User Manual



3-Heights™
Java Document Viewer

PDF/A compliant

Version 4.5date

PDF-TOOLS.COM
Premium PDF Technology

Contents

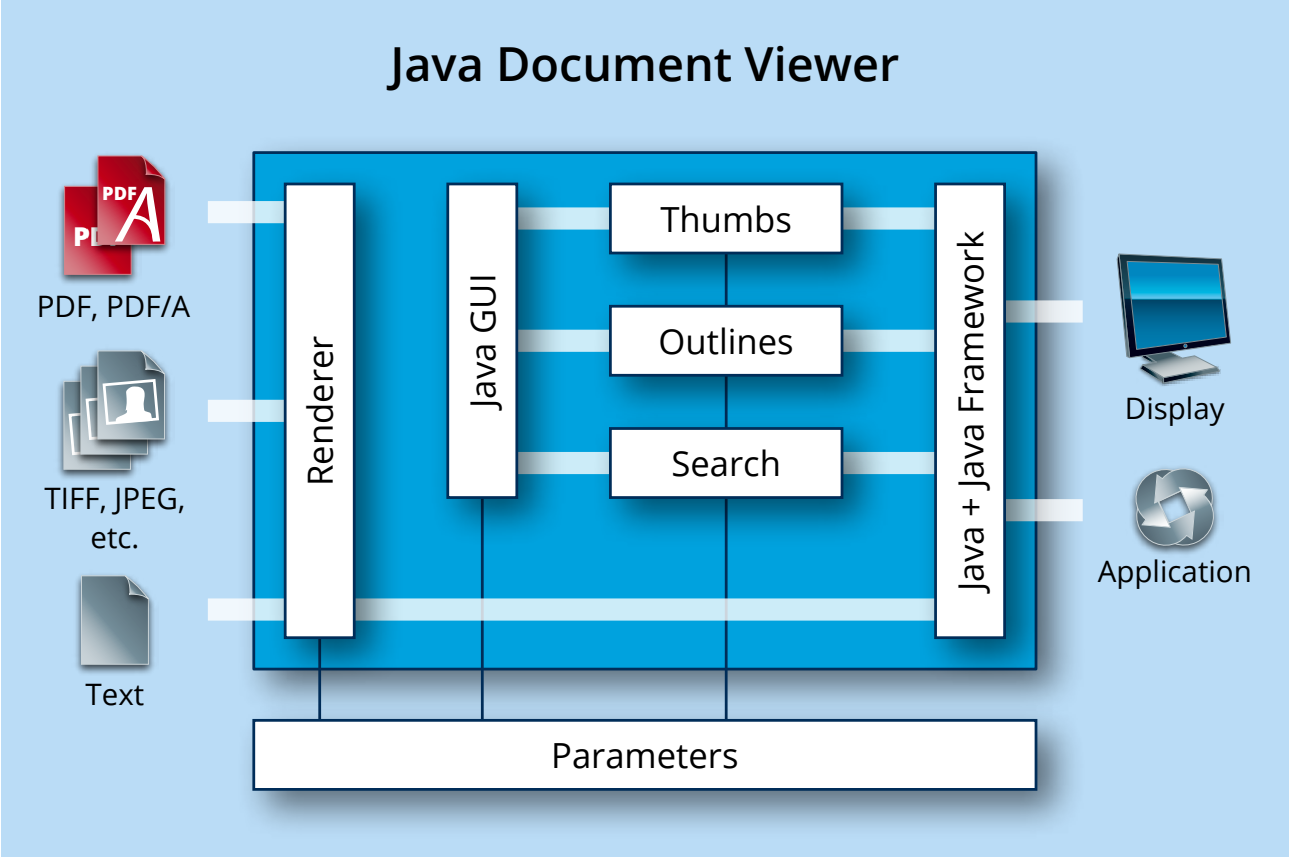
1 Introduction 1
1.1 Description 1
1.2 Functions 2
1.3 Platforms 3
2 Installation 3
2.1 Troubleshooting 4
2.2 Unix 7
3 Design Principles 8
4 License Management 9
4.1 Graphical License Manager Tool 9
4.2 Command Line License Manager Tool 10
4.3 License Key Storage 10
5 User's Guide 11
5.1 Model-View Design 11
5.2 Setting Up the GUI 16
5.3 Menubars and Buttons 17
5.4 Handling Events 18
5.5 Open and Close a Document 19
5.6 Draw a Page 21
5.7 Rotation 21
5.8 Go to Page 22
5.9 Fit Modes 23
5.10 Zoom In and Out 25
5.11 Difference View 27
5.12 Extract Text 28
5.13 Find Text 28
6 Licensing and Copyright 30
7 Contact 30
A Appendix Appendix 30
A.1 Font Replacement Strategy 30

1 Introduction

1.1 Description

The 3-Heights™ Java Document Viewer is a quick and easy method of integrating flexible and customized viewer functionality in an application. Elements such as comments or highlights can be superimposed on the PDF, PDF/A, image and text files displayed on screen.

The user interface is freely designable and can be equipped with your own operating elements. This component is suitable for document management and enterprise content management systems, among others.



1.2 Functions

The 3-Heights™ Java Document Viewer can display and print images and documents and convert them into raster images. Pages are navigated by page number or the use of bookmarks or links. Displayed pages are freely scalable and visual comparisons between two documents are possible. Another tool is text extraction for full text search functionality. Extractable text can be displayed in a specific display mode. Elements such as comments or highlights from separate sources can be superimposed on the display. The Viewer also supports Asian writing systems, fill patterns, color gradients and transparencies.

Features

- Rendering of PDF documents (in accordance with the draft standard ISO 32000 (PDF 1.7))
- Rendering of PDF/A documents in accordance with ISO standard 19005-1
- Rendering of image files (TIFF, JPEG, BMP, JB2, JP2000, JPX, PNG, GIF)
- Rendering of text files (ASCII and Unicode UTF-16)
- Adaptable sample application as Java source code with customized menus, pushbuttons and much more
- Broad range of basic functions such as browse, scroll, zoom, rotate
- Page navigation by entering the page number or through the use of bookmarks or links
- Scaling (free scaling, no scaling, fit to page width, fit entire page)
- Display one or more pages, e.g. as miniature images (thumbnails)
- Display mode for displaying extractable text for use in digital signature applications
- Text extraction
- Text searching with regular expressions and high-lighting of multiple occurrences
- Visual comparison of two files
- Supports streaming interfaces for integrating components, for instance in archive systems
- Rendering of Chinese, Japanese and Korean writing systems (CJK)
- Rendering of fill patterns, color gradients and transparency

Formats

- PDF
- PDF/A
- Text files (ASCII, Unicode UTF-16)
- BMP (1, 2, 4, 8, 24 bit)
- GIF (2 to 8 bit)
- JBIG2 (lossy compression, lossless compression)
- JPEG (Grayscale, RGB, CMYK)
- JPEG2000 and JPEG-LS (Grayscale, RGB, CMYK)
- PBM (1 to 8, 24 bit)
- PNG (1 to 8, 24 bit)
- TIFF
  - Bitonal: uncompressed, CCITT G3, CCITT G3-2D, CCITT G4, LZW, ZIP, Packbits
  - Grayscale: uncompressed, LZW, JPEG, JPEG (old), ZIP, Packbits
  - RGB: uncompressed, LZW, JPEG, JPEG (old), ZIP, Packbits
  - CMYK: uncompressed, LZW, JPEG, JPEG (old), ZIP, Packbits

Compliance

- ISO 19001-1 (PDF/A-1)
- ISO 32000 (PDF 1.7)

1.3 Platforms

- Windows XP, Vista, 7, 8, 8.1 - 32 and 64 bit
- Windows Server 2003, 2008, 2008 R2, 2012, 2012 R2 - 32 and 64 bit
- HP-UX 11 and later PA-RISC2.0 32 bit or HP-UX 11i and later ia64 (Itanium) 64 bit
- IBM AIX 5.1 and later (64 bit)
- Linux (32 and 64 bit)
- Mac OS X 10.4 and later (32 and 64 bit)
- Sun Solaris 2.8 and later, SPARC and Intel
- FreeBSD 4.7 and later 32 bit or FreeBSD 9.3 and later 64 bit (on request)

2 Installation

- Download the ZIP archive of the product from your download account at www.pdf-tools.com.
- Open the ZIP archive.
- 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\`).
- The unzip process now creates the following subdirectories and files:
  `bin\PDFViewerAPI.dll` This DLL contains the main functionality (required).
  `bin\pdcjk.dll` This DLL contains support for Asian languages (optional). It is loaded from the module path.
  `doc\*` Contains documentation files.
  `jar\vwja.jar` Java API archive.
  `app\PdfViewerApp.java` A Java Swing application.

Installation and Running on the CMD Shell

There are different ways how to compile and run a Java application, here is one:

1. Set the environment variable CLASSPATH to include the Jar file *vwja.jar*. (Note: do not use blanks before and after the equal sign.)
   ```
   set CLASSPATH=%CLASSPATH%;C:\pdf-tools \JavaViewer\jar\vwja.jar
   ```
   or set it new, ignoring the old setting:
   ```
   set CLASSPATH=.;C:\pdf-tools\JavaViewer \jar\vwja.jar
   ```
2. Compile the application.
   ```
   javac PdfViewerApp.java
   ```
3. Adjust the environment variable PATH to include the directory where the file *PdfViewerAPI.dll* resides.
   ```
   set PATH=%PATH%;C:\pdf-tools\JavaViewer\bin
   ```
4. Run the application.
   ```
   javaw PdfViewerApp
   ```
   Optionally with a file name as parameter:
   ```
   javaw PdfViewerApp C:\pdf\somefile.pdf
   ```

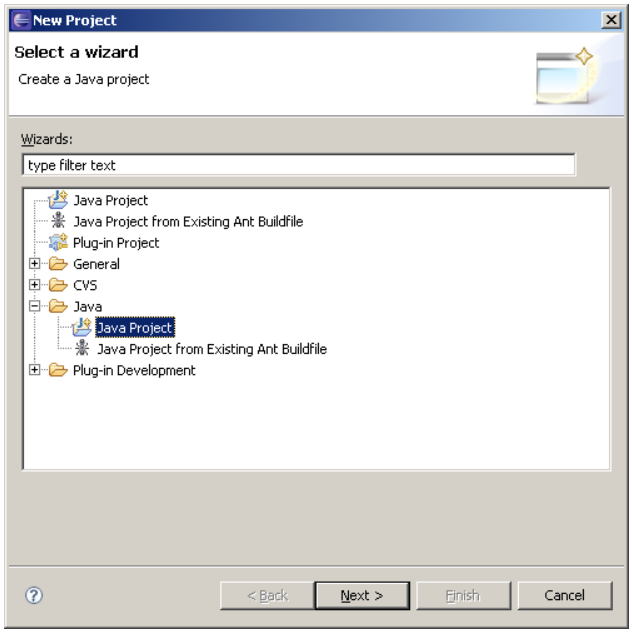
2.1 Troubleshooting

Message	`Exception in thread "main" java.lang.UnsatisfiedLinkError: no PdfViewerAPI in java.library.path`
Problem	The file *PdfViewerAPI.dll* cannot be found.
Solution	Add the directory where it resides to the PATH.
Message	`PdfViewerApp.java:29: package com.pdftools.pdfviewer does not exist`
Problem	The *vwja.jar* file cannot be found.
Solution	Add it to the CLASSPATH.
Message	`Exception in thread "main" java.lang.ExceptionInInitializerError Caused by: java.lang.Exception: The library version 1.6.0.37 doesn't match with the java interface version 1.6.0.36.`
Problem	Problem: The *vwja.jar* and *PdfViewerAPI.dll* are of two different versions.
Solution	Use files from the same version and make sure you are referring to those when running the application.

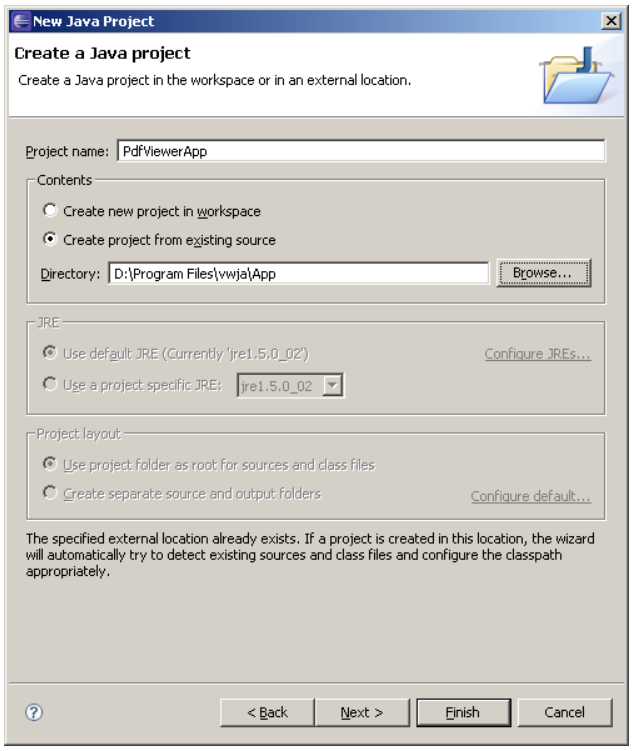
Installation for Eclipse

(The following steps are described for the Windows Eclipse 3.2.)

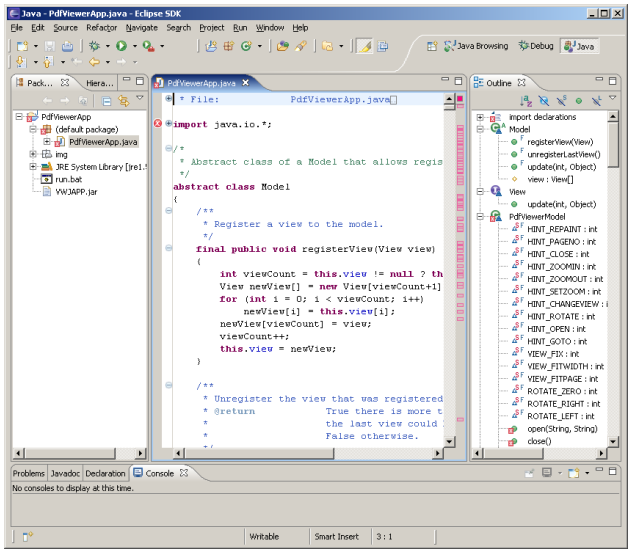
1. Start Eclipse.
2. Create a new Project by selecting “File” ->“New” ->“Project..”
3. Select item “Java Project”.
4. Press button “Next”.



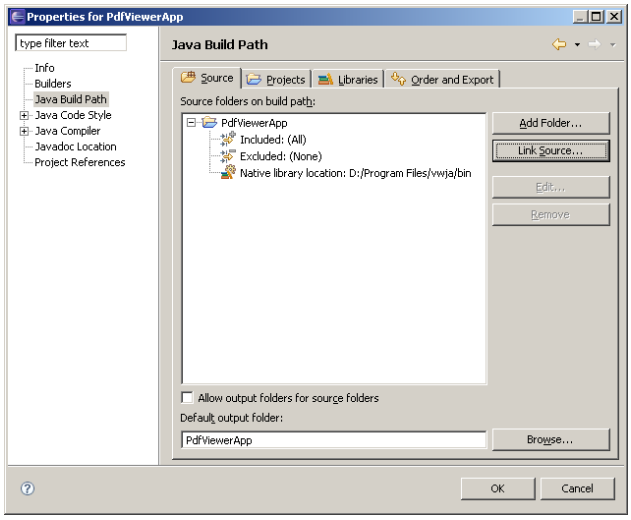
5. Name the project “PdfViewerApp”.
6. Select the radio button “Create project from existing source”.
7. Set the directory to where the file PdfViewerApp.java resides.
8. Press button “Finish”.



9. A project is now created, that should look similar as in the screenshot to the right.
10. Right-click the project “PdfViewerApp” and select properties.

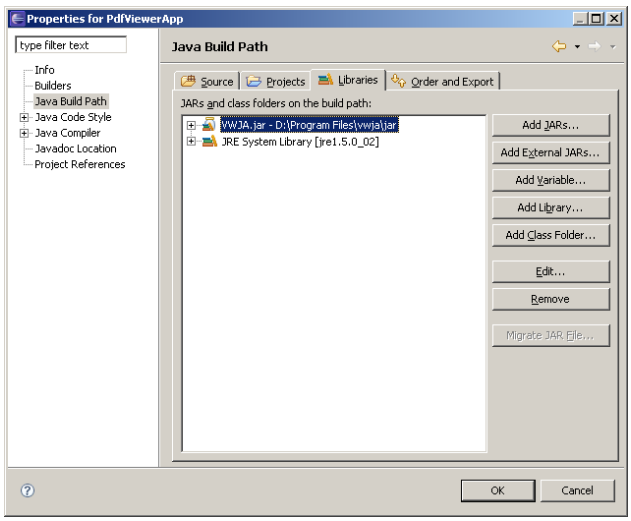


11. On the left side select “Java Build Path”.
12. On the right side select the tab “Source”.
13. Set the native library location to where the PDFViewerAPI.dll resides.

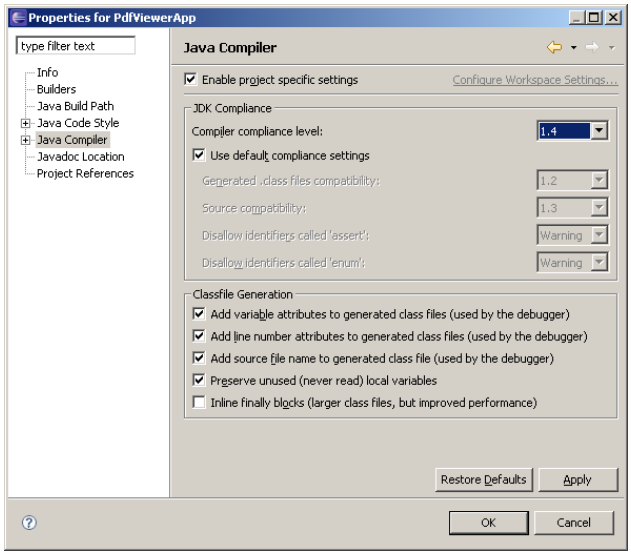


14. On the right side select the tab “Libraries”.

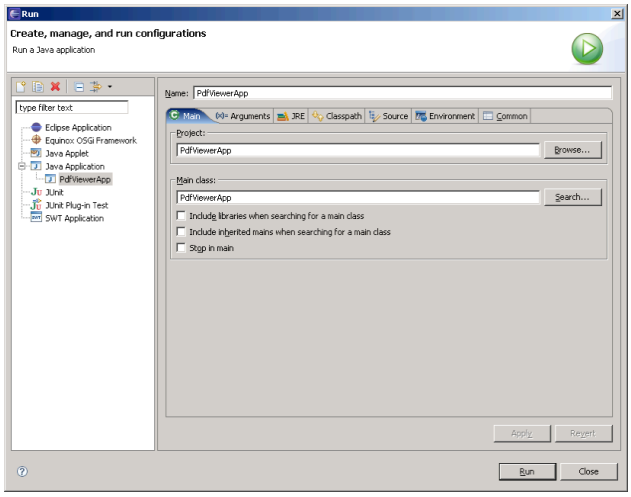
15. Add the Jar file “vwja.jar”.



16. Optional: Set the JDK Compliance level to 1.4 as indicated in the screenshot to the right.



17. From the menu “Run” select “Run..”
18. Name the configuration (e.g. “PDFViewerApp”).
19. Select the project “PDFViewerApp”.
20. Select the main class “PDFViewerApp”.
21. Press button “Run”.
22. We are done.



2.2 Unix

This section describes installation steps required on all Unix platforms, which includes Linux, Mac OS X, Sun Solaris, IBM AIX, HP-UX, FreeBSD and others.

Here is an overview of the shared libraries and other files that come with the 3-Heights™ Java Document Viewer:

Table: File Description

Name	Description
`bin/libPdfViewerAPI.so`	This is the shared library that contains the main functionality (required). The file extension varies depending on the UNIX platform.
`doc/*.pdf`	Various documentation
`app/VWJA.jar`	The Java wrapper
`gui/PdfViewerApp.java`	A Java Swing application.

Example code written in different programming languages are available at product page of the PDF Tools AG website (http://www.pdf-tools.com).

All Unix Platforms

1. Unpack the archive in an installation directory, e.g. */opt/pdf-tools.com/*
2. Copy or link the shared object into one of the standard library directories, e.g:
   `ln -s /opt/pdf-tools.com/bin/libPdfViewerAPI.so /usr/lib`
3. Verify that the GNU shared libraries required by the product are available on your system:
   - *On Linux*: `ldd libPdfViewerAPI.so`
   - *On AIX*: `dump -H libPdfViewerAPI.so`

   In case you have not installed the GNU shared libraries yet, proceed as follows:
   (a) Go to http://www.pdf-tools.com and navigate to "Support" → "Resouces".
   (b) Download the GNU shared libraries for your platform.
   (c) Extract the archive and copy or link the libraries into your library directory, e.g */usr/lib* or */usr/lib64*.
   (d) Verify that the GNU shared libraries required by the product are available on your system now.
4. Optionally register your license key using the Command Line License Manager Tool.
5. Add the directory containing *libPDFViewerAPI.dylib* to the *DYLD_LIBRARY_PATH.*
6. Add the *jar/VWJA.jar* file to the CLASSPATH.

MAC OS/X

The shared library must have the extension .jnilib for use with Java. We suggest that you create a file link for this purpose by using the following command:

```
ln libPdfViewerAPI.dylib libPdfViewerAPI.jnilib
```

3 Design Principles

1. The native code contains the implementation of the graphics engine independent objects (PdfViewerAPI.dll).
   a. For performance reasons as much as possible, e.g. converting colors, is done in the native code.
   b. The contents of a page are represented by two types of objects: resources and operators which are created by the native code.
2. The Graphcis2D specific code is concentrated in the compiled Java code (vwja.jar).
   a. The interface design follows a suitable subset of the PDF semantics; the subset is chosen such that the objects can easily be used by Graphics2D, e.g. only sRGB colors are used.
   b. The interface to the native code consists of a few, easy-to-use classes:
      i. PdfDocument: Opens a document and provides access to its pages.
      ii. PdfPage: Provides access to the page's attributes and operator stream.
      iii. PdfTextExtractor: Returns the text contained in a page.
      iv. PdfTextFinder: Finds the text contained in a page and matching a regular pattern.
      v. PdfPageFinder: Finds all pages containing text matching a regular pattern.
3. The user interface is provided as Java source code (AWTViewerAdvanced, AWTViewerSimple). It contains the code for the following features:
   a. Display a pages
   b. Links and go to link destination
   c. Scroll, zoom, fit modes and rotate
   d. Navigate: first, next, previous, last page, go to page
   e. Keyboard navigation
   f. Mouse scroll, zoom, move and activate link
   g. Bookmark tree
   h. Page thumbnails
   i. Find text, highlight and go to next occurrence
4. Some other applications of the component such as printing and conversion of a PDF to an image are provided as source code as well (AWTConverter, AWTPrinter).

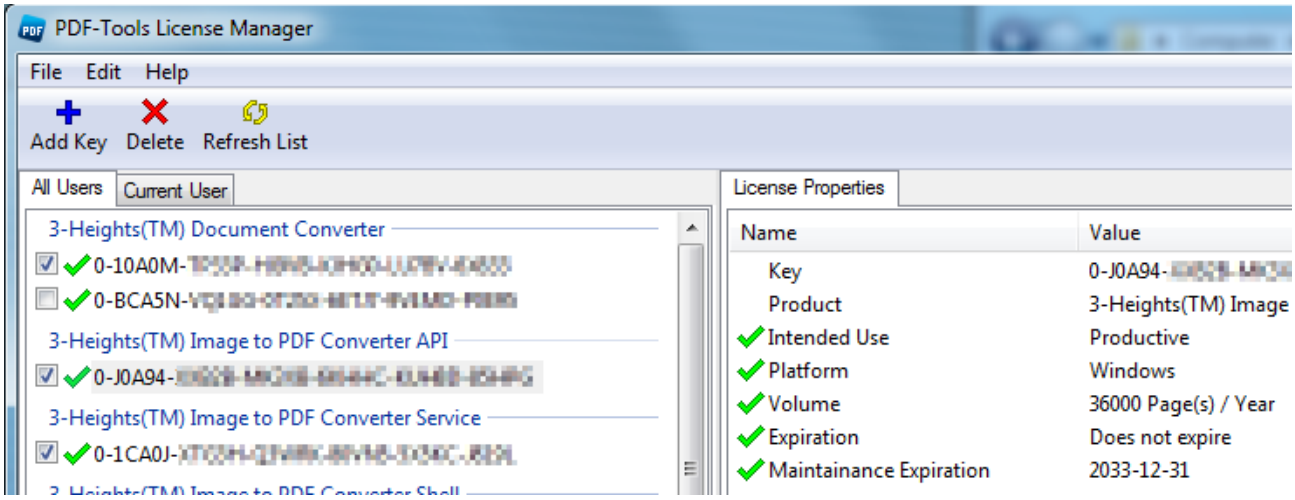
4 License Management

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.

4.1 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.

4.2 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
```

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
```

4.3 License Key Storage

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

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)

Mac OS X

The license keys are stored in the file system:

- `/Library/Application Support/PDF Tools AG` (for all users)
- `~/Library/Application Support/PDF Tools AG` (for the current user)

Unix/Linux

The license keys are stored in the file system:

- `/etc/opt/pdf-tools` (for all users)
- `~/.pdf-tools` (for the current user)

Note: The user, group and permissions of those directories are set explicitly by the license manager tool. It may be necessary to change permissions to make the licenses readable for all users. Example:

```
chmod -R go+rx /etc/opt/pdf-tools
```

5 User's Guide

There is an extensive GUI application called PDFViewerApp.java included in the 3-Heights™ Java Document Viewer (Release and Evaluation Versions) that uses Java Swing. It shows how to display pages in certain modes like fit-page, fit-width or original size, it shows how to scroll, zoom, rotate a page or create an outline (bookmark) tree, etc. The idea of this chapter is to understand how this application is built and how change can be applied or enhancements can be added.

Note: Code snippets in this manual might differ from the actual code, due to modifications or enhancements.

5.1 Model-View Design

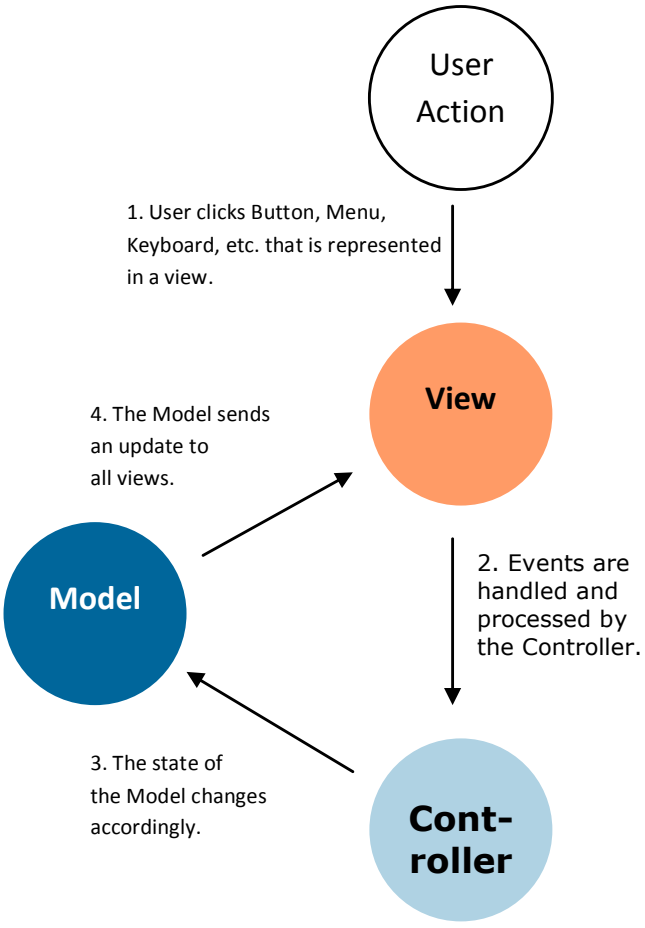
The idea of introducing a model and a view is to divide the code into the common Swing architecture that consists of a Model, a View and a Controller. There are many articles about the so called MCV architecture, one can be found at http://www.oracle.com/technetwork/java/architecture-142923.html.

Here is a brief overview:

The Model The model represents the data of the application. It keeps track of which file currently is open, how many pages there are, and the current page.

The View The visual representation of the data. It displays the actual page. There may be multiple views, for example a main view and a thumbnail view. Different views can potentially display different pages at different zoom levels and with different rotations. Therefore view specific information is saved in the view itself.

The Controller The controller handles interactions with the user. Inputs from views are processed and forwarded to update the model.



The Model

The model must inform all views about changes in the model. e.g. if the document is closed and another document is opened, the data in the model changes and all views must be informed and react adequately, e.g. display the new document.

The abstract class *Model* provides the functionality to register views. All views that are registered in the model are sent an update if the model has been updated.

The derived class *PdfViewerModel* contains a *PdfViewer* and the current page number as member variables. The class *PdfViewer* contains information about the actual document, such as the filename or the number of total pages. It provides functions to *open* and *close* a document as well as to go to a certain page number or changing the way how to display a page. Static hints that are provided with the update function give additional information what and how the view must update.

```
abstract class Model
{
    // View Registering/Update
    final public void registerView(View view)
    final public boolean unregisterLastView()
    final public void update(int hint, Object param)
    // Member Variables
    protected View view[];
}
```

```
class PdfViewerModel extends Model
{
    // Document Open/Close
    public boolean open(String fileName, String password)
    public void close()
    // Goto Page
    public void goFirstPage()
    public void goLastPage()
    public void goNextPage()
    public void goPreviousPage()
    public void goPage(int pageNo)
    public void goTo(int pageNo, float x, float y, float z)
    // Zooming/Scaling/Rotating
    public void zoomIn()
    public void zoomOut()
    public void setZoom(float Zoom)
    public void setActualSize()
    public void setFitWidth()
    public void setFitPage()
    public void rotateRight()
    public void rotateLeft()
    public void rotateZero()
    // TextMode
    public void setTextMode(boolean bsetTextMode)
    // Member Variables
    protected PdfViewer viewer;
    protected int pageNo;
    protected Rectangle visibleRect;
}
```

The View

The view visualizes the data. For a Java Document Viewer, the most obvious way how to achieve this is to draw the page content. The class *PdfViewerPageView* does exactly that. Its *update* function reacts to commands sent by the model. It contains a paint function that can actually draw the current page when needed. It contains several member variables, the three most important are: The current *pageNo* as integer, the current *page* object and a *PdfViewerApp*. You might ask why the page number has to be stored in the view as well as in the model? There are different reasons for that. The page number in the model represents what is being considered the current page, i.e. the page number that is displayed in the main view. Different views however might display different page numbers, the main view can display page 1, whereas the thumbnail view might currently display pages 4 to 8. When the user selects a new page number using for example the thumbnail view interface, the page number of the model changes, and as a result the page number in the views are updated as well.

There is also an outline (bookmark) and list (page list) which provide textual views of the pages.

```
interface View
{
    abstract public void update(int hint, Object param);
}
```

```
class PdfViewerPageView extends JComponent implements View,
ActionListener,MouseListener
{
    // Constructor
    PdfViewerPageView(PdfViewerApp app)
    // Painting
    private void paintDescription(String description, Point pt)
    private void paintMarkRect()
    public Dimension getPreferredSize()
    public void paint(Graphics g)
    // Class Specific/Helper
    private PdfDestination getDestAt(MouseEvent e)
    private AffineTransform getTrans(boolean isInverted)
    // Events/Update
    public void mouseClicked(MouseEvent e)
    public void mouseEntered ...
    public void actionPerformed(ActionEvent a)
    public void update(int hint, Object param)
    // Member Variables
    protected final PdfViewerApp app;
    protected PdfViewerPage page;
    protected int pageNo;
    protected float zoomFactor;
    protected int pageView;
    protected int viewingRot;
    private final float dpiFactor;
    private Point p1, p2;
    private PdfAnnotation links[];
    private Rectangle rectPrev, rectCur;
    private int curserMode;
}
```

```
class PdfViewerThumbnailView extends Component implements View, MouseListener
{
    // Painting
    public Dimension getPreferredSize()
    public void paint(Graphics g)
    // Events/Update
    public void mouseClicked(MouseEvent e)
    public void mouseEntered ...
    public void update(int hint, Object param)
    // Member Variables
    protected PdfViewerPage page;
    protected final int thumbSize;
    protected final PdfViewerApp app;
}
```

```
abstract class PdfTreeView extends JPanel implements View
{
    // Constructor
    PdfTreeView(PdfViewerApp app)
    // Tree
    abstract protected void buildTree();
    protected void expand(JTree tree, DefaultMutableTreeNode node)
    // Update
    public void update(int hint, Object param)
    public Dimension getPreferredSize()
    // Member Variables
    protected JTree tree;
    protected final PdfViewerApp app;
}
```

```
class PdfOutlineView extends PdfTreeView
{
    // Constructor
    PdfOutlineView(PdfViewerApp app)
    // Tree
    protected void buildTree()
    // Member Variables
    protected JTree treeExp;
}
```

```
class PdfListView extends PdfTreeView
{
    // Constructor
    PdfListView(PdfViewerApp app)
    // Tree
    protected void buildTree()
}
```

The Controller

The controller launches the application. It sets up the GUI and processes user actions in its function *actionPerformed*.

```
// Main Class
public final class PdfViewerApp extends JFrame implements View, ActionListener
{
    // Constructor
    public PdfViewerApp(String fileName, String password)
    // Main
    public static void main(String args[])
    // GUI Builders
    public void showView(int showViewNew)
    private final void makeMenu()
    private final void addMenuFile(JMenuBar menuBar)
    private final void addMenuDocument(JMenuBar menuBar)
    private final void addMenuView(JMenuBar menuBar)
    private final void makePanel()
    // Class Specific/Helper
    public boolean isDisableLink()
    private ImageIcon loadIcon(String name)
    // Events/Update
    public void update(int hint, Object param)
    public void actionPerformed(ActionEvent e)
    public Dimension getPreferredSize()
    // Member Variables
    protected static int showView;
    protected JScrollPane thumbPane, outlinePane, listPane;
    protected final PdfViewerModel model;
    protected String fileName;
}
```

```
// Helper Classes
class PdfViewerScrollPane extends JScrollPane
{
    PdfViewerScrollPane(PdfViewerPage View component)
    protected final PdfViewerPageView view;
}

class JTextFieldZoom extends JTextField
{
    JTextFieldZoom(int columns, final ActionListener listener)
}
```

5.2 Setting Up the GUI

The graphics user interface (GUI) is setup in the constructor of the *PdfViewerApp*. The relevant steps done are as following:

1. Input parameters are parsed for in the *main* function and a file name and password is set if provided.
2. The main function instantiates a *PdfViewerApp* which is derived from *JFrame*.
3. A model is instantiated. Any view will be registered to the model and be notified by the model using the *update* function.
4. The *PdfViewerApp* implements a *view* itself in (order to update the frame title if a new document is opened). It is therefore registered in the model.
5. In the next step a menu bar and its menu icons are created.
6. A Panel with buttons is added.
7. The main-window-view *PdfViewerPageView* is instantiated and registered in the model.
8. A derived *ScrollPane* is instantiated and added the *PdfViewerPageView*. The scroll pane is added to the content pane of the frame.
9. A document is opened if initially provided.
10. The frame is displayed.

```
public PdfViewerApp(String fileName, String password)
{
3.      model = new PdfViewerModel();
4.      model.registerView(this);

5.      makeMenu();
6.      makePanel();

7.      PdfViewerPageView mainView = new PdfViewerPageView(this);
        model.registerView(mainView);

8.      PdfViewerScrollPane viewerPane = new PdfViewerScrollPane(mainView);
        getContentPane().add(viewerPane, "Center");
        viewerPane.setMinimumSize(new Dimension(10, 10));

        setTitle("PDF Viewer");
9.      if (fileName.length() > 0)
        {
            this.fileName = fileName;
            model.open(fileName, password);
        }
        showView(PdfViewerApp.showOutline);
}

public static void main(String args[])
{
        String fileName = "";
        String password = "";
1.      if (args.length >= 1)
            fileName = args[0];
        if (args.length >= 2)
            password = args[1];
        showView = PdfViewerApp.showNone;
2.      PdfViewerApp app = new PdfViewerApp(fileName, password);
10.     app.pack();
        app.setResizable(true);
        app.setVisible(true);
```

```
    app.setDefaultCloseOperation(Window Constants.EXIT_ON_CLOSE);
}
```

5.3 Menubars and Buttons

How to add menu bars and buttons is not really related to the PDF Java Viewer, however they are required to provide the user the possibility to interact with the application. Most functionalities require an interaction, therefore this is described first.

1. Menu and Panel related objects (JMenuItem, JMenu, JCheckBoxMenuItem, JLabel, etc.) are defined in the Controller.
2. The Menu and Panel are created in functions that are called by the constructor of *PdfViewerApp*.
3. *makeMenu* calls three sub-functions. Each of those creates one menu including all menu items. Also key shortcuts are added.
4. *makePanel* creates a panel and adds buttons and labels onto it. Most buttons use an image which is imported using the helper function loadIcon.

```
public final class PdfViewerApp extends JFrame implements View, ActionListener
{
1.      private JMenuItem menuItem_Open, menuItem_Close, ...
        private JMenu menu_ZoomTo, menu_Navigation;
        private JCheckBoxMenuItem checkBoxMenuItem_TextMode,
        ...

        public PdfViewerApp(String fileName, String password)
        {
            ...
2.          makeMenu();
            makePanel();
            ...
        }

3.      private final void makeMenu()
        {
            JMenuBar menuBar = new JMenuBar();
            addMenuFile(menuBar);
            addMenuDocument(menuBar);
            addMenuView(menuBar);

            setJMenuBar(menuBar);
        }

        private final void addMenuFile(JMenuBar menuBar)
        {
            JMenu menu = new JMenu("File");
            menu.setMnemonic(KeyEvent.VK_F);
            menuBar.add(menu);

            menuItem_Open = new JMenuItem("Open", KeyEvent.VK_O);
            menuItem_Open.setAccelerator( KeyStroke.getKeyStroke(KeyEvent.VK_O,
              ActionEvent. ALT_MASK));
            menuItem_Open.addActionListener(this);
            menu.add(menuItem_Open);
            menuItem_Close = new JMenuItem("Close", KeyEvent.VK_C);
            ...
```

```
    }

    private final void addMenuDocument(JMenuBar menuBar)

    private final void addMenuView(JMenuBar menuBar)

    private ImageIcon loadIcon(String name)
    {
        return new ImageIcon(PdfViewerApp. class.getResource("gfx/"+ name));
    }
```

4.

```
    private final void makePanel()
    {
        // Buttons
        JPanel panel = new JPanel();
        panel.setLayout(new FlowLayout(FlowLayout.LEFT));
        button_Open = new JButton(loadIcon("open.gif"));
        button_Open.addActionListener(this);
        button_First = new JButton(loadIcon("first.gif"));
        button_First.addActionListener(this);
        ...

        // Page number
        ...

        getContentPane().add(panel, "North");
    }
}
```

5.4 Handling Events

Events (open file, go to page, etc.) triggered by a menu or a button are caught by the function *actionPerformed* in the controller. An event is processed and results in a change of the state of the model.

```
public void actionPerformed(ActionEvent e)
{
    if (e.getSource() == menuItem_Open || e.getSource() == button_Open)
    {
        final JFileChooser fc = new JFileChooser();
        if (fc.showOpenDialog(this) == JFileChooser.APPROVE_OPTION)
        {
           final String filename = fc.getSelectedFile(). getAbsolutePath();
        if (model.open(filename, ""))
           model.setTextMode(checkBox MenuItem_ TextMode.getState());
        this.repaint();
        }
    }
    else if (e.getSource() == menuItem_Close)
       model.close();
    ...
}
```

5.5 Open and Close a Document

The currently opened document is stored in the model.

1. Closing a document or opening a new document is handled by the model.
2. A call to *update* informs all views. A hint given as parameter in the *update* function provides additional information, i.e. what exactly happened (document closed, document opened, etc.).
3. Each active view handles the updates individually. For a view it is not required to address all update calls. The main view (*PdfViewerPageView*) for example handles the close-hint, but not the open-hint. For this view it is only relevant when something happens to the active page. This is the case when no page needs to be drawn (close), however it does not happen with the open-hint, but with the subsequent pageno-hint
4. which results as a consequence of *goFirstPage*.

```
class PdfViewerModel extends Model
{
    final static int HINT_CLOSE = 4;
    final static int HINT_OPEN = 256;

1.  public boolean open(String fileName, String password)
    {
        if (viewer != null)
           close();
        try
        {
             viewer = new PdfViewer(fileName, password);
2.           update(PdfViewerModel.HINT_OPEN, fileName);
             setZoom(1.0f);
4.           goFirstPage();
        }
        catch (FileNotFoundException e)
        {
             return false;
        }
        return true;
    }

    public void close()
    {
        if (viewer != null)
        {
             pageNo = 0;
2.           update(HINT_CLOSE, new Integer(0));
             viewer.destroyObject();
             viewer = null;
        }
    }
}
```

```
class PdfViewerPageView extends JComponent implements View, ActionListener,
  MouseListener
{
3.  public void update(int hint, Object param)
    {
        if ((hint & PdfViewerModel.HINT_CLOSE) != 0)
        {
            if (page != null)
```

```
            {
                page.destroyObject();
                page = null;
                pageNo = 0;
                viewingRot = 0;
                repaint();
                setSize(getParent(). getSize());
            }
        }
        else if ((hint & PdfViewerModel.HINT_PAGENO) != 0)
        {
            final int pageNo = ((Integer) param).intValue();
            if (pageNo < 1 || pageNo > app.model.viewer. getPageCount())
                return;

            if (this.pageNo != pageNo)
            {
                this.pageNo = pageNo;
                if (page != null)
                    page.destroyObject();
                page = app.model.viewer. getPage(pageNo);
                viewingRot = page.getRotate();
                repaint();
            }
        }
    }
}
```

3.
```
class PdfViewerThumbnailView extends Component implements View, MouseListener
{
    public void update(int hint, Object param)
    {
        if ((hint & PdfViewerModel.HINT_OPEN) != 0)
        ...

        else if ((hint & PdfViewerModel.HINT_CLOSE) != 0)
        ...
    }
}
```

5.6 Draw a Page

Drawing a page is done using the *paint* function if the view.

1. In a first step the currently viewed window of the page must be determined. This is done using the function *getSize*. *getSize* returns the size of the page in form of a dimension object. The size was previously set using *setSize* and depends on the current zoom factor.
2. The content of the page is drawn into a given viewport container using a Graphics 2D container.
   What is a viewport? The viewport is the region of a frame to be displayed. Content outside the viewport is not displayed.
3. If no page is active, a blue background rectangle is drawn instead.

```
class PdfViewerPageView extends JComponent implements View, ActionListener,
  MouseListener
{
    public void paint(Graphics g)
    {
1.      final Dimension size = getSize();
        if (page != null)
        {
            final Rectangle rect = new Rectangle(0, 0, size.width, size.height);
2.          page.draw((Graphics2D) g, rect);
            ...
        }
3.      else
        {
            g.setColor(new Color(174, 209, 226));
            g.fillRect(0, 0, size.width, size.height);
        }
    }
}
```

5.7 Rotation

When it comes to rotation, it needs to be considered that each page in a PDF document may have an individual viewing rotation. Often the viewing rotation attribute is abused to display a rotated portrait page as landscape. The initial viewing rotation and the user rotation define the resulting rotation. Here is how it works:

1. The User triggers an event which is caught by the controller. The controller calls the corresponding function in the model, e.g. *rotateRight*.
2. *rotateRight* updates all views by sending a rotate-hint.
3. All views, which should rotate pages, have the appropriate hint handled by their update function. Rotation is usually only handled by the main view, the thumbnail view should rather stay not rotated.
4. The view rotates the page by the selected value, the initial rotation (*viewingRot*) and previous rotations (*page.getRotate*) need to be considered.
5. The page needs to be repainted.

```
class PdfViewerModel extends Model
{
    final static int HINT_ROTATE = 128;
    final static int ROTATE_ZERO = 0; // Set rotation to 0
    final static int ROTATE_RIGHT = 1; // Rotate by 90 degrees
    final static int ROTATE_LEFT = 2; // Rotate by -90 degrees
1.  public void rotateRight()
    {
        if (viewer != null)
```

```
2.              update(HINT_ROTATE, new Integer(ROTATE_RIGHT));
    }

    public void rotateLeft()

    public void rotateZero()
}
```

```
class PdfViewerPageView extends JComponent implements View, ActionListener,
  MouseListener
{
3.  public void update(int hint, Object param)
    {
        ...
        else if ((hint & PdfViewerModel.HINT_ROTATE) != 0)
        {
            final int rotate = ((Integer) param).intValue();
            if (rotate == PdfViewerModel.ROTATE_RIGHT)
4.              page.setRotate(page.getRotate() - viewingRot + 90);
            else ...
            setSize(getPreferredSize());
5.          repaint();
        }
    }
    protected int viewingRot;
}
When opening a new document, the rotation is reset as one can see in the open-
update part of the previous chapter.
```

5.8 Go to Page

1. The model provides some high level function that jumps directly to a certain page number (first, last, next, previous).
2. They are all interpreted to end up in a call to *goPage*, which sets an actual page number.
3. *goPage* verifies the page is within a valid range and sends an update to all views. (For the update of the pageno-hint, see chapter “Open and Close a Document”.)

```
class PdfViewerModel extends Model
{
1.  public void goFirstPage()
    {
        if (viewer != null)
        {
            pageNo = 1;
            goPage(pageNo);
        }
    }
    public void goLastPage()

    public void goNextPage()

    public void goPreviousPage()

2.  public void goPage(int pageNo)
```

```
    {
        if (viewer != null)
        {
3.          if (pageNo < 1)
                pageNo = 1;
            else if (pageNo > viewer.getPageCount())
                pageNo = viewer.getPageCount();
            this.pageNo = pageNo;
            update(HINT_PAGENO, new Integer(pageNo));
        }
    }
}
```

5.9 Fit Modes

The PDF Java Viewer GUI application supports three viewing mode:

- Fixed: Means the zoom factor does not depend on the frame size, the content is displayed with a defined size (e.g. actual size).
- Fit width: Means the zoom factor is set so that page width is always adjusted to that it exactly fits the width of the frame.
- Fit page: Means to zoom factor is set so that the full page is always visible.

These fit modes are pure GUI functions that take into account the size of the current page and the size of the view port. There is no fit mode function in the API. The function *getPreferredSize* is usually called before a call to *repaint*. It retrieved the size of the current page and the current view port (depending on the size of the frame). It calculates the zoom factor and returns the dimensions of the page that needs to be drawn.

1. The GUI applications draws the page onto a pane. In the fixed mode, the pane can have any size. The larger the pane, the smaller the relative part that can be displayed given a fixed viewport size and using a *JScrollPane*. In this case the zoom factor is given by the user and does not need to be calculated. *getPreferredSize* returns the dimension of the page at the given zoom factor multiplied by a scaling dpiFactor, to take into account a PDF has a resolution of 72 dpi (dots per inch) where as a monitor has usually a resolution of 96 dpi.
2. In the fit-width mode, the page must always fit within the given width of the pane. Therefore the zoom factor needs to be re-calculated. The dimension that is returned has the appropriate size to paint the page at the given zoom factor so that its width fits in.
3. For the fit-page mode, both width and height have to be considered.

```
class PdfViewerModel extends Model
{
    final static int VIEW_FIX = 1; // Fixed zoom, e.g. Actual size
    final static int VIEW_FITWIDTH = 2; // Fit page width into window
    final static int VIEW_FITPAGE = 3; // Fit page width and height

    public void setActualSize()
    {
        if (viewer != null)
            update(HINT_CHANGEVIEW, new Integer(VIEW_FIX));
    }

    public void setFitWidth()
    {
        if (viewer != null)
            update(HINT_CHANGEVIEW, new Integer(VIEW_FITWIDTH));
    }
```

```
    public void setFitPage()
}

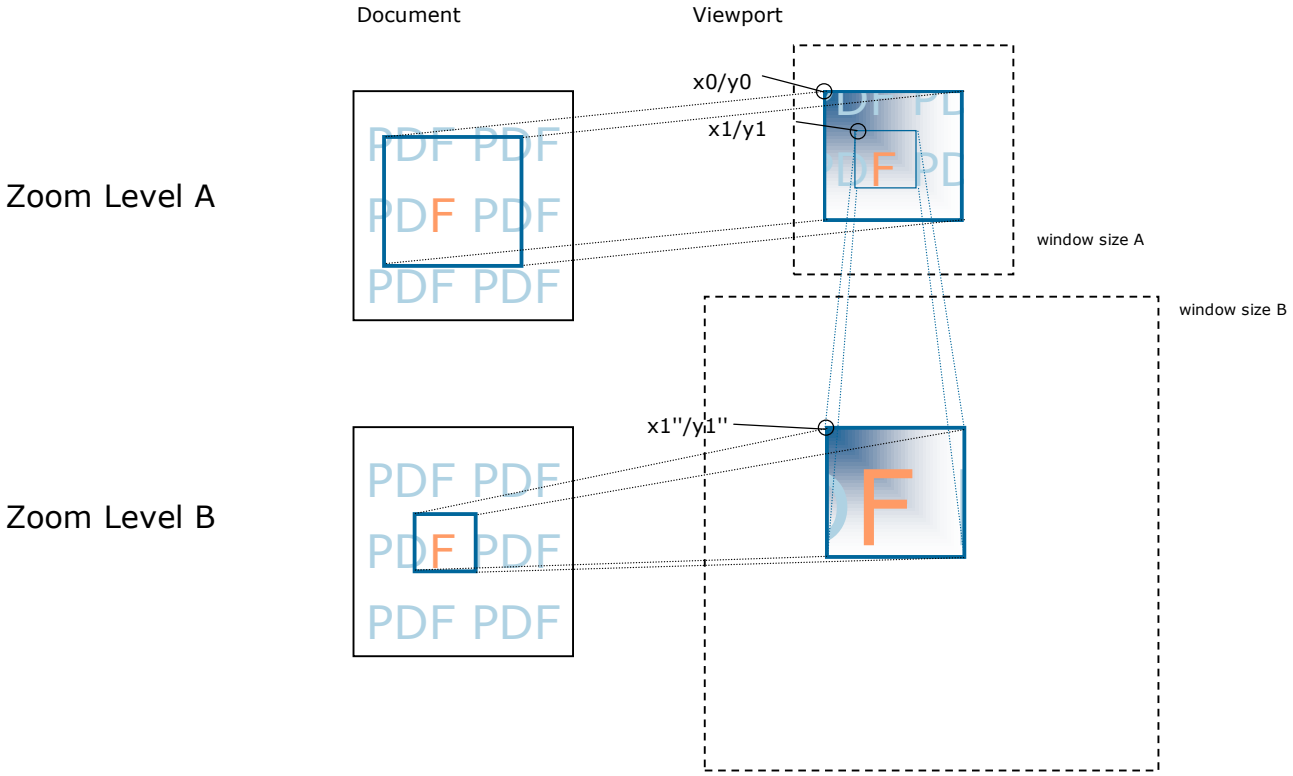
class PdfViewerPageView extends JComponent implements View, ActionListener,
  MouseListener
{
    dpiFactor = 96.0f / 72.0f;

    public Dimension getPreferredSize()
    {
        if (page != null)
        {
            final Rectangle rect = page.getWindowRectangle();
2.              if (pageView == PdfViewerModel.VIEW_ FITWIDTH)
                    zoomFactor = (float)getParent(). getWidth() /rect.width
3.                      /dpiFactor;
                else if (pageView == PdfViewerModel.VIEW_ FITPAGE)
                {
                    final float zoomX = (getParent().getWidth()) /rect.width
1.                      /dpiFactor;
                    final float zoomY = (getParent().getHeight()) /rect.height
                        /dpiFactor;
                    zoomFactor = zoomX < zoomY ? zoomX : zoomY;
                }
                final int w = (int) ((float)rect.width *dpiFactor *zoomFactor);
                final int h = (int) ((float)rect.height *dpiFactor *zoomFactor);
                return new Dimension(w, h);
        }
        return getSize();
    }
}
```

5.10 Zoom In and Out

The GUI applications draws the page onto a pane. Zooming-in means the window size of that pane size is increased, the viewport size remains, therefore displaying a smaller part of the full page.

1. The model provides three functions related to zooming in and out: Increase current zoom level by 10%, decrease by 10% and set to an absolute value.
2. The model sends the corresponding update to the views (only the PdfViewerPageView handles this update).
3. Zooming in and out requires to set a new window and the positioning onto the new window. If one only increases the zoom factor and keeps the coordinates, one does indeed zoom in, however the region displayed is a different one, since the window rectangle has changed. The new position on the new rectangle needs to be calculated. The Code shows one way how to zoom in and out centered.




```
1.  class PdfViewerModel extends Model
    {
        final static int HINT_ZOOMIN = 8; // Change the zoom level +10%
        final static int HINT_ZOOMOUT = 16; // Change the zoom level -10%

        public void zoomIn()
        {
            if (viewer != null)
2.          update(HINT_ZOOMIN, new Integer(0));
        }

        public void zoomOut()
        {
            if (viewer != null)
2.          update(HINT_ZOOMOUT, new Integer(0));
        }

        public void setZoom(float Zoom)
        {
```

2.
```
        if (viewer != null)
        {
            if (Zoom > 0.01f && Zoom < 1000.0f)
            update(HINT_SETZOOM, new Float(Zoom));
        }
    }
}
```

3.
```
class PdfViewerPageView extends JComponent implements View, ActionListener,
  MouseListener
{
    public void update(int hint, Object param)
    {
        ...
        else if ((hint & PdfViewerModel.HINT_ZOOMIN) != 0 ||
            (hint & PdfViewerModel.HINT_ZOOMOUT) != 0)
        {
            pageView = PdfViewerModel.VIEW_FIX;
            final int zoom = hint;

            // 1. Get (x0, y0) in current window size
            final double x0 = getLocation().getX();
            final double y0 = getLocation().getY();
            final int w = getParent().getWidth(); // width of viewport
            final int h = getParent().getHeight();

            // 2. Calculate (x1, y1) the new upper left corner after zooming
            final double zoomMod = zoom == PdfViewerModel.HINT_ZOOMIN ? 1.1 :
                1/1.1;
            final double x1 = x0 - w/2 * (1-1/zoomMod);
            final double y1 = y0 - h/2 * (1-1/zoomMod);
            zoomFactor *= zoomMod;

            // 3. Convert (x1, y1) to ViewPort coordinates (x1', y1')
            AffineTransform trans1 = getTrans(true);
            double[] pt1 = {x1, y1};
            trans1.transform(pt1, 0, pt1, 0, 1);

            // 4. Set size and thereby change the windows size
            setSize(getPreferredSize());

            // 5. Convert (x1', y1') to window coordinates (x1'', y1'')
            AffineTransform trans2 = getTrans(false);
            trans2.transform(pt1, 0, pt1, 0, 1);

            // 6. Set location to (x1'', y1'')
            setLocation((int)pt1[0], (int)pt1[1]);
            repaint();
        }
        else if ((hint & PdfViewerModel.HINT_SETZOOM) != 0)
        {
            pageView = PdfViewerModel.VIEW_FIX;
            zoomFactor = ((Float) param).floatValue();
            setSize(getPreferredSize());
            repaint();
```

```
        }
    }
}
```

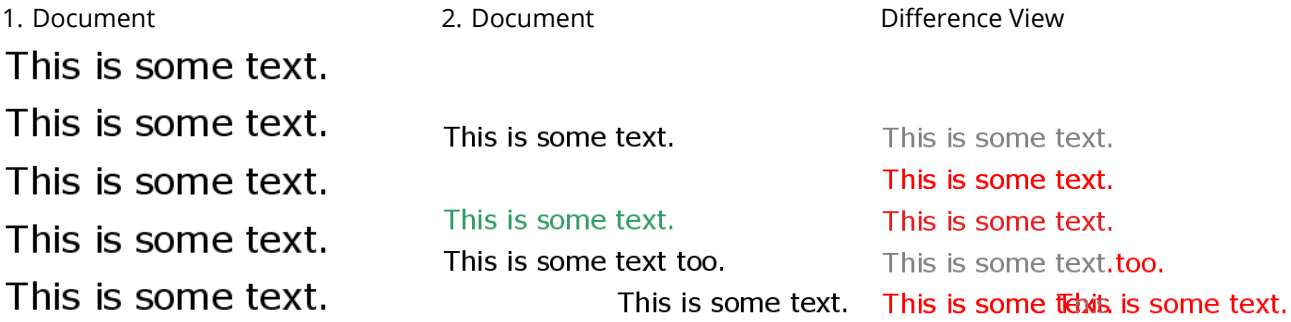
5.11 Difference View

The 3-Heights™ Java Document Viewer 1.8.20.1 and later provides the functionality for a difference view.

The difference view can be used to compare two PDF documents and visualize differences. More concretely it works like this:

- Comparison is done pixel based
- Pixels that are equal in both documents are converted to grayscale
- Pixels that are different are displayed in red color, the larger the difference of the two pixels, the higher the intensity of the red color
- If the two documents have a different amount of pages, the larger amount is taken, pages that are missing in either document are different in every pixel

Here is an example with two documents:



1. Line: No differences, the result is displayed in grayscale
2. Line: Text is missing, difference is displayed in red
3. Line: Text color changed, the difference of the color is represented by the intensity of the red
4. Line: The additional word and the moved period are marked red
5. Line: The location of the text is moved, every pixel that is only existing in either document is marked red, if there are any overlapping text portions, they are marked grayscale.

The two new classes to support the difference view are: *PdfDifferencePage* and *PdfDifferenceView*. They are inherited from *PdfPage* and *PdfView*. This allows for using the difference view equally as the view of a normal (PDF) document. See also chapter “Open and Close a Document”.

Normal Viewer:

```
viewer = new PdfViewer(fileName, password);
```

Difference Viewer:

```
import com.pdftools.pdfviewer.PdfDifferenceViewer;
viewer = new PdfDifferenceViewer(fileName1, pwd1, fileName2, pwd2);
```

5.12 Extract Text

The text extraction interface is used to access the text contained in a page. Each individual text word and its properties such as the text string, position, bounding rectangle and the font will be returned in the order in which they appear in the page's content stream (so called z-order).

```
public void extractTextFromPage(PdfViewerPage page)
{
    PdfTextExtractor extractor = page.getTextExtractor();
    for (;;)
    {
        PdfText text = extractor.getNextText();
        if (text == null)
          break;
        String s = text.getString();
        ...
    }
}
```

In order to get access to the text of the whole document the pages can be enumerated first.

```
public void extractTextFromDocument(PdfViewer viewer)
{
    for (int i = 1; i < viewer.getPageCount(); i++)
    {
        PdfViewerPage page = viewer.getPage(i);
        extractTextFromPage(page);
    }
}
```

The position property of the text can be used extract text in a specific range e.g. a mark rectangle. The coordinates are returned in point units of the document's coordinate system. To get the viewport coordinates use the page's window to viewport transform method.

```
    Point2D pt = text.getPosition();
    if (is_in_range(pt))
    ...
```

5.13 Find Text

All occurrences of a text which matches a given regular expression, e.g. all words ending with "is", can be found using the text finder interface.

```
public void findTextInPage(PdfViewerPage page)
{
    Color clr = new Color(255, 255, 255);
    PdfTextFinder finder = page.getTextFinder(".*?is", clr);
    for (;;)
    {
        PdfText text = finder.getNextText();
        if (text == null)
            break;
        ...
    }
}
```

Finding text in the whole document is made easier using the page finder interface instead of just enumerating the pages and then using the text finder. This has the advantage that only those pages with at least one occurrence are returned.

```
public void findTextInDocument(PdfViewer viewer)
{
    Color clr = new Color(255, 255, 255);
    PdfPageFinder pages = viewer.getPageFinder(".*?is", clr);
    for (;;)
    {
        PdfTextFinder finder = pages.getNextFinder();
        for (;;)
        {
            PdfText text = finder.getNextText();
            if (text == null)
            break;
            ...
        }
    }
}
```

All occurrences of the found text can be high-lighted on the page display using the draw method of the text finder.

```
public void draw(Graphics2D g, Rectangle rect,
PdfViewerPage page, PdfTextFinder finder)
{
    page.draw(g, rect);
    finder.draw(g, rect);
}
```

If one needs to navigate to a specific occurrence of a text word then the position property and some context information can be used to accomplish this.

```
PdfTextFinder finder = page.getTextFinder();
...
PdfText text = finder.getNextText();
...
Point2D pos = text.getPosition();
goTo(finder.getPageNo(), pos.getX(), pos.getY(), zoom);
...
```

6 Licensing and Copyright

The 3-Heights™ Java Document Viewer 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.

7 Contact

PDF Tools AG

Kasernenstrasse 1

8184 Bachenbülach

Switzerland

http://www.pdf-tools.com

A Appendix

A.1 Font Replacement Strategy

1. A Type3 font is expanded inline. Glyphs are cached.
2. If the font is embedded and the eOptionOutline is turned on then the glyphs are converted to graphics paths.
3. If the font is embedded and is installable it will be converted to the windows format and temporarily installed. While printing the font is converted to a TrueType font prior to being installed.
4. If the font is not embedded and it is a standard font then a standard font in the fonts directory (ZapfDingbats) or a pre-installed font is used instead.
5. If the font is not a standard font and it is a pre-installed font then the pre-installed font is used.
6. If the font is not a pre-installed font and it is listed in the fonts.ini file then the replacement font is used.
7. If the font is not listed in the fonts.ini file the nearest pre-installed font is used depending on the font's properties (family, pitch, Unicode subset, weight, italic, etc.).