement the INotifyPropertyChanged Interface and must then manually handle the PropertyChanged Event to update these values.

The java interface is not described in detail, however the java interface is very similiar to the here described interfaces. Java language limitations have required some changes though:

- Each C# property corresponds to a pair of get and set methods in java. (Example: C# Property `Zoom` corresponds to java methods **getZoomFactor** and **setZoomFactor**)
- Each C# event corresponds to a java Interface, with a single method. Classes implementing this interface are called **EventListeners** and you can then register the listener on the API (Example: C# event `SearchCompleted` corresponds to java interface IOnSearchCompletedListener with method onSearchCompleted and can be registered at PdfViewerComponent.registerOnSearchCompleted)

6.1 Enumerations

The following enumerations are defined in
`PdfTools.PdfViewerCSharpAPI.Model.IPdfViewerController`

> **Java API Note:** In Java these definitions are found in
> `com.pdf_tools.pdfviewer.Model.IPdfViewerController`

6.1.1 TPageLayoutMode

TPageLayoutMode Table

TPageLayoutMode	
`SinglePage`	Shows one single page, scrolling when reaching the top/bottom of the page will jump to the next/previous page.
`OneColumn`	Shows one column of pages, that can be scrolled through.
`TwoColumnLeft`	Shows a column of pairs of pages, that can be scrolled through. The first page is placed left.
`TwoColumnRight`	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.

TPageLayoutMode Table

`TwoPageLeft`	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.
`TwoPageLeft`	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.

6.1.2 FitMode

FitMode Table

FitMode	
`FitPage`	The window is zoomed to fit the whole page into the viewport.
`FitWidth`	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.
`FitTrueSize`	The window is zoomed to reflect the true size of the page.

6.1.3 TDestination

TDestination

TDestination	
`eDestinationInvalid`	Placeholder for invalid Destinations.
`eDestinationFit`	Fits the viewport to the destination page
`eDestinationFitH`	Fits the viewport to the horizontal dimension of the destination page
`eDestinationFitV`	Fits the viewport to the vertical dimension of the destination page
`eDestinationFitR`	Fits the viewport to the destination rectangle
`eDestinationFitB`	Fits the viewport to the bounding box of the destination page

TDestination

eDestinationFitBH	Fits the viewport to the horizontal dimension of the destination page bounding box
eDestinationFitBV	Fits the viewport to the vertical dimension of the destination page bounding box
eDestinationFitXYZ	Fits the viewport to be at a specific location with a specific zoom level

6.1.4 TMouseMode

TMouseMode

TMouseMode	
eMouseUndefMode	Placeholder for undefined mousmode.
eMouseMoveMode	Use the mouse to move the viewport on the document.
eMouseZoomMode	Use the mouse to draw a rectangle. The viewport will then zoom in to fit said rectangle.
eMouseTextExtractMode	Use the mouse to draw a rectangle. All contained text will be extracted and returned as the event TextExtracted.
eMouseHighlightMode	Use the mouse to draw a rectangle. All textfragments contained within said rectangle will be highlighted.

6.1.5 TViewerTab

TViewerTab

TViewerTab	
eOutlineTab	Select the outline tab (bookmarks tab).
eThumbnailTab	Select the thumbnails tab.

6.2 PDF Viewer Application Program Interface

In this section there is a detailed interface description of the .NET WPF Control **PdfTools.PdfViewerWPF.PdfViewerWPF**. The interface of the .NET Windows Forms Control **PdfTools.PdfViewerWinForms.PdfViewerWinForms**

is very similar, with a few key differences imposed by Windows Forms. For the java swing Viewer Component **com.pdf_tools.pdfviewer.SwingAPI.PdfViewerComponent** the interface is somewhat different. Key differences are indicated in the following sections.

6.2.1 Properties

Note that this is the documentation of the .NET interface. In java properties are implemented as simple set and get methods. E.g. the equivalent of property **Rotate** in java are the methods **setRotation** and **getRotation**. For exact method names, consult the provided JavaDoc.

For the WPF interface most of these properties are also available as dependency properties with the same name.

PageCount

Property (get): `int PageCount`

The number of pages in the opened document. 0 if no Document open.

Destination

Property (get, set): `PdfTools.PdfViewerCSharpAPI.Utilities.Destination Destination`

Get or set the currently displayed destination.

SelectedViewerTab

Property (get, set): `PdfTools.PdfViewerCSharpAPI.Model.TViewerTab SelectedViewerTab`

Get or set the currently selected tab on the sidepanel (Outlines or thumbnails). See TViewerTab.

Rotate

Property (get, set): `int Rotate`
Default: `0`

The “rotate” angle in multiples of 90 degrees. Each individual page is rotated by the given angle.

Border

Property (get, set): `double Border`
Default: `6.0`

The border displayed in between pages in user units.

Resolution

Property (get, set): `PdfTools.PdfViewerCSharpAPI.Model.Resolution Resolution`
Default: `(96,96)`

The display resolution in DPI.

FitMode

Property (get, set): `PdfTools.PdfViewerCSharpAPI.Model.FitMode FitMode`
Default: `FitDocument`

The FitMode used for displaying the document. See also `FitMode`.

PageLayoutMode

Property (get, set): `PdfTools.PdfViewerCSharpAPI.Model.TPageLayoutMode PageLayoutMode`
Default: `PageLayoutDocument`

The PageDisplayMode used for displaying the document. See also `TPageLayoutMode`.

IgnoreEmbeddedPreferences

Property (get, set): `bool IgnoreEmbeddedPreferences`
Default: `false`

When this property is set to true before opening a file, the viewer will ignore all viewing preferences that are embedded in said file. This includes open destination and the preferred layout mode.

PageNo

Property (get, set): `int PageNo`

The topmost pagenumber on the viewport.

PageOrder

Property (get, set): `List<int> PageOrder`
Default: `List<int>(Enumerable.Range(1, document.PageCount))`

Changing the viewing order of the displayed pages. The index of the list corresponds to the viewer page (viewer page 1 is at index 0, viewer page 2 at index 1 etc.) and the entry at that index corresponds to which PDF page will be displayed.

Zoom

Property (get, set): `double Zoom`
Default: `1.0`

The zoomFactor that the document is displayed with.

ShowOutlines

Property (get, set): `bool ShowOutlines`
Default: `true`

Determines if Outlines should be shown.

ShowThumbnails

Property (get, set): `bool ShowThumbnails`
Default: `true`

Determines if Thumbnails should be shown.

FileName

Property (get): `string FileName`
Default: `"-"`

The currently opened file name. File name is "-" if nothing is opened.

SearchOverlayBrush

Property (get, set): `System.Windows.Media.Brush SearchOverlayBrush`
Default: `Colors.LightBlue with Opacity=0.5`

Is the brush to use for the rectangular overlays of search results.

> **Windows Forms Note:** This property is implemented as a **System.Drawing.Color** with name **SearchOverlayColor**

MouseMode

Property (get, set): `TMouseMode MouseMode`
Default: `eMouseModeMove`

Sets the currently active mouse mode. See TMouseMode.

SearchMatchCase

Property (get, set): `bool SearchMatchCase`
Default: `true`

Configure, whether the search algorithm only finds matches with matching character case.

SearchWrap

Property (get, set): `bool SearchWrap`
Default: `true`

Configure, whether the search algorithm should wrap around when reaching the end of the document (or the start, if searching backwards).

SearchPrevious

Property (get, set): `bool SearchPrevious`
Default: `false`

Configure, whether the search algorithm searches for the next match (false, direction towards the end of the document) or the previous one (true, direction towards the frond of the document).

SearchRegex

Property (get, set): `bool SearchRegex`
Default: `true`

Configure, whether the search algorithm should interpret the given string as a regular expression (true), or a plain string (false).

ProductVersion

Property (get): `String ProductVersion`

Get the version of the 3-Heights™ PDF Viewer API R2 in the format "A.C.D.E".

ViewerPaneRectangle

Property (get): `System.Windows.Rect ViewerPaneRectangle`

The rectangle, where the viewer pane is situated within the WPF viewer Control in screen coordinates.

> **Windows Forms Note:** This property is implemented as a **System.Drawing.Rectangle**

6.2.2 Methods

PdfViewerWPF

Method: `PdfViewerWPF()`

Constructor of Interfacing class PdfViewerWPF, instantiates an empty viewer, which can now be configured and used to open files.

> **Windows Forms Note:** This method is implemented with name **PdfViewerWinForms**

Dispose

Method: `Dispose()`

Disposes Viewer and all its children. Must be called to terminate all running threads.

SetLicenseKey

Method: `SetLicenseKey(string licenseKey)`
Static

Set the key of your license.

GetLicenseIsValid

Method: `bool GetLicenseIsValid()`
Static

Returns whether the currently set key is valid.

InvokeCallbackOnDispatcher

Method: `InvokeCallbackOnDispatcher()`

Invoke the given callback on the Thread which owns this control. Returns after execution of callback terminates. All calls on the API should be performed on this thread.

BeginInvokeCallbackOnDispatcer

Method: `BeginInvokeCallbackOnDispatcer()`

Invoke the given callback on the Thread which owns this control. Returns immediately (not necessarily after termination of callback). All calls on the API should be performed on this thread.

Open

Method: `bool Open(string filename, string password)`

Instructs the viewer to open a file. Blocks until open operation has completed.

OpenMem

Method: `bool OpenMem(byte[] memBlock, string password)`

Instructs the viewer to open a file. Blocks until open operation has completed.

Search

Method: `Search(string searchText)`

Searches text of PDF for the inserted text. Search engine can be configured using Properties SearchWrap, SearchPrevious, SearchMatchCase and SearchRegex. Search results can be aquired by listening to the event SearchCompleted.

OnApplyTemplate

Method: `OnApplyTemplate()`

Initializes graphic components of the viewer. If PdfViewerWPF_R2 is used as a standard wpf component, this is called when tempaltes are applied. Otherwise manual invocation is necessary

> **Windows Forms Note:** This method is implemented seperately with name **Initialize**

SendInitialPropertyChanges

Method: `SendInitialPropertyChanges()`

Triggers DependencyPropertyChangedEvents of dependecyProperties LayoutMode, FitMode and PageDisplayMode, so Listeners to these events can read out initial values.

NextPage

Method: `NextPage()`

Set viewport to the next page.

PreviousPage

Method: `PreviousPage()`

Set viewport to the previous page.

SuspendLayout

Method: `SuspendLayout()`

Suspends Layouting the document until ResumeLayout is called.

ResumeLayout

Method: `ResumeLayout()`

Excplicitely forces layouting the document once and ends suspension caused by SuspendLayout.

6.2.3 Events

Java Swing API note: Events do not exist in older versions of Java. Thus there is an Interface defined for each event, containing a single method. One can implement said interface and write the code of the event handler in the given method. Then one has to register the implemented class on the viewer controller using the registerOn**PropertyName** name with the corresponding property name. E.g. the **OpenCompleted** event can be used by creating a class implementing **IOnOpenCompletedListener** and inserting the event handling code in the method **onOpenCompleted**. After registering an instance of the implemented class of the API using the method **registerOnOpenCompleted**, the **onOpenCompleted** method is then always called when a open operation terminates.
Also the **INotifyPropertyChanged** pattern and the associated **PropertyChanged** event does not exist in java. Instead there exists a separate event (i.e. the structure described above with an interface and a register method) for each property, notifying all registered listeners of the change in the properties value.

SearchCompleted

Event: `Action<PdfTools.PdfViewerCSharpAPI.Model.PdfSearcher.SearchResult> SearchCompleted`

Event triggers when method Search has been called and the search terminated. The event deliveres the result of the performed search as an Object of Type SearchResult, containing information about the found match:

- Pagenumber on which the match was found (if the match spans multiple pages, the first page will be returned)
- List of rectangles, that describe the location of the found match. (Long search terms yield multiple rectangles on multiple pages)
- Index describes the index of the first letter of the match within its page, to identify this match

TextExtracted

Event: `Action<String> TextExtracted`

Event triggers when text has been selected for extraction using the TMouseMode.eMouseTextExtractMode returning the selected text as one concatenated string.

OpenCompleted

Event: `Action<PdfTools.PdfviewerCSharpAPI.Utilities.PdfViewerException OpenCompleted`

Event triggers when the opening of a file (e.g. by using Open) has completed. If there have been no errors while opening, the exception parameter will be null, otherwise it will carry the causing exception.

PropertyChanged

Event: `System.ComponentModel.PropertyChangedEventHandler PropertyChanged`

Event triggers when any of the following properties changes its value: PageLayoutMode, FitMode, MouseMode, PageNo, PageCount, Zoom or Rotate. The event passes the string of the name of the property along.

7 Tipps, Tricks and Troubleshooting

7.1 Troubleshooting: TypeInitializationException

The most common issue when using the .NET interface is that the correct native DLL `PDFViewerAPI .dll` is not found at execution time. This normally manifests when the constructor is called for the first time and an exception of type `System.TypeInitializationException` is thrown.

This exception can have two possible causes, distinguishable by the inner exception (property `InnerException`):

`System.DllNotFoundException` Unable to load DLL `PDFViewerAPI .dll`: The specified module could not be found.

`System.BadImageFormatException` An attempt was made to load a program with an incorrect format.

The following sections describe in more detail, how to resolve the respective issue.

7.1.1 Troubleshooting: DllNotFoundException

This means, that the native DLL `PDFViewerAPI .dll` could not be found at execution time.

Resolve this by either:

- adding `PDFViewerAPI .dll` as an existing item to your project and set its property "Copy to output directory" to "Copy if newer", or
- adding the directory where `PDFViewerAPI .dll` resides to the environment variable `%Path%`, or
- copying `PDFViewerAPI .dll` to the output directory of your project.

7.1.2 Troubleshooting: BadImageFormatException

The exception means, that the native DLL `PDFViewerAPI .dll` has the wrong "bitness" (i.e. platform 32 vs. 64 bit). There are two versions of `PDFViewerAPI .dll` available: one is 32-bit (name of product kit download contains `WIN32`) and the other 64-bit (name of download contains `X64`). It is crucial, that the platform of the native DLL matches the platform of the application's process.

The platform of the application's process is defined by the project's platform configuration for which there are 3 possibilities:

AnyCPU This means, that the application will run as a 32-bit process on 32-bit Windows and as 64-bit process on 64-bit Windows. When using AnyCPU one has to use a different native DLL, depending on the platform of Windows. This can be ensured either when installing the application (by installing the matching native DLL) or at application start-up (by determining the application's platform and ensuring the matching native DLL is loaded).

x86 This means, that the application will always run as 32-bit process, regardless of the platform of the Windows installation. The 32-bit DLL runs on all systems, which makes this the simplest configuration. Hence, if an application needs to be portable and does not require any specific 64-bit features, it is recommended to use this setting.

x64 This means, that the application will always run as 64-bit process. As a consequence the application will not run on a 32-bit Windows system.