3-Heights™ PDF Viewer .NET WPF Control

Version 4.5

User Manual

Contact: pdfsupport@pdf-tools.com

Owner: **PDF Tools AG**

Kasernenstrasse 1
8184 Bachenbülach
Switzerland

http://www.pdf-tools.com

Table of Content

1 Introduction

1.1 Description

The 3-Heights™ PDF Viewer .NET WPF Control is a component which can be seamlessly integrated in .NET Windows Presentation Foundation (WPF) applications. The control offers many options for displaying PDF documents: files can be viewed in single page or multi-page mode, and the navigation supports links, bookmarks and calling up pages arbitrarily. The component supports PDF documents in many languages: the reproduction of Japanese, Chinese, Korean, Russian, etc., is no problem at all.

Viewing

- Display PDF file in single page or multi-page mode
- Enter password to decrypt PDF documents
- Open files from file system or from internet (HTTP, HTTPS, FTP)
- Query the number of pages in a document
- Scale the display (fit to page size, fit to page width, actual size, zoom)
- Set display resolution
- Reproduce documents with Chinese, Japanese and Korean fonts (CJK)
- Read document from file or memory
- Convert viewer coordinates to PDF coordinates
- Rotate the page
- Set the border size

Navigation

- Show and hide windows for bookmarks and page thumbnails
- Jump to a bookmark's location
- Jump to the location of a link within the document
- Move windows vertically and horizontally using the mouse or keyboard
- Enlarge and reduce windows with the aid of a zoom rectangle or zoom factor
- Freely select any page in the document for display
- Query the position of the cursor
- Text search
- Support for touch input: Multi touch zoom and scrolling inertia

Formats

Input Formats

- PDF 1.0 - 1.7, PDF/A

Compliance

Standards: ISO 32000-1 (PDF 1.7), ISO 19005-1 (PDF/A-1), ISO 19005-2 (PDF/A-2), ISO 19005-3 (PDF/A-3)

1.2 Interfaces

The API interface technology is suitable for all .NET languages such as C# and VB.NET.

1.3 Operating Systems

Windows XP, Vista, 7, 8 - 32 and 64 bit

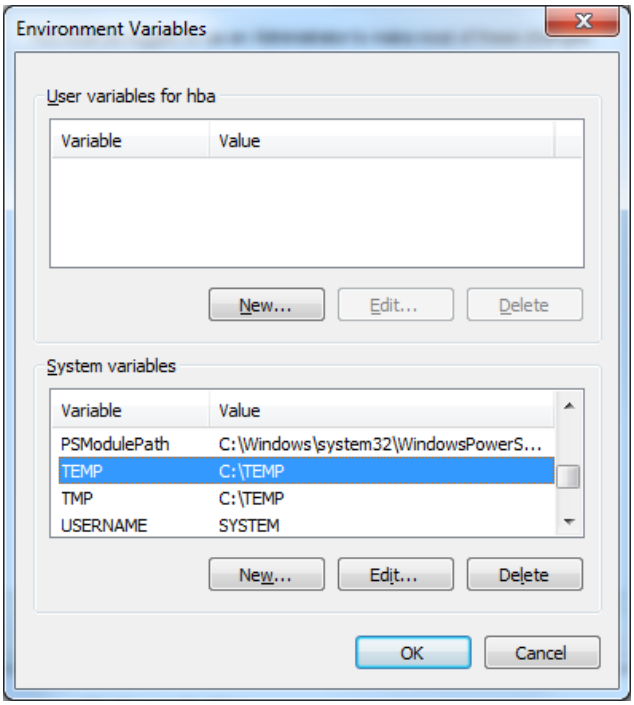
Windows Server 2003, 2008, 2008-R2, 2012 - 32 and 64 bit

2 Installation

The software distribution kit of the 3-Heights™ PDF Viewer .NET WPF Control comes as a ZIP[1] archive containing various files including runtime binary executable code, files required for the developer, documentation and license terms. There is a single ZIP archive available, suitable for both 32 bit platforms as well as 64 bit platforms.

1. Download the ZIP archive of the product for your specific platform from your download account at www.pdf-tools.com.
2. Open the ZIP archive.
3. Check the appropriate option to preserve file paths (folder names) and unzip the archive to a local folder (e.g. *C:\program files\pdf-tools*).
4. The unzip process now creates the following subdirectories:
   - *bin*: Contains the standard runtime executable binary code
   - *bin\x64*: Contains the runtime executable binary code, that is specific for 64 bit platform
   - *doc*: Contains documentation files
   - *Samples*: Contains sample programs to run the component
5. Ensure the two system environment variables TEMP and TMP exist and point to an existing directory. This directory is required to temporarily install fonts that are embedded in PDF documents.

   Control Panel -> System -> Advanced -> Environment Variables



Here is an overview of the files that come with the PDF Viewer:

[1] There is also the option to download the software as MSI file, which makes the installation easier.

bin\PdfViewerAPI.dll	The native DLL contains the main functionality for 32-bit platforms.
bin\x64\PdfViewerAPI.dll	The native DLL contains the main functionality for 64-bit platforms.
bin\PdfViewerNET.dll	The .NET interface DLL is a wrapper to the native DLL (required).
bin\PdfViewerWPF.dll	The .NET assembly of the PDF Viewer component (required).
bin\PdfViewerWPF.xml	The interface documentation for Visual Studio.
doc.pdf*	The user's manual and programming reference.
*Samples**	Sample programs to run the component.

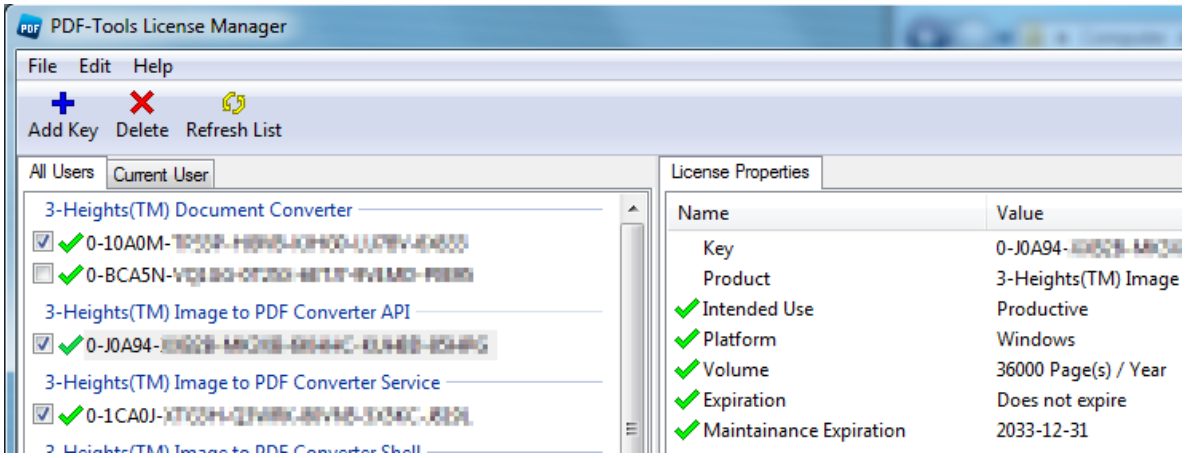
3 License Management

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 runtime via the “LicenseKey” property. 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.



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.

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.

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.

Select between different license keys for a single product

More than one license key can be installed for a specific product. The checkbox 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 ist 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

The license keys are stored in the registry:

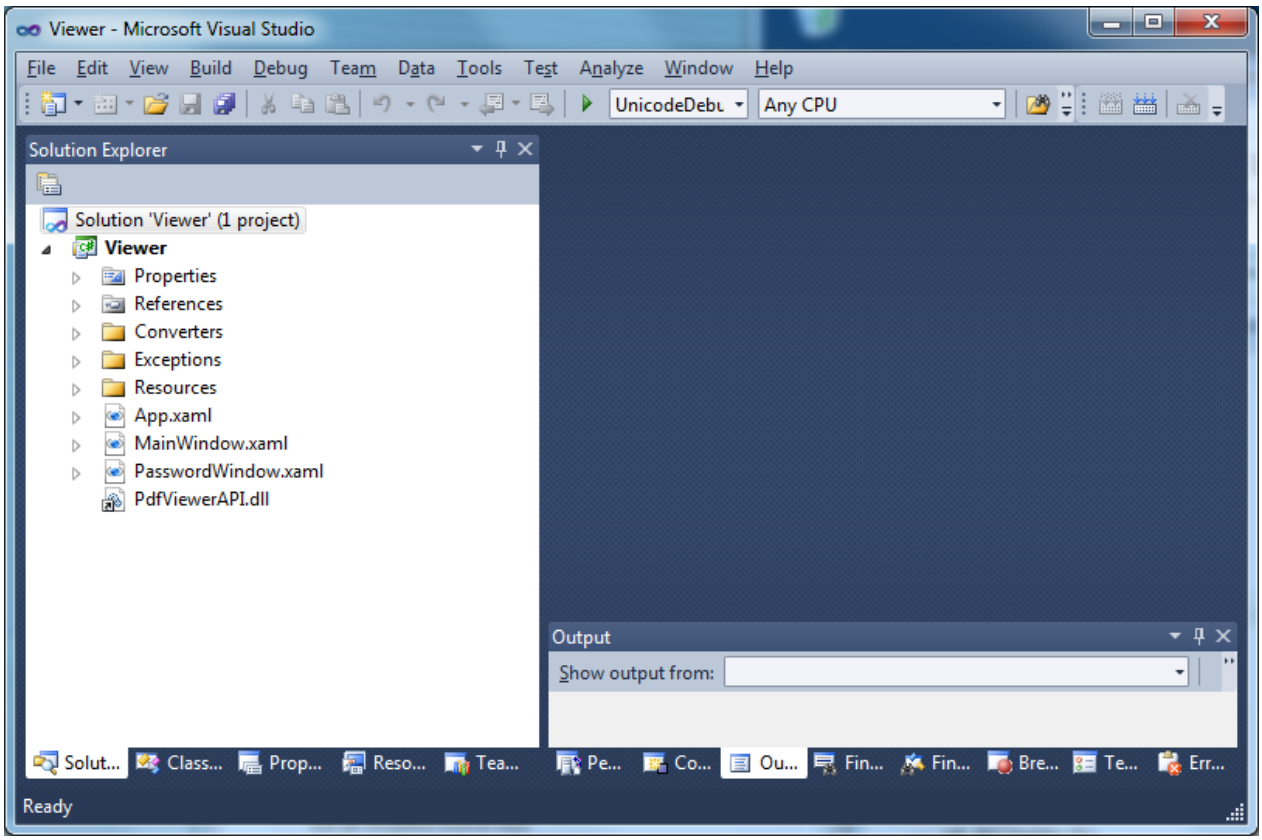
- HKLM\Software\PDF Tools AG (for all users)
- HKCU\Software\PDF Tools AG (for the current user)

4 User's Guide

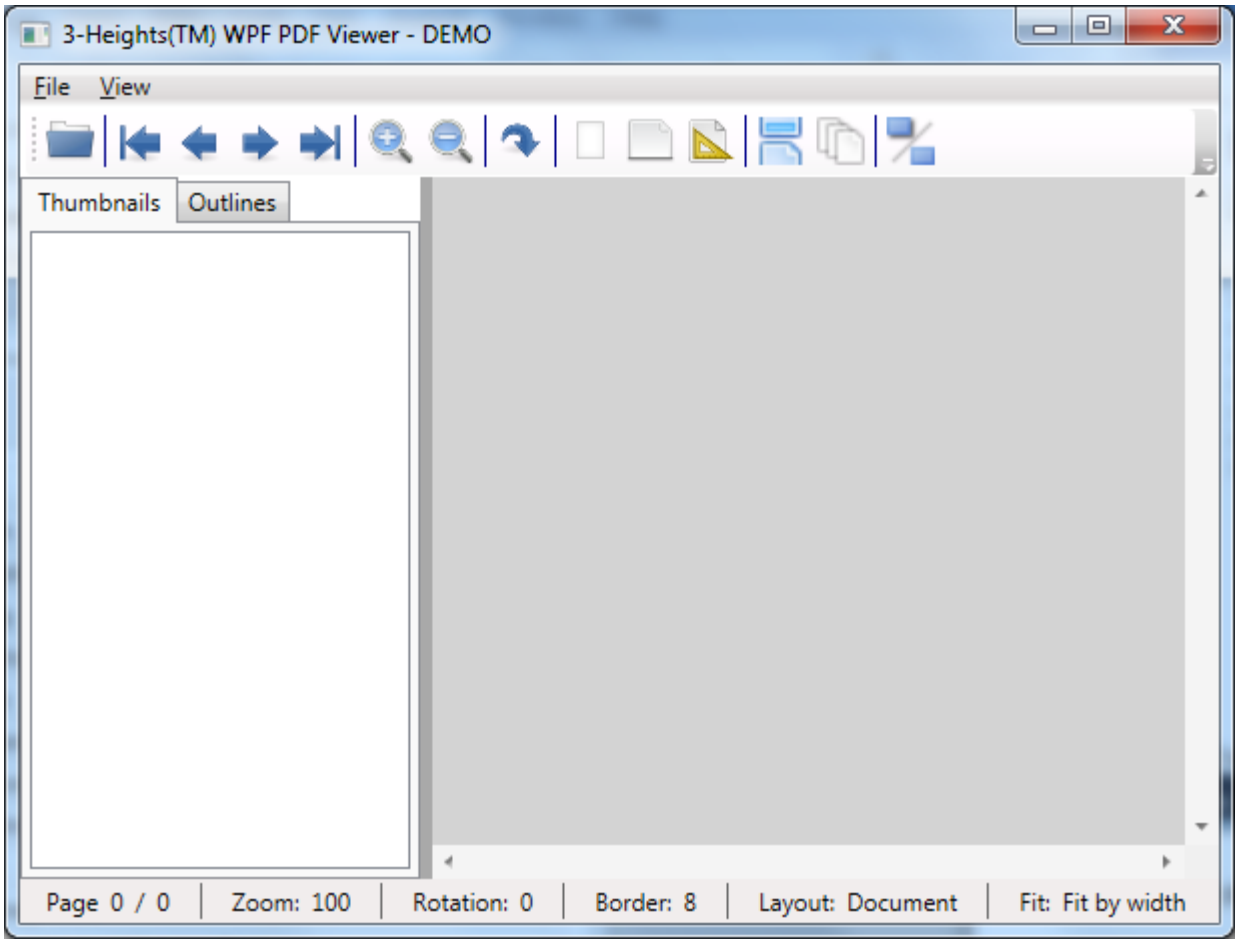
4.1 Getting started

After installing PDF Viewer, you can open the sample program in Visual Studio to run and test the component. Here are the steps:

1. Go to the sample directory, e.g. C:\Program Files\PDF Tools\VWWA\Samples\CS.NET\Viewer
2. Open the file *Viewer.sln*.
3. This should now look like this



Now press the "run" button and the viewer should pop up.



Now you can open a file using the left-most button in the toolbar and try out the component.

4.2 Open a file

Use the *Open* method to open a PDF file from disk. If the call fails then the license might be invalid, so check for a valid license in this case.

```
if (!Viewer.Open(dialog.FileName, null))
{
    if (!PdfViewerWPF.GetLicenseIsValid())
    {
        MessageBox.Show(Properties.MainWindowRes.invalid_license,
                        Properties.MainWindowRes.open_error,
                        MessageBoxButton.OK, MessageBoxImage.Error);
    }
    else
    {
      ///
    }
}
```

To open a PDF file for memory, use the method *OpenMem.*

4.3 Open a file using a password

A password must be provided if the file is encrypted with a user password. For documents that have no user password set, use the default password - an empty string.

```
if (!Viewer.Open(dialog.FileName, null))
{
    if (!PdfViewerWPF.GetLicenseIsValid())
    {
      ///
    }
    else
    {
        PasswordWindow window = new PasswordWindow();
        Viewer.Open(dialog.FileName, window.Password);
    }
}
```

4.4 Navigate

There are various ways how the user can navigate in a document with the viewer control. The application programmer has all options to enable, disable or limit the user's navigation. The following features can be used to navigate:

- Use the property *Page* to set page number to be displayed. The page range goes from 1 to *PageCount*.
- Use the mouse.
- Use horizontal and vertical scroll bars.
- Use the mouse wheel to scroll.
- 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 random access a position.
- Internal and external links within the content of the PDF document can also be used to navigate.

4.5 Use Custom Styles to Modify the Control

The PdfViewerWPF control can be customized using custom styles. Note that named controls (e.g. PART_MainViewer) may not be removed from the style file. They may be hidden though.

Example: create a custom style resource CustomStyle.xaml and add it to the control in MainWindow.xaml

```
<custom:PdfViewerWPF Name="PdfViewer" Grid.Row="1" FitMode="FitWidth"
LayoutMode="LayoutDocument" Style="{StaticResource CustomStyle}" >
```

5 Programmer's Reference

The following section lists all methods and properties of the COM interface of the 3-Heights™ PDF Viewer .NET WPF Control.

5.1 Methods

Close

```
bool Close()
```

Close the currently open document. The method blocks until the document is closed.

Return value:

True if the file was open.

False otherwise

Dispose

```
void Dispose()
```

Release unmanaged resources. After calling this method the object cannot be used anymore.

Open

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

Open a document from an input file. This method makes the objects in the document accessible. If there is an already open document it is closed first.

Parameters:

filename The input file name and optionally the file path, drive or server string according to the operating systems file name specification rules.

password The user or the owner password of the encrypted PDF document. If this parameter is null or an empty string the default (i.e. no password) is used. If the document is not a PDF document this parameter is ignored.

Return value:

True if the file could be opened.

False otherwise

OpenMem

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

Open a document from a memory block. This method makes the objects in the document accessible. If there is an already open document it is closed first.

Parameters:

memBlock Memory block containing the document stream as one dimensional byte array.

password The user or the owner password of the encrypted PDF document. If this parameter is null or an empty string the default (i.e. no password) is used. If the document is not a PDF document this parameter is ignored.

Return value:

True if the file could be opened.

False otherwise

Search

```
void Search(string searchText, SearchResultDelegate resultDelegate)
```

Search text in the document. The search starts at the current page. In the case of a successful search, the document is positioned to the next occurrence of the text and the found string is highlighted.

Parameters:

searchText The text to search

resultDelegate Callback for the result of the current search

SearchNext

```
void SearchNext(SearchResultDelegate resultDelegate)
```

Continue a previously started text search operation. The search starts at the current page. In the case of a successful search, the document is positioned to the next occurrence of the text and the found string is highlighted.

Parameters:

resultDelegate Callback for the result of the current search

SearchEnd

```
void SearchEnd()
```

Terminate a previously started text search operation. Highlights from previous text searches are removed.

ScrollToBottom

```
Method void ScrollToBottom()
```

Scrolls vertically to the bottom of the current view.

ScrollToLeftEnd

```
Method void ScrollToLeftEnd()
```

Scrolls to the horizontally to the beginning of the current view.

ScrollToRightEnd

```
Method void ScrollToRightEnd()
```

Scrolls to the horizontally to the end of the current view.

ScrollToTop

```
Method void ScrollToTop()
```

Scrolls vertically to the top of the current view.

SetLicenseKey

```
bool SetLicenseKey(string licenseKey)
```

Set the license key. The call to this method is required if the license key is passed programmatically. If the license key has been installed using the license manager then this call is obsolete.

Parameters:

licenseKey The license key string

Return value:

True if the license key is valid.

False otherwise

GetLicenseIsValid

```
bool GetLicenseIsValid()
```

Check whether the license key is valid.

Return value:

True if a valid license key was found.

False otherwise

5.2 Properties

Border

```
double Border
```

Get and set the border size in user units. Default: 0

Destination

```
Destination Destination
```

Get and set the actual destination in the document. Setting the destination navigates to the corresponding position in the document. If the zoom is 0, the current zoom is used. The page denotes the first page in the viewport. The x and y positions denote the top left corner in the viewport.

FitMode

```
FitMode FitMode
```

Get and set the fit mode. The fit mode defines whether the whole page or the page width fits into the view port or the page is shown in its true size. Default: FitWidth

LayoutMode

```
LayoutMode LayoutMode
```

Get and set the layout mode. The layout mode defines whether the view port shows a part of a single page or of the whole document. Default: LayoutDocument

PageCount

```
int PageCount
```

Get the number of pages in the document. Zero if the document is not open.

PageNo

```
int PageNo
```

Get and set the page number. Setting the page number navigates to the given page (see Destination).

Resolution

```
Resolution Resolution
```

Get and set the display resolution in dpi. Initially the resolution is set to the resolution of the display device.

Rotate

```
int Rotate
```

Get and set the 'rotate' angle in multiples of 90 degrees. Each individual page is rotated by the given angle. Default: 0

SearchOverlayBrush

```
Property Brush SearchOverlayBrush
Accessors: Get , Set
Default: lightblue SolidColorBrush with opacity of 0.5
```

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

ShowOutlines

```
bool ShowOutlines
```

Get and set the show status of the document outlines. If set to true the outline pane is visible.
Default: true

ShowThumbnails

```
bool ShowThumbnails
```

Get and set the show status of the page thumbnails. If set to true the thumnail pane is visible.

Default: true

UserUnit

```
double UserUnit
```

Get the documents user unit measure in points (1 pt = 1/72 inch). Default: 1.0

Zoom

```
double Zoom
```

Get and set the zoom factor. A zoom factor of 1.0 denotes true size. The zoom factor depends on the fit mode and the size of the viewport. Default: depends on fit mode.

5.3 Delegates

SearchResultDelegate

```
void SearchResultDelegate(string searchText, bool success)
```

If “Search” and “SearchNext” is used then this delegate is called back if the search operation terminates. The string parameter returns the text which was searched for and a bool which returns true if the text was found.

6 Tips, Tricks and Troubleshooting

6.1 Performance

The 3-Heights™ PDF Viewer .NET WPF Control provides a variety of settings to tune the performance. In most cases a simple rules applies: Higher quality takes more resources (memory, CPU) and therefore lowers the performance.

The following settings have an impact on the performance:

- Content of the PDF: A document with thousands of objects requires more time for rendering than a page with plain text.
- Resolution: The higher the size of the viewer window the more pixels need to be drawn and the lower the performance.
- Thumbnails: Displaying thumbnails reduces the performance. Use outlines instead.

6.2 Fonts and Text

Font Replacement Strategy

This section describes the exact behavior of font handling of the rendering engine. It is rather technical and it is not required to be understood in order to properly use the software.

The following steps are performed sequentially in the search of a font. If a font is found, the search is stopped; otherwise the next step is performed.

1. If the font is not embedded:
    a. If the font name appears in the [Replace] section in the configuration file “fonts.ini” the name is replaced and looked up in the installed font collection
    b. if it is a standard font[2] it is replaced by the equivalent TrueType font name and it is looked up in the installed font collection
    c. If the font name appears in the [Fonts] section in the configuration file “fonts.ini” the name is replaced and looked up in the installed font collection
    d. If the font has “Italic” or “Bold” in its name the font without these styles is looked up in the installed font collection
2. If a font name is looked up in the installed font collection then the name compare is performed as follows:
    a. PostScript name
    b. TrueType name without blanks (a missing style is interpreted as “Regular” or “Normal”)
    c. TrueType name without modifications
3. If the font is embedded, it is converted to a Windows compatible font and temporarily installed
4. If the font is not embedded and the Unicodes are available then the nearest font from the installed font collection is tailored to the metrics of the font.

[2] e. g. Times−Roman, Helvetica, Courier

5. If the font is embedded then it is converted to outlines.
6. In all other cases the nearest font from the installed font collection is used

Using the Font Mapping File *fonts.ini*

"fonts.ini" is a configuration file to map fonts used in the PDF to fonts pre-installed on the system. The mapping file must reside a directory named "Fonts", which must be a direct sub-directory of where the main DLL or executable resides.

The mapping file is optional. It consists of two sections: [fonts] and [replace].

Both sections are used to map fonts in the PDF to fonts in the installed font collection on the operating system. This comes into play when the font in the PDF document does not have an embedded font program, or the embedded font is not usable.

The mapping only works if the font types of the specified fonts are matching; e. g. if the font in the PDF is a symbolic font, such as "Symbol" or "ZapfdingBats", the mapped font must be symbolic too.

The section [fonts] is only considered if the font-matcher does not find an appropriate font amongst the existing installed fonts. It is suggested to only use this section.

The section [replace] is stronger and applied before the font-matcher. This means a font will be replaced as defined, even if the correct installed font is available on the system.

Syntax

The syntax of the mapping file is this:

```
[fonts]
PDF_font_1=installed_font_1,{font_style}
PDF_font_2=installed_font_2,{font_style}
[replace]
PDF_font_n=installed_font_n,{font_style}
```

PDF_font_* is the name of the font in the PDF. This name can be found in one of the following ways:

- Use any tool that can list fonts. Such as *3-Heights PDF Extract* or *3-Heights PDF Optimization*. Ignore possible prefixes of subset fonts. A subset prefix consists of 6 characters followed by the plus sign. For example "KHFOKE+MonotypeCorsiva", in this case only use "MonotypeCorsiva" as font name in the mapping file.
- Open the document with Adobe Acrobat, use the "MarkUp Text Tool"  , mark the text of which you would like to know the font name, right-click it, select "Properties…"

installed_font_* is the font family name of the installed font. To retrieve this name, find the font in the Windows' font directory and open it by double-clicking. The first line in the property window displays the font family name (this may vary depending on the operating system). The font family name does not include font styles; so an example of a font family name is "Arial", but not "Arial Italic".

font_style is an optional style, that is added coma-separated after the font family name. The style is always one word. Examples of font styles are "Italic", "Bold", BoldItalic".

Example

```
[fonts]
Ryumin-Light=MS Mincho
GothicBBB-Medium=MS Gothic
[replace]
```

```
ArialIta=Arial,BoldItalic
```

6.3 Unimplemented features

The 3-Heigths™ rendering engine supports transparency functions such as a number of blend modes as well as isolated and non-isolated transparency groups, but not transparency in general.

Certain types of tiling and shading patterns may not correctly be reproduced by the rendering engine.