1. Getting Started

The major difference with the previous PD4ML versions is that the new API performs the conversion in two phases.

First you need to invoke `readHTML()` method to read a source document. By default an HTML (styled with CSS) is expected. However it can be arbitrary XML, where tag nature is specified with CSS **display** property, e.g. `<sideblock style="display: block; float: right">Side content</sideblock>` .

After the document is read and parsed you can render and write it as **PDF**, **PDF/A**, **RTF** or an raster image with `writePDF()` or another corresponding method. If you need a conversion result in different formats – invoke corresponding `write*()` methods multiple times, there is no need to reread the source HTML.

That's all you need to know about PD4ML to write your first converter application...

```
1.  PD4ML pd4ml = new PD4ML();
2.
3.  String html = "TEST<pd4ml:page.break><b>Hello, World!</b>";
4.  ByteArrayInputStream bais =
5.          new ByteArrayInputStream(html.getBytes());
6.
7.  // read and parse HTML
8.  pd4ml.readHTML(bais);
9.
10. File pdf = File.createTempFile("result", ".pdf");
11. FileOutputStream fos = new FileOutputStream(pdf);
12.
13. // render and write the result as PDF
14. pd4ml.writePDF(fos);
15.
16. // alternatively or additionally:
17. // pd4ml.writeRTF(rtfos, false);
18. // BufferedImage[] images = pd4ml.renderAsImages();
19.
20. // open the just-generated PDF with a default PDF viewer
21. Desktop.getDesktop().open(pdf);
```

Suggest edit

Last updated on September 4, 2018

2. Obtaining PD4ML

Since v4 PD4ML is distributed as an universal library, whose features like TTF embedding support or PDF/A output are activated depending on license key. So binaries for all license types can be obtained from a single location.

Each PD4ML v4.x release is published to **our Maven repository**. You may either download the library from the location or, if you use Apache Maven, tune your project build to automatically download the latest library build.

The binaries in the repository are supplied with Javadoc JARs. The most actual Javadoc is always available online.

PD4ML source code licensees also have an access to debugger-friendly library versions with debug info not stripped off. The library versions and their source code JARs are available from **the access-restricted Maven repository**.

Nightly builds are available in both repositories as snapshots.

To use PD4ML in Maven environment you'd need to configure something like that:

1. Access to regular binaries

Project **pom.xml**

```
...
<dependncy>
  <groupId>com.pd4ml</groupId>
  <artifactId>pd4ml</artifactId>
  <version>4.0.0</version>
</dependency>
...
<repository> <!-- move the section to settings.xml, if desired -->
  <id>pd4ml</id>
  <url>https://pd4ml.tech/maven2/</url>
  <snapshots>
    <enabled>true</enabled>
  </snapshots>
</repository>
```

2. Access to the binaries with source code

~/.m2/settings.xml

```
<server>
  <id>pd4ml-src</id>
  <username>login_name_from_license</username>
  <password>password_from_license</password>
</server>
```

Of course, it is recommended to store the password in encrypted form there. See **Maven documentation** how to do that.

Project **pom.xml**

```
...
<dependncy>
  <groupId>com.pd4ml</groupId>
  <artifactId>pd4ml</artifactId>
  <version>4.0.0</version>
</dependency>
...
<repository>
  <id>pd4ml-src</id>
  <url>https://pd4ml.tech/maven2-src/</url>
  <snapshots>
    <enabled>true</enabled>
  </snapshots>
</repository>
```

Suggest edit

3. Running PD4ML Converter as a standalone application

PD4ML is not only a software library/component, but also a set of tools, which can be used as standalone applications.

First download the most recent version of PD4ML JAR file from **our Maven repository**, save it to a directory of your choice and make sure there is a **JRE v1.6+** installed on your workstation/server.

There are two tools available:

- **Pd4Cmd** – command line wrapper of PD4ML API
- **PD4Browser** – GUI viewer/converter

3.1. PD4ML as a command-line converter tool

The command line tool is implemented as `com.pd4ml.tools.Pd4Cmd.class` . The class is specified as a main class of the JAR, so it can be executed a simple way:

```
java -jar pd4ml*.jar
```

where `java*.jar` is an actual JAR file name, like `pd4ml-4.0.1.jar` .

There are four typical usage scenarios:

1. HTML conversion to PDF, RTF or a raster image
2. PDF processing (merging, updating)
3. Reading of document meta information
4. Indexing of TTF fonts

HTML conversion to PDF, RTF or a raster image

HTML-to-PDF conversion with the absolute minimum of parameters

```
java -Djava.awt.headless=true -Xmx512m -jar pd4ml.jar "http://pd4ml.com" 1200
```

- The command line overrides the default Java memory heap size limit with **-Xmx512m**. Here it is set to 512Mb.
- On UNIX platform `-Djava.awt.headless=true` allows to run the application on non-graphics-enabled servers or from remote ssh/telnet sessions.
- `http://pd4ml.com" 1200` are *HTML source URL* and *htmlWidth* (virtual "browser" frame width) parameters.

> Note: on Win32 the URL is enclosed, if needed, to double quotes, on UNIX – to single quotes.

- The default PDF document format: **A4 / PORTRAIT**
- In the example 1200**px** width of rendered document will be mapped to 595**pt** widths of A4 page format.
As long as an output file path omitted, the output is sent to ***STDOUT*** and can be piped to another application.

Customized HTML-to-PDF conversion

```
java -Djava.awt.headless=true -Xmx512m -jar pd4ml.jar "http://pd4ml.com" 1200 LETTER -
bookmarks HEADINGS -pdfforms -debug -out pd4ml.pdf
```

- In the examples the generated PDF is written to a file, defined with **-out** parameter. That makes possible to use ***STDOUT*** for debug output (**-debug** parameter).
- The examples also force PD4ML to produce PDF outlines (bookmarks) from `<h1>-<h6>` hierarchy of the document (`-bookmarks HEADINGS`) and to convert HTML forms to interactive PDF forms (`-pdfforms`).

PDF processing (merging, updating)

PDF page removal

```
java -Djava.awt.headless=true -Xmx512m -jar pd4ml.jar -tools file:c:/docs/test.pdf -pagerange 2-3,5+ -out c:/docs/newdoc.pdf
```

The call extracts a selected range of document pages and saves them as a new document.

PDF documents merge

```
java -Djava.awt.headless=true -Xmx512m -jar pd4ml.jar -tools file:c:/docs/test.pdf -merge file:c:/docs/tomerge.pdf after -out c:/docs/newdoc.pdf
```

Note: `-pagerange` option is not available by a PDF merge

PDF permissions update

```
java -Djava.awt.headless=true -Xmx512m -jar pd4ml.jar -tools file:c:/docs/test.pdf -permissions 28 -out c:/docs/newdoc.pdf
```

`-permissions 28` is a sum of permissions: `AllowDegradedPrint = 4` , `AllowModify = 8` and `AllowCopy = 16` .

[See API reference for more details...]

Reading of document meta information

```
java -Djava.awt.headless=true -Xmx512m -jar pd4ml.jar -tools file:c:/docs/test.pdf -printpermissions -printauthor -printtitle -printpagenum
```

The call prints to **STDOUT** basic PDF info: document permissions (as a hex number), document author, document title, number of document pages (decimal number)

Indexing of TTF fonts

```
java -Xmx512m -jar pd4ml.jar -configure.fonts <fontdir> [pd4fonts.properties location]
```

[See the command line tool documentation...]

Last updated on October 27, 2018

3.2. PD4ML as a GUI application

PD4ML library includes a simple GUI tool for HTML document rendering preview and for test conversion to PDF or to RTF.

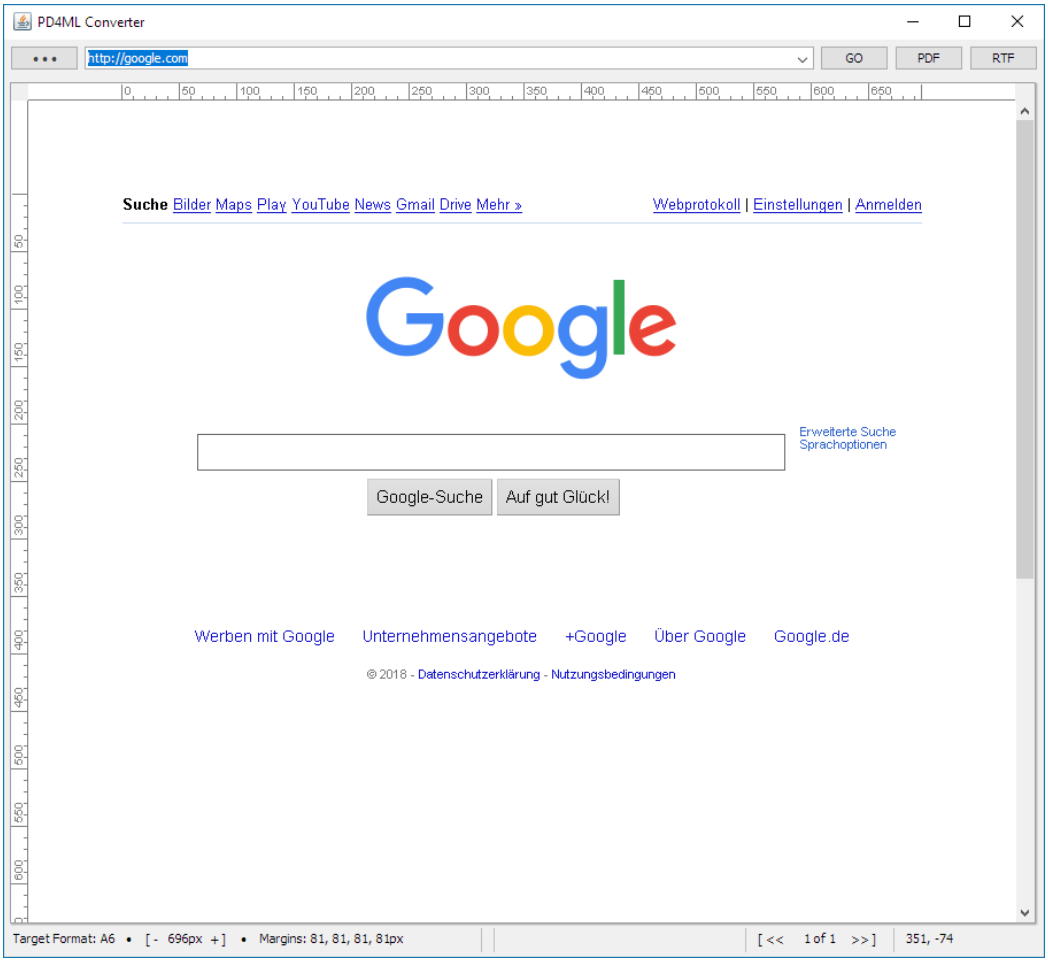
The tool can be activated by a passing `-gui` parameter to the command line:

```
java -jar pd4ml*.jar -gui
```

where `java*.jar` is an actual JAR file name, like `pd4ml-4.0.1.jar` .

There is also a "legacy" way to run the GUI converter, kept for backward compatibility with PD4ML versions prior to v4.0.0

```
java -cp pd4ml*.jar org.zefer.pd4ml.tools.PD4Browser
```



> After the first start the GUI tool creates `pd4browser.properties` file (if no previously created instance found) with the conversion parameter defaults. The file can be customized according your needs in a text editor.

In the top section of the application you can find the controls:

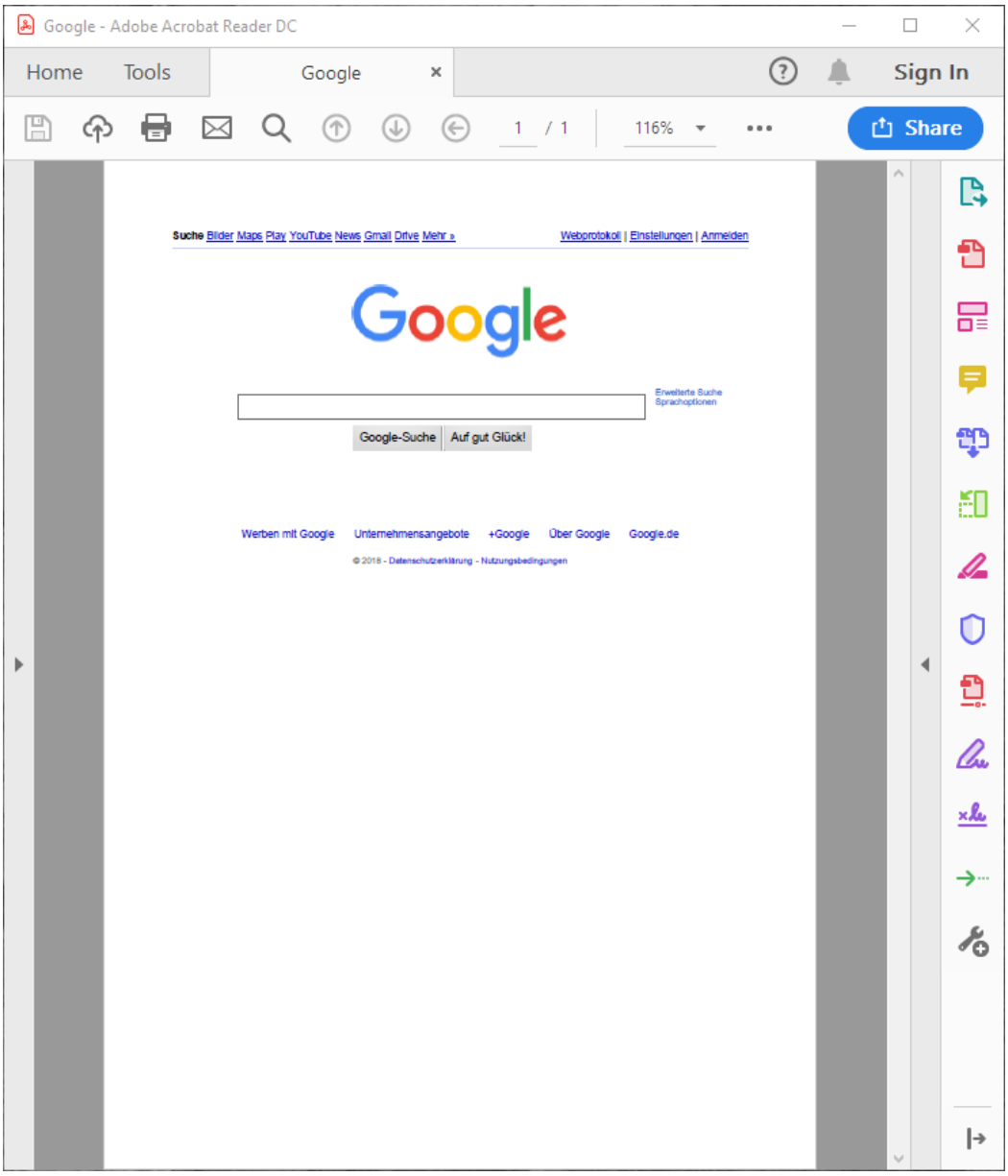
- [•••] – local HTML file open dialog. Accepts `.htm(l)` and `.svg` (and `.dxl` from v4.1.0)
- [GO] – reloads current document and render it in the main application view as an image
- [PDF] – renders current document as PDF and opens it in default PDF viewer application
- [RTF] – renders current document as RTF and opens it in default RTF viewer application

In the status bar there are:

- A label with the target page format. The target file format can be changed in `pd4browser.properties` file or in the source HTML/CSS with `@page` at-rule
- A label with the actual `htmlWidth` conversion parameter value. The value can be changed by a horizontal resizing of the application window. For a fine tuning there are `[+` and `-]` buttons
- Target page margins. The margins can be changed in `pd4browser.properties` file or in the source HTML/CSS with `@page` at-rule.
- Current page number label with page browse buttons `[<<` and `>>]`
- Mouse position / drag box coordinates

> A pointing of the mouse over the status bar labels opens a balloon with additional details.

A clicking `[PDF]` button converts current document to PDF and open the resulting file in a separate PDF viewer window:



The conversion defaults can be adjusted in `pd4browser.properties`. The default properties still include some entries, inherited from older PD4ML versions, and not always usable in v4.0.0.

Here we'll review the important ones.

- `debug.info.enable=0` – logging/debug output verbosity. The value is a bit mask to control logging aspects
- `document.author=` – a string to override document author metadata

- `document.title=` – a string to override document title metadata
- `insets.bottom=10` – bottom page margin value
- `insets.left=10` – left page margin value
- `insets.right=10` – right page margin value
- `insets.top=10` – top page margin value
- `insets.units=mm` – margin value units: `mm` or `pt`
- `page.bookmarks.destinations=false` – enables a generation of PDF page bookmarks (outlines) from named anchors (`<a name="an1"/>`). Not compatible with `page.bookmarks.headings=true`
- `page.bookmarks.headings=false` – enables a generation of PDF page bookmarks (outlines) from `<H1>`-`<H6>` hierachy. Not compatible with `page.bookmarks.destinations=true`
- `page.format=A4` – specified target page format (e.g. `A6` , `LETTER`)
- `page.hyperlinks=true` – enables a conversion of HTML hyperlinks to active PDF hyperlinks
- `page.orientation=false` – by `true` rotates given target page format to 90 degrees (e.g. portrait to landscape)
- `pdf.forms.enable=false` – enabled a conversion of HTML forms to submittable native PDF forms.
- `pdf.pdfa=false` – enables PDF/A output
- `pdf.protect.pud=false` – forces PD4ML to keep HTML object sizes given in physical units (mm, pt, in etc) in resulting PDF, independently on HTML-to-PDF scale factor
- `proxy.host=` – proxy host address (if needed)
- `proxy.port=0` – proxy port (if needed)
- `style.add=` – CSS style string to be applied to loaded documents
- `ttf.fonts.default.monospace=` – default monospaced font. Not recommended, use CSS style for that instead
- `ttf.fonts.default.sansserif=` – default Sans Serif font. Not recommended, use CSS style for that instead
- `ttf.fonts.default.serif=` – default Serif font. Not recommended, use CSS style for that instead
- `ttf.fonts.dir=` – location of TTF fonts dir directory with `pd4fonts.properties` index file
- `userSpace.adjustToContent=false` – if `true` , forces PD4ML to set `htmlWidth` value to the measured width of the HTML content.

Last updated on October 28, 2018

4. PD4ML Java API

API Documentation

Each release of PD4ML comes with actual JavaDocs. When you use Maven for dependency management, your IDE will automatically download the JavaDocs JARs and when you hover on a PD4ML class or method, show you the corresponding documentation.

For reference we also publish the API documentation online so you can link to it from emails or websites:
https://pd4ml.tech/javadoc/com/pd4ml/PD4ML.html

Pre-Requisites

The following are the pre-requisites needed to use the PD4ML Java API library:
* JDK installed on your PC
* Apache Maven build automation tool
* Eclipse installed on your PC (While Eclipse is not compulsory it is highly recommended)

* PD4ML Java API can be downloaded from our website directly (or let Maven do that automatically)
* PD4ML evaluation license

Developing your first application

Creating blank Maven Java project

As a good starting point we recommend to create a blank Java project as described in the tutorial:

https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html

After you've got the example application to run, let's add to it PD4ML.

First add PD4ML library dependency to project's `pom.xml` . See lines 17-22 below:

```
1.  <project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2.    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-
    v4_0_0.xsd">
3.    <modelVersion>4.0.0</modelVersion>
4.    <groupId>com.mycompany.app</groupId>
5.    <artifactId>test-pd4ml</artifactId>
6.    <packaging>jar</packaging>
7.    <version>1.0-SNAPSHOT</version>
8.    <name>test-pd4ml</name>
9.    <url>http://maven.apache.org</url>
10.   <dependencies>
11.     <dependency>
12.       <groupId>junit</groupId>
13.       <artifactId>junit</artifactId>
14.       <version>3.8.1</version>
15.       <scope>test</scope>
16.     </dependency>
17.     <dependency>
18.       <groupId>com.pd4ml</groupId>
19.       <artifactId>pd4ml</artifactId>
20.       <version>[1.0.0,)</version>
21.     </dependency>
22.   </dependencies>
23.   <build>
24.     <plugins>
25.       <plugin>
26.         <groupId>org.apache.maven.plugins</groupId>
27.         <artifactId>maven-compiler-plugin</artifactId>
28.         <version>3.1</version>
29.         <configuration>
30.           <!-- or whatever version you use -->
31.           <source>1.7</source>
32.           <target>1.7</target>
33.         </configuration>
34.       </plugin>
35.     </plugins>
36.   </build>
37.   <repositories>
38.     <repository> <!-- move the section to settings.xml, if desired -->
39.       <id>pd4ml</id>
40.       <url>https://pd4ml.tech/maven2/</url>
41.       <snapshots>
42.         <enabled>true</enabled>
43.       </snapshots>
44.     </repository>
45.   </repositories>
46. </project>
```

In order to resolve the dependency, add a location of PD4ML repository. The right location for the repository address is

`$HOME/.m2/settings.xml` (See Maven documentation). We'll keep the example configuration simpler and add the repository info to `pom.xml`, despite it is not a recommended practice. See lines 37-45 in the example above.

> If you own PD4ML license with product source code access, use `https://pd4ml.tech/maven2-src/` as the repository URL. You would also need to configure the repository access credentials (login/password pair) obtained from the license email.

Also for runtime environment simplicity we'll add the example PD4ML API calls to the application test suite. A typical test case class looks like the example:

```
1.  public class AppTest extends TestCase {
2.    public AppTest(String testName) {
3.      super(testName);
4.    }
5.
6.    public static Test suite() {
7.      return new TestSuite(AppTest.class);
8.    }
9.
10.   public void testApp() {
11.   }
12. }
```

We are going to extend `testApp()` method with PD4ML usage example code.

Creating a PD4ML object

When writing an application with PD4ML API, the very first thing to do is to create the PD4ML object:

```
1.  try {
2.    PD4ML pd4ml = new PD4ML();
3.  } catch (Exception e) {
4.    e.printStackTrace();
5.  }
```

The constructor in the example has no parameters and it expects to find license `pd4ml.lic` file in the working dir of the application or in the root folder of the classpath or of `pd4ml*.jar`.

Alternatively you may pass to PD4ML constructor a string with a serial number (obtained from the license email) or a string with an URL of an alternative `pd4ml.lic` location.

Where to obtain `pd4ml.lic`? Download it from your license page or save the serial number from the license email to a text file and name the file `pd4ml.lic`

Reading HTML

After you've got an instance of PD4ML you can read and parse a source HTML document with one of a variety of `readHTML()` methods.

In the example it reads from a prepared input stream. But it can also read from an arbitrary URL.

```
1.  String html = "<b>Hello, <i>World!</i></b>";
2.  ByteArrayInputStream bais =
3.  new ByteArrayInputStream(html.getBytes());
4.
5.  // read and parse HTML
6.  pd4ml.readHTML(bais);
```

If you want to point the converter to a font directory or to apply an additional style, headers, footers, watermarks etc – the

corresponding API calls need to be performed before `readHTML()`

Writing target format

A parsed source document can be rendered as a sequence of images, PDF or RTF documents.

Here it writes a PDF:

```
File pdf = File.createTempFile("result", ".pdf");
FileOutputStream fos = new FileOutputStream(pdf);

// render and write the result as PDF
pd4ml.writePDF(fos);
```

Launching default document viewer (optional)

It is rarely needed in real-life applications, but very handy in our case: lookup for a default viewer and launch it for just generated PDF.

```
// open the just-generated PDF with a default PDF viewer
Desktop.getDesktop().open(pdf);
```

It looks for a viewer, associated with given file extension. For PDF it launches `Adobe Reader` (or similar). If we write to a file with .rtf extension it likely launches `MS Word` or `WordPad`

First PD4ML application complete

The application, in its entirety, is as follows:

```
package com.mycompany.app;

import java.awt.Desktop;
import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.FileOutputStream;

import com.pd4ml.PD4ML;

import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;

public class AppTest extends TestCase {
  public AppTest(String testName) {
    super(testName);
  }

  public static Test suite() {
    return new TestSuite(AppTest.class);
  }

  public void testApp() {
    try {
      PD4ML pd4ml = new PD4ML();

      String html = "<b>Hello, <i>World!</i></b>";
      ByteArrayInputStream bais =
        new ByteArrayInputStream(html.getBytes());

      // read and parse HTML
      pd4ml.readHTML(bais);

      File pdf = File.createTempFile("result", ".pdf");
```

```
        FileOutputStream fos = new FileOutputStream(pdf);

        // render and write the result as PDF
        pd4ml.writePDF(fos);

        // alternatively or additionally:
        // pd4ml.writeRTF(rtfos, false);
        // BufferedImage[] images = pd4ml.renderAsImages();

        // open the just-generated PDF with a default PDF viewer
        Desktop.getDesktop().open(pdf);

      } catch (Exception e) {
        e.printStackTrace();
      }
    }
  }
```

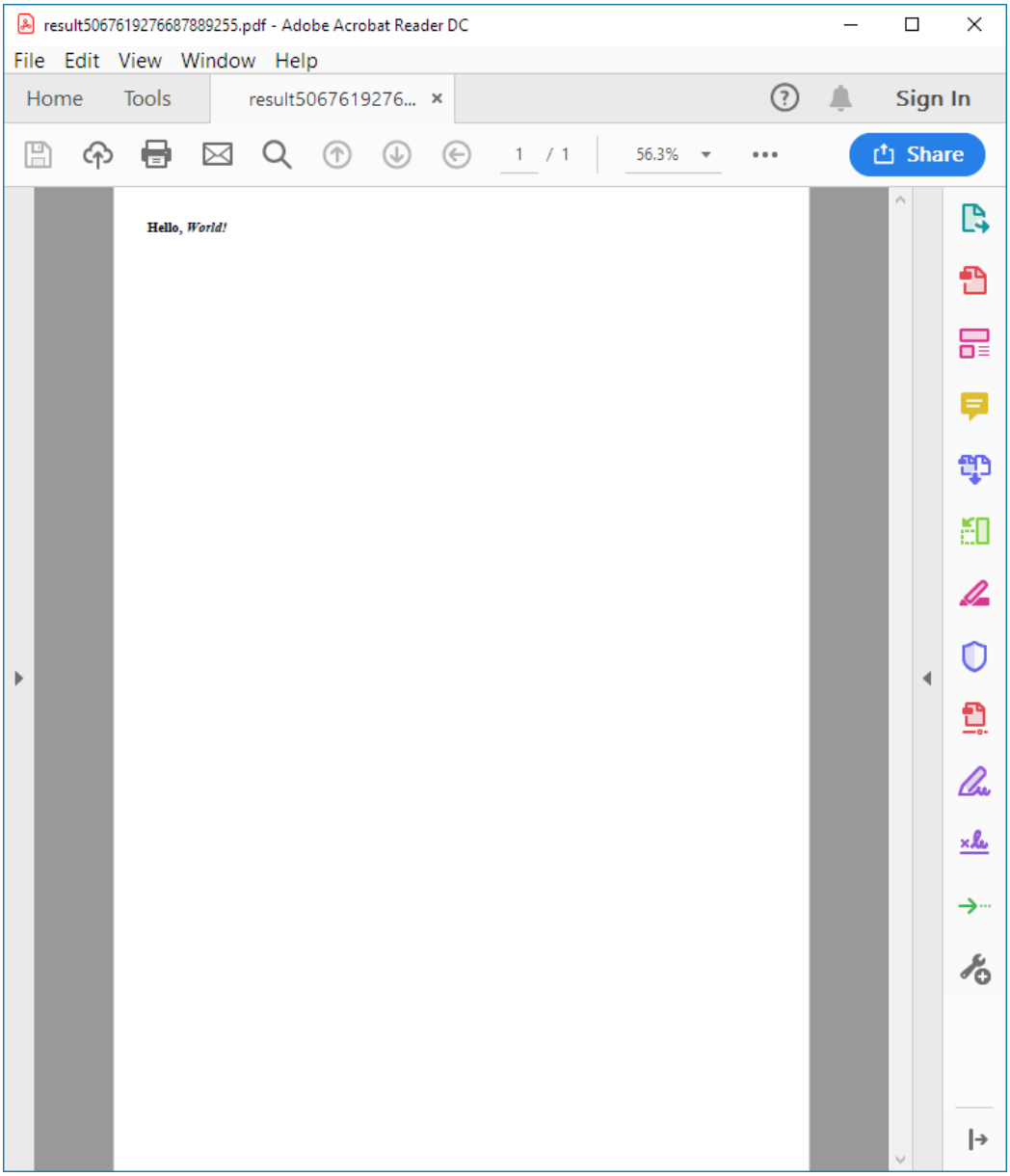
Running the application

Now we are ready to run the demo application.

Go to the Maven project root (where `pom.xml` is) and execute the command line:

```
mvn test
```

The `test` target builds the projects and runs its test suite. As our example code is implemented as a test case, it will be executed. If you did everything correct you'll see a PDF viewer window with a generated PDF file in it:



The document is minimalistic and built using PD4ML defaults: **A4** page format, **10mm** margins, **727px** htmlWidth, no TTF fonts.

> **727px** are calculated from `(A4.width - margins.left - margins.right) * PIXELS_PER_POINT`, which is `(595pt - (10mm + 10mm) * 2.835pt/mm) * 1.35px/pt`

Customizing the conversion

PDF page format and content scaling

There are 3 important PD4ML properties, impact resulting PDF page format and layout: `pageSize`, `pageMargins`, `htmlWidth`.

`pageSize` defines output paper format: `A4`, `ISOB4`, `LETTER`, `LEGAL` etc.

Corresponding constants are defined in `java.pd4ml.Constants` class, but you may define any non-standard page dimension with, for instance, `new com.pd4ml.PageSize(400,400)` (by default the dimensions are in typographic points) and pass it to `pd4ml.setPageSize()` method.

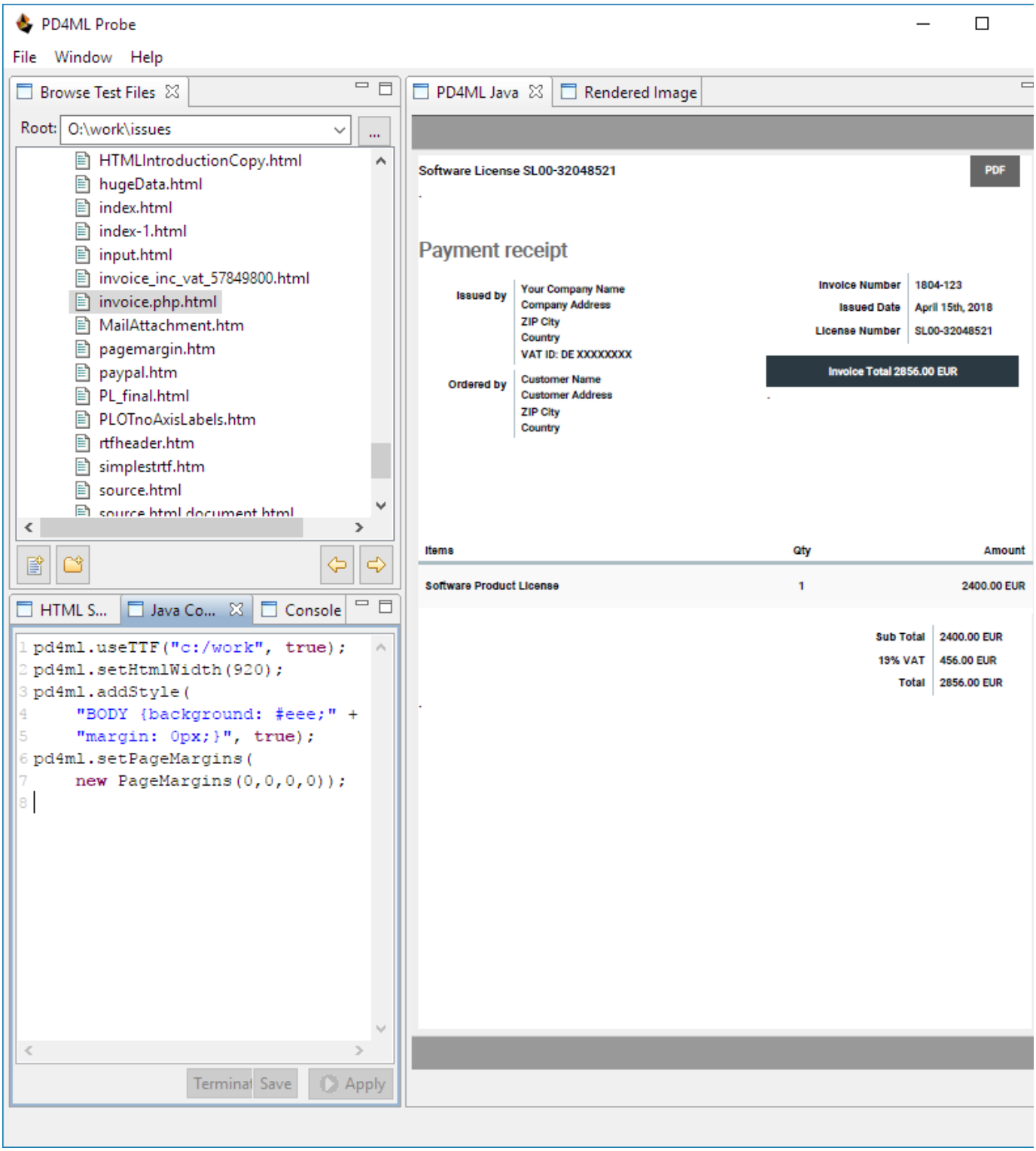
Default page format is `A4`, orientation `portrait`.

PDF file format does not explicitly specifies on meta level if a particular page is **landscape**– or **portrait**-oriented. If a page has its width greater than height – it is landscape, otherwise it is portrait. `com.pd4ml.PageSize` implements a page rotation by a simple swapping of page height and width.

`pageMargins` define blank page area width around the page content. The default value for it `new PageMargins(10, 10, 10, 10, Units.MM)`

The page margins can be additionally increased by HTML document margins (e.g. `<body style="margin: 50px">`). The HTML document margins have some specifics, when converted to PDF: the top margin is applied to the top of the document (in other words, on the first page only), the bottom margin on the last page only.

Document with no margins:



`htmlWidth` value defines "virtual web browser" frame width (in screen pixels), and by default for `A4` it is **727px** (to keep 1:1 scale factor by 72dpi).

PD4ML renders source HTML page using `htmlWidth` parameter and maps the resulting layout to the efficient PDF page width (which is `pageFormat.width - pageMargins.left - pageMargins.right`).

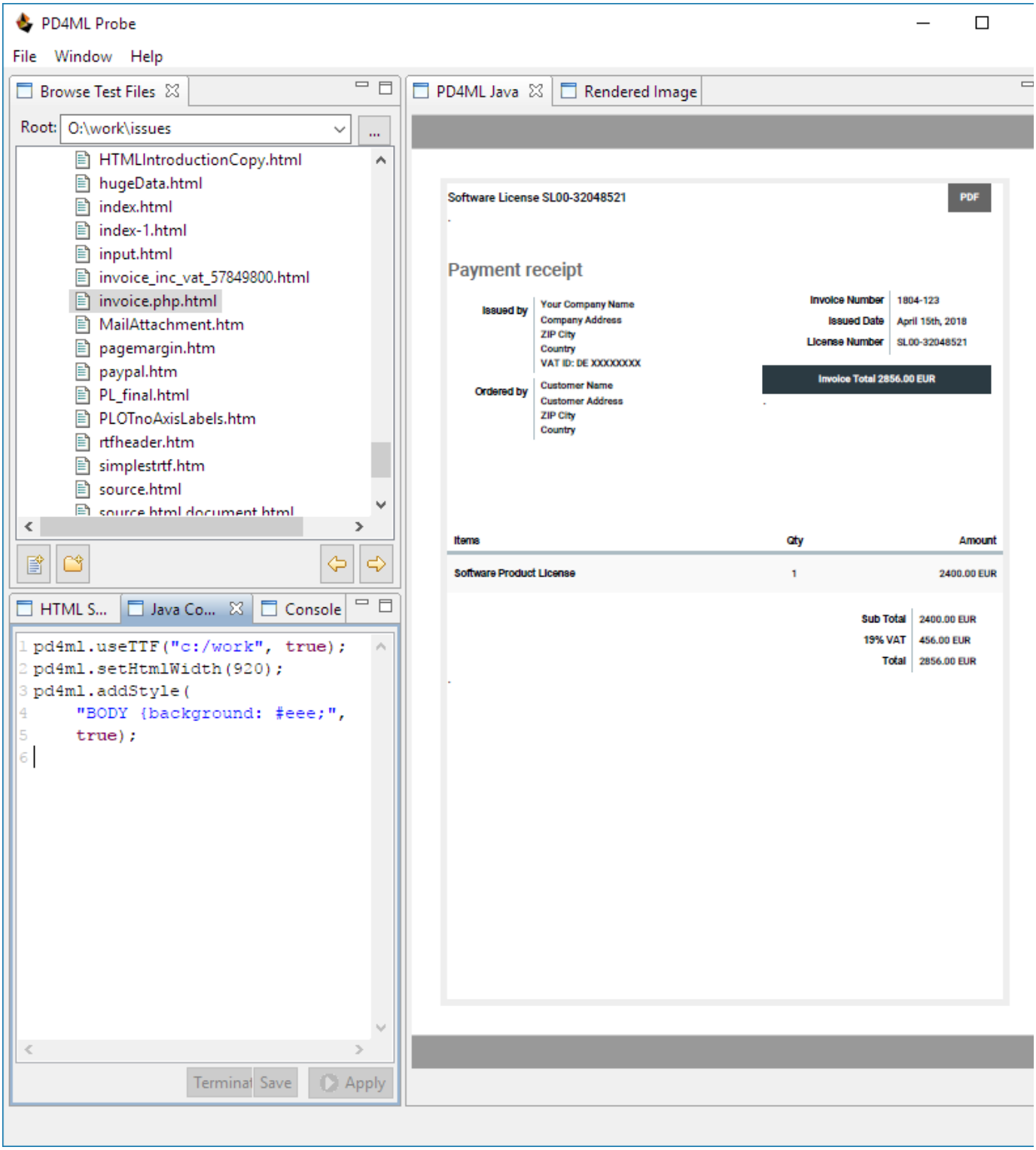
That makes HTML-to-PDF scale factor computed like that:

```
scale = (pageFormat.width - pageInsets.left - pageInsets.right) / htmlWidth
```

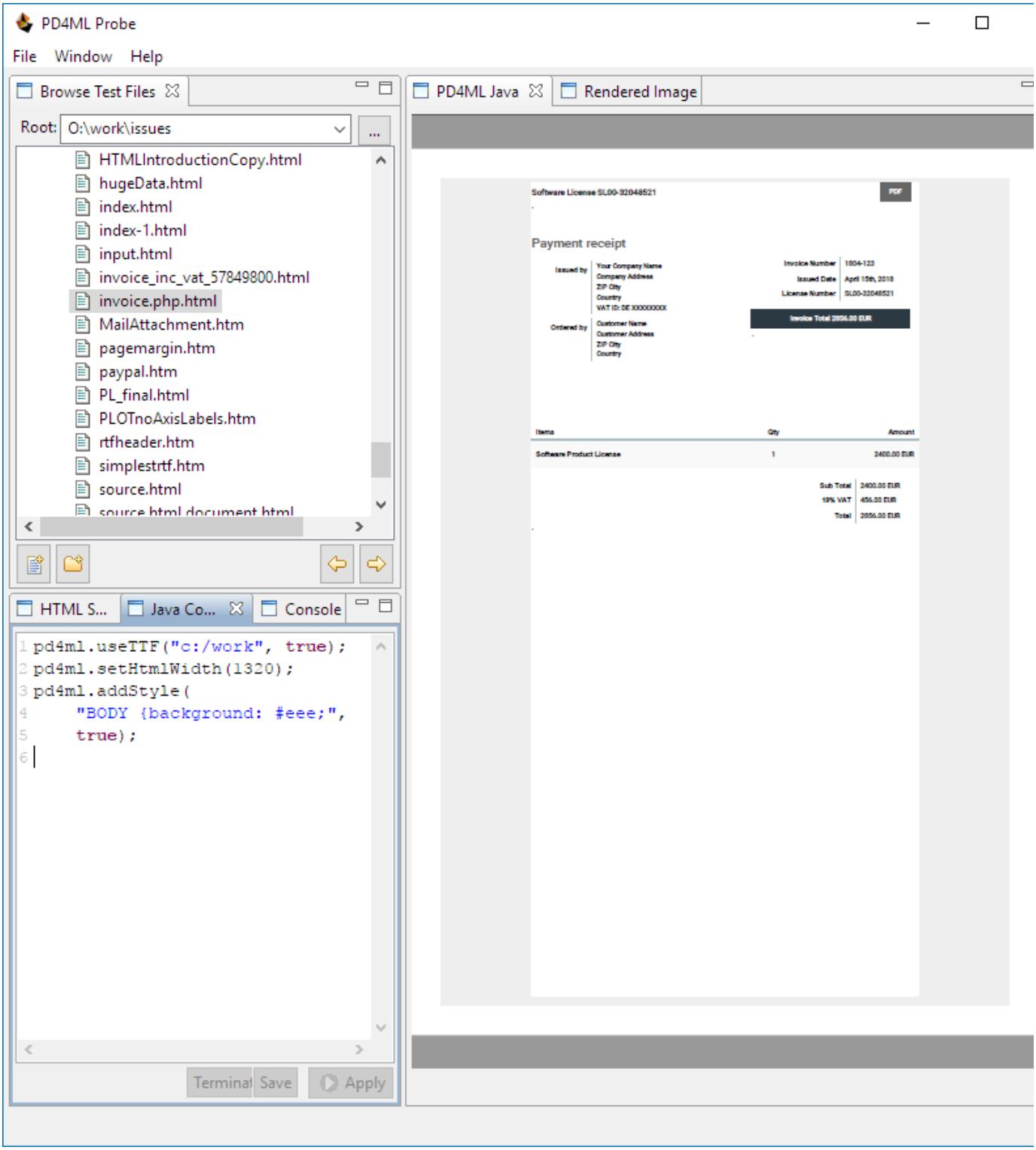
From the above it is obvious, that an increasing of `htmlWidth` makes the resulting document content appears smaller.

Examples:

htmlWidth = 920



htmlWidth = 1320



Defining page header and footer

`setPageHeader()` and `setPageFooter()` methods are used to define page headers/footers using HTML as a layout definition.

Optional **scope** parameter allows to apply particular header or footer to a specified range of pages.

The HTML code may include the placeholders: **$[page]** – to be substituted with current page number; **$[total]** – total number of pages; **$[title]** – document title as defined in **<title>** HTML tag or overridden with `setDocumentTitle()` API call.

```
1.  // define page header for the first page 30px high
2.  pd4ml.setPageHeader("$[title]", 30, "1");
3.
```

```
// define page footer for the first page
pd4ml.setPageFooter("Total pages: $[total]", 30, "1");

// define page header for the second page
pd4ml.setPageHeader("<b>$[title]</b> $[page]/$[total]", 30, "2+");

// define page footer for the second page
pd4ml.setPageFooter("<div style='width: 100%; text-align: right'>Page: $[page]</div>",
30, "2+");
```

E003SetPageHeaderFooter.java

Protecting resulting document

`setPermissions()` method allows to apply the standard PDF security options: define a document password or restrict particular document actions (like a hi-res print).

See a list of applicable permission flags (`Allow*` and `DefaultPermissions`).

It is possible to define a positive list of permissions:

```
pd4ml.setPermissions(null, Constants.AllowAnnotate | Constants.AllowDegradedPrint);
```

or to disable only selected ones:

```
pd4ml.setPermissions(null, Constants.DefaultPermissions ^ Constants.AllowModify);
```

```
// protect the document with "test" password. No permission restrictions applied
pd4ml.setPermissions("test", Constants.DefaultPermissions);
```

E009SetDocumentPassword.java

Reading resources from non-standard sources

If some HTML resources like images or stylesheets are not accessible with the standard methods (file read, HTTP(S), etc), you may define your own resource reading "driver".

First, define a resource addressing syntax, that matches your needs. For example `<a src="database:table=pictures;id=4711">`

Second, implement a resource loader, which knows what to do with **"database:table=pictures;id=4711"** URI.

The loader has to be derived from **com.pd4ml.ResourceProvider** class and to implement two methods: `boolean canLoad(String resource, FileCache cache)` to test if it can read the URL; `BufferedInputStream getResourceAsStream(String resource, FileCache cache)` to actually read the resource bytes.

```
public class DummyProvider extends ResourceProvider {

  public final static String PROTOCOL = "dummy";

  @Override
  public BufferedInputStream getResourceAsStream(String resource, FileCache cache)
throws IOException {
    if (!resource.toLowerCase().startsWith(PROTOCOL)) {
      return null;
    }

    // interpret the "resource" parameter according to your protocol (e.g. as a key to
a database record etc)
```

```
12.
13.       // in the example we simply dump the resource parameter string
14.       String buf = "[" + resource.substring(PROTOCOL.length()+1) + "]";
15.       ByteArrayInputStream baos = new ByteArrayInputStream(buf.getBytes());
16.       return new BufferedInputStream(baos);
17.     }
18.
19.     @Override
20.     public boolean canLoad(String resource, FileCache cache) {
21.       if (resource.toLowerCase().startsWith(PROTOCOL)) {
22.         return true;
23.       }
24.       return false;
25.     }
26.   }
27.
28.   pd4ml.addCustomResourceProvider("advanced.DummyProvider");
```

A005AddCustomResourceLoader.java

Suggest edit

Last updated on November 10, 2018

5. PD4ML JSP taglib and Web applications

Using PD4ML custom tags in JSP

PD4ML distribution among other features also includes JSP custom tag library. PD4ML JSP custom tags provide a simple mechanism for a converting of HTML or JSP page output into PDF in Web scenarios.

PD4ML JSP taglib deployment

PD4ML tag library classes and the tag library descriptor (TLD) are packaged together in PD4ML JAR file. To make the classes and the TLD accessible for your Web application, you only need to copy `pd4ml*.jar` library to the lib directory `WEB-INF/lib`

Using <pd4tl:transform> tag

Once PD4ML custom tags are deployed, its actions can be called from your HTML using an XML syntax.

First, reference a JAR file containing a tag library from your JSP: `<%@ taglib uri="http://pd4ml.com/tlds/4.0" prefix="pd4tl" %>`

The `uri` attribute is a unique TLD identifier. It does not point to an existing Web document, but refers to `pd4tl.tld` file, packaged to `pd4ml*.jar` and registered there under the same `uri` identifier.

> In PD4ML v4.0.0 we changed the taglib structure and removed some wrapper classes. Now we recommend to use `pd4tl:` tag prefix instead of previous `pd4ml:` in order to distinguish the custom JSP tags from PD4ML proprietary tags (e.g. `<pd4ml:page.break>`).

Next, you need to surround your HTML or JSP content with `<pd4tl:transform>` and `</pd4tl:transform>` tags.

That's it:

```
1.  <%@ taglib uri="http://pd4ml.com/tlds/4.0"
2.    prefix="pd4tl"%><%@page
3.    contentType="text/html; charset=ISO8859_1"%><pd4tl:transform
```

```
    screenWidth="400"
    pageFormat="A5"
    pageOrientation="landscape"
    pageInsets="100,100,100,100,points">
<html>
<head>
<title>pd4ml test</title>
<style type="text/css">
body {
  color: red;
  font-family: Tahoma, "Sans-Serif";
  font-size: 10pt;
}
</style>
</head>
<body>
  <p>
    Hello, World!
  </p>
  <pd4ml:page.break />
  <table style="border: 1px solid gray; border-radius: 5px; background-color: #f8f8f8; color: #000000">
    <tr>
      <td>Hello, New Page!</td>
    </tr>
  </table>
</body>
</html>
</pd4tl:transform>
```

Comments:

Line 1. Make sure there is no whitespaces before the directive start (before `<%@`). The taglib is intended to return a PDF (binary) file. Any content or whitespace (incl. new line characters) before `<pd4tl:transform>` implicitly switches HTTP output to text mode and prefixes the PDF bytes with unexpected characters. As the last resort PD4ML taglib tries to reset such undesired content (if any), but it is not possible in all environments and `java.lang.IllegalStateException: getOutputStream() has already been called for this response` is thrown.

Other symptoms of the undesired content are either corrupted PDF, or PDF, which is always being reindexed when opened by a PDF Viewer.

Line 2 and 3 No characters or whitespaces between tags or directives allowed (See `%><%` and `%><pd4tl:transform`). A reason is as in the comment above.

Line 8 From this point there is no whitespace restriction – you may use an arbitrary HTML or JSP code.

Line 23 The proprietary tag forces PD4ML converter to insert a page break to the output PDF.

Line 31 The closing transform tag. Any content follows the tag is ignored.

Click here to see PD4ML taglib documentation

Prompt to save just-generated PDF

By default if you open a PD4ML-enabled JSP with a Web browser, it opens a generated PDF inline (within the browser frame). If you'd like the browser to propose to save the PDF under particular name or open it with the system PDF viewer, add two transform attributes `inline` and `fileName`

```
<%@ taglib uri="http://pd4ml.com/tlds/4.0"
  prefix="pd4tl"%><%@page
  contentType="text/html; charset=ISO8859_1"%><pd4tl:transform
  screenWidth="800"
  pageFormat="A4"
  pageOrientation="landscape"
  inline="false"
```

```
    fileName="file.pdf"
    pageInsets="10,10,10,10,mm">
<html>
<head>
<title>pd4ml test</title>
</head>
<body>
  Hello, World!
</body>
</html>
</pd4tl:transform>
```

Defining PDF document footer (or header) with JSP custom tag.

Since PD4ML v4.0.0 a dedicated header/footer JSP custom tags are not supported anymore. Instead of it use PD4ML proprietary `<pd4ml:page.header>` and `<pd4ml:page.footer>` tags (they take effect not only in JSP context, but also in regular HTML).

Identically you may define a watermark with `<pd4ml:watermark>` tag.

```
<%@ taglib uri="http://pd4ml.com/tlds/4.0"
  prefix="pd4tl"%><%@page
  contentType="text/html; charset=ISO8859_1"%><pd4tl:transform
  screenWidth="800"
  pageFormat="A4"
  pageOrientation="landscape"
  pageInsets="10,10,10,10,mm">
<html>
<head>
<title>Page title.</title>
</head>
<body>
<pd4ml:page.header height=30 scope="1">$[title] Visible only on the first
page</pd4ml:page.header>
<pd4ml:page.footer height=30>Page $[page] of $[total]</pd4ml:page.footer>
Page 1
<pd4ml:page.break>
Page 2
</body>
</html>
</pd4tl:transform>
```

Comments:
Line 13 Page header definition, whose scope is limited only by the first page. `$[title]` placeholder is to be substituted with a title value taken from HTML's `<title>` > tag.

Line 14. Page footer definition to be applied to all pages (as there is no `scope` attribute). Placeholder `$[page]` and `$[total]` to be substituted with the current page number and total number of pages correspondingly.

How to add dynamic data (like current date) to PDF header or footer

It is JSP. Forget about PDF and do it you would normally do it in a regular JSP:

```
<%
String template = getFormattedDate() ;
%>
<pd4ml:page.footer height=30><span style="color: tomato"><%=template%></span>, page
$[page]</pd4ml:page.footer>
```

Temporary saving generated PDF to hard drive.

With `<pd4tl:savefile>` tag you have a possibility to store just generated PDF to a hard drive (on server side) and redirect user's browser to read the PDF as a static document or to redirect the request to another URL for PDF post-processing.

Note: the tag should be nested to `<pd4ml:transform>` and have no body.

Usage 1.

```
<pd4ml:savefile
    uri="/WEB/savefile/saved/"
    dir="D:/spool/generated_pdfs"
    redirect="pdf"
    debug="false"/>
```

The tag above forces PD4ML to save the generated PDF to `D:/spool/generated_pdfs` with an autogenerated name.

It is expected, that local directory `D:/spool/generated_pdfs` corresponds to URL `http://yourserver.com/WEB/savefile/saved/` (as given in `uri` attribute)

After generation PD4ML will send to client's browser a redirect command with URL like that:

```
http://yourserver.com/WEB/savefile/saved/generated_name.pdf
```

Usage 2.

```
<pd4ml:savefile
    dir="D:/spool/generated_pdfs"
    redirect="/mywebapp/send_pdf_by_email.jsp"
    debug="false"/>
```

The tag above forces PD4ML to save the generated PDF to `D:/spool/generated_pdfs` with an autogenerated name.

After that it forwards to `/mywebapp/send_pdf_by_email.jsp` with a parameter `filename=<pdfname>` .

So `send_pdf_by_email.jsp` can read file name with

```
String fileName = request.getParameter("filename");
```

build the full path

```
String path = "D:/spool/generated_pdfs" + "/" + fileName;
```

read the just-generated PDF file and perform post-processing or other actions (like a sending of the PDF by email).

In both cases above you can predefine PDF file name with `name` attribute. If a file with the name is already exists in `D:/spool/generated_pdfs` , then the new file name is appended with an auto-incremented numeric value.

Using PD4ML custom tags with Struts or with any other J2EE UI frameworks, if JSP taglib integration is problematic

The specifics of many UI frameworks (like **Struts**) is that **in some cases** they take control and open output stream in text mode before PD4ML-enabled JSP page is loaded. On the other hand PD4ML needs an exclusive control over the output stream: it outputs binary data and any other output writers can corrupt it.

In order to solve the problem there are workarounds.

As described in the above chapter, a just generated PDF can be temporally stored to the hard drive and the current HTTP request can be forwarded to the new static PDF.

Alternatively PD4ML transformer JSP pages can be deployed to a separate web application outside of the framework (e.g. Struts) context. It can be even the same web application, but the PD4ML-enabled JSP pages should be out of control of the Struts dispatching servlet. (Can be achieved by `web.xml` settings).

For our example the separate Web application is associated with name `separate_web_app` .

Create a dedicated PD4ML transformer JSP page `transformer.jsp` in `separate_web_app` like the following.

```
1.  <%@ taglib uri="http://pd4ml.com/tlds/4.0"
2.    prefix="pd4tl"%><%@page
3.    contentType="text/html; charset=ISO8859_1"%><pd4tl:transform
4.      screenWidth="400"
5.      pageFormat="A5"
6.      pageOrientation="landscape"
7.      pageInsets="15,15,15,15,points"
8.      enableImageSplit="false"
9.      inline="true"
10.     fileName="myreport.pdf"
11.     url="http://mainserver/struts/report.jsp">
12.   <pd4ml:footer
13.     titleTemplate="$[title]"
14.     pageNumberTemplate="page $[page] of $[total]"
15.     titleAlignment="left"
16.     pageNumberAlignment="right"
17.     fontSize="12"
18.     areaHeight="14"/>
19.   </pd4ml:transform>
```

Replace `url` attribute value of `<pd4ml:transform>` with the actual URL of your Struts (JSP) page, that should be converted to PDF.

Because of `url` attribute presence (line 11), the HTML content of `transformer.jsp` is ignored completely. But as usually: it should be no any whitespace character before `<pd4ml:transform>` tag.

At this point you can access the `transformer.jsp` and force PDF conversion of the specified source. However in most of the cases a session ID (`JSESSIONID`) propagation is important for proper Struts (or other JSP-based frameworks) functionality.

If the `url` attribute of `<pd4ml:transform>` is set, then PD4ML takes its value and appends to it `JSESSIONID` parameter. The value for the parameter it takes from HTTP request variable `pd4session`

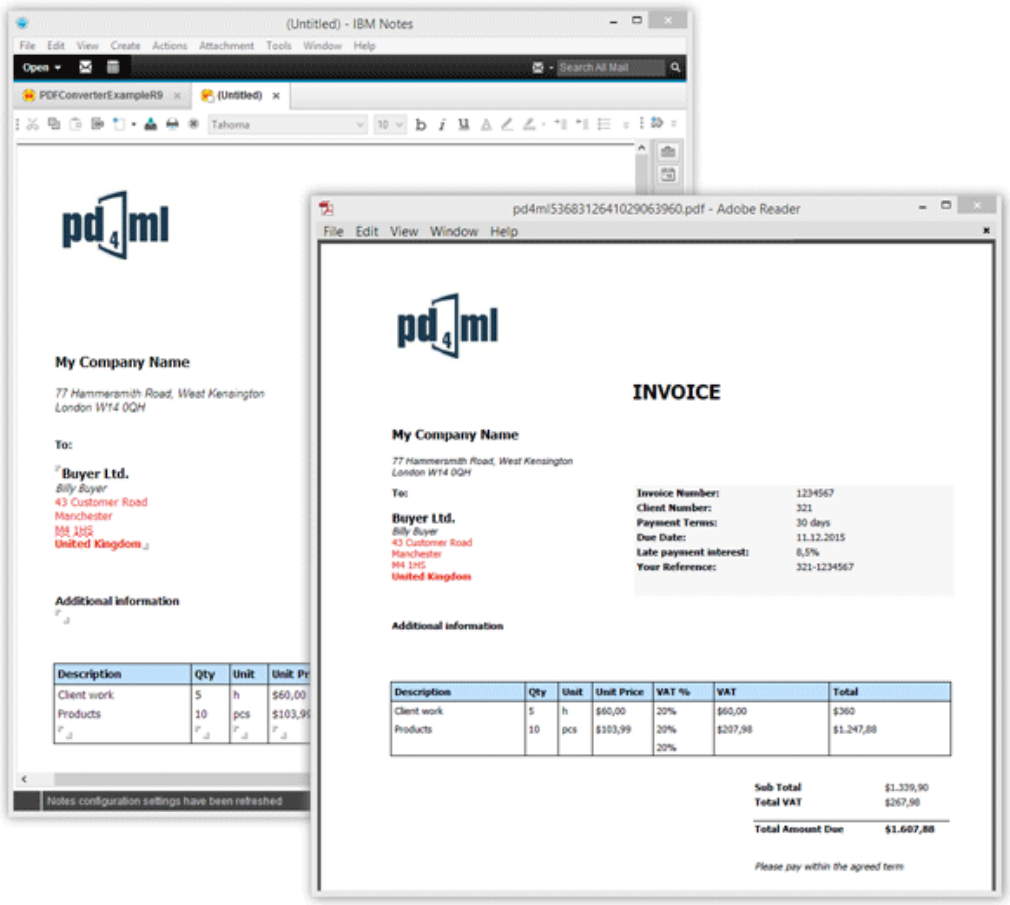
A request for PDF generation from Struts context can look like that:

```
1.  <%
2.  response.sendRedirect("/separate_web_app/transformer.jsp?pd4session=" +
3.  session.getId() );
4.  %>
```


Last updated on November 4, 2018

6. PD4ML and IBM Notes/Domino

v3.x.x v4.1.0



PD4ML can be used to convert IBM Notes documents to PDF (as well as to RTF or to a raster image) a variety of ways.

The most straightforward method is to capture HTML documents, returned by Notes Domino via HTTP: an URL like one of the following is to be passed to `render()` method of PD4ML.

`http://Host/Database/PageUNID?OpenPage`
(i.e. http://www.acme.com/discussion.nsf/35AE8FBFA573336A852563D100741784?OpenPage)

`http://Host/Database/View/DocumentUniversalID?OpenDocument`
(http://www.acme.com/leads.nsf/By+Rep/35AE8FBFA573336A852563D100741784?OpenDocument)

`http://Host/Database/FormUniversalID?ReadForm`
(http://www.acme.com/products.nsf/625E6111C597A11B852563DD00724CC2?ReadForm)

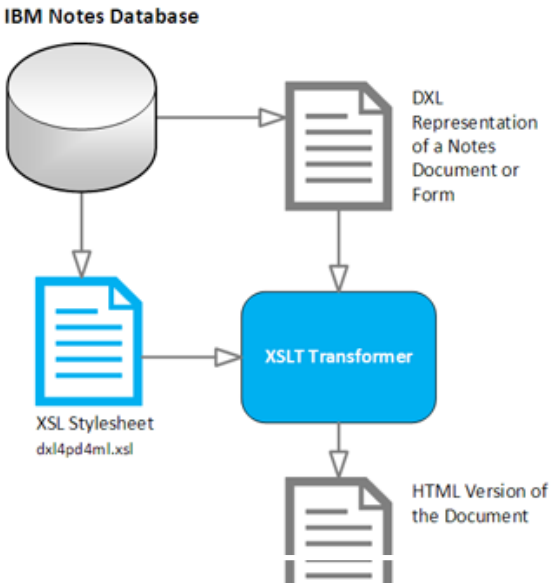
More IBM Notes URL syntax info...

If you run JSP infrastructure on IBM Domino server, another online method of PDF generation would be to use PD4ML JSP custom tag library **(TODO)**.

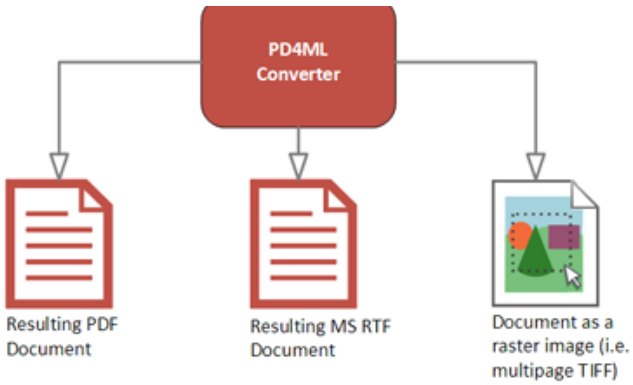
If an online document generation method (involving HTTP, JSP etc) is undesired, there is a possibility to generate PDF documents from their XML representations (so called DXL).

DXL is an adaptation of XML used to describe, in detail, the structure and data contained in a Domino database. It describes data and design elements such as views, forms, and documents and provides a basis for importing or exporting XML representations of data to and from a Domino/Notes application.

Schematically a process of Notes document conversion to PDF can be seen like that:



pd[4]ml Products Licenses Support Try Free Buy Now [ ]



All conversion steps can be implemented at once as a IBM Lotus Notes Java agent, or can be implemented as two-step batch: DXL export request followed by a conversion with Pd4Cmd command-line tool

```
java -Djava.awt.headless=true -Xmx512m -jar pd4ml.jar test.dxl 1200 -xsl notesdefault
```

The command line overrides the default Java memory heap size limit with `-Xmx512m` . Here it is set to 512Mb.

On UNIX platform `-Djava.awt.headless=true` allows to run the application on non-graphics-enabled servers or from remote ssh/telnet sessions.

`test.dxl 1200` are **DXL location** and **htmlWidth** (virtual "browser" frame width) parameters.

> On Win32 the path is enclosed, if needed, to double quotes, on UNIX – to single quotes.

`-xsl notesdefault` applies XSL stylesheet to the input document. In the case it refers to the built-in default DXL-to-HTML XSL, but it can be an URI of an arbitrary external stylesheet.

The default PDF document format: **A4 / PORTRAIT**

In the example 1200**px** width of rendered document will be mapped to 595**pt** widths of A4 page format.

As long as an output file path omitted, the output is sent to **STDOUT**and can be piped to another application.

See more command line arguments...

PD4ML as Notes Agent...

Suggest edit

Last updated on November 4, 2018

7. Usage Examples

Go to Usage Examples section...

Suggest edit

Last updated on September 4, 2018