PDFspy
Version 2.0

Everything you wanted to know about your PDF…

1. Introduction

PDFspy is the ultimate "get info" utility for your PDF documents. It can extract a comprehensive list of attributes from a PDF file into an XML-based format. This list can incorporate as little or as much information as you need - whether it's a list of used font, security settings or color choices.

Some example uses for PDFspy might be:

- Asset management system: extract page count, metadata, font & image information
- Document management: determine text or image only documents, extract comments
- Preflight: extract information about colorspaces, compression & font types
- Validation: find broken PDF files
- Developers: easily examine the structure of complex PDF documents

2. Installing PDFspy

Installation

Windows

PDFspy for Windows is delivered with an Installer. Simply double-click on the Installer application and follow the instructions. This manual covers the features of the PDFspy command line program (CLI) that is in the archive.

Mac OS X

PDFspy for Mac OS X is delivered as a ZIP archive containing the PDFspy command line interface (CLI).

To install the software expand the archive by double clicking on the archive file and then copy the pdfspy directory to the /Applications directory or another location. You may also

wish to add the installation path to the PATH environment variable.

Unix

PDFspy for Unix is delivered as a gzip compressed tar file. You extract the contents of this with GNU tar with either:

tar -xvz -f pdfspy_1.7.X_<OS>.tar.gz

or

gunzip pdfspy_17.X_<OS>.tar.gz
tar xvf pdfspy_1.7.X_<OS>.tar

There is a README.txt file in the pdfspy directory with installation instructions specific to the OS.

Obtaining a serial number

You will need to run the pdfspy program to retrieve your system's "host id". Send an email to support@apago.com with this information requesting a pdfspy license key.

Entering a serial number

To register the application, run it from the command line with **only** the -r parameter to specify the serial number. For example:

```
pdfspy -r XXXXXXXXXXXXXXXXX
```

This will create a file in the same directory as PDFspy called "license.psl" which is your registration file. PDFspy needs to be able to locate this in order to run, so we recommend that you either keep it in the same directory OR use the -license command line option when running the software.

3. Using PDFspy

You can use PDFspy either manually on the command line, or (more likely) by integrating it into your server tools using integration languages such as ASP, Visual Basic, Perl, and Python.
The simplest command to retrieve information about a PDF file would look like:

```
pdfspy somedoc.pdf
```

This command will create an XML description of everything that PDFspy can determine about the PDF. Because this output can be quite long as not as easy to work with directly

as “stdout” (standard output), it is recommend that you add the `-o` option to the command line to have PDFspy write the XML to a file of your choosing. You can either specify a filename (and location) or just a location, which will have PDFspy name the output file the same as the PDF (except .xml instead of .pdf).

```
pdfspy -o output.xml somedoc.pdf
```

In order to reduce the plethora of information that PDFspy generates, you can use the `-skip` option along with comma delimited string listing the features you do not need. For example, to not include information about annotations, bookmarks and form fields contained in the PDF, we might say:

```
pdfspy -skip "annots,outlines,fields" -o output.xml somedoc.pdf
```

The list of valid strings that can be included in the `-skip` options are:

actions	annots	colors	fields
fonts	images	links	names
ocgs	opi	outlines	pagedata
patterns	pdfx	shadings	text
transparency	vectors	xmp	

Another way to select which information is extracted is the `-only` option. It takes the same comma delimited string as `-skip`. Only the features listed will be extracted. Make sure you enable pagedata if you want related information such as colors and fonts.

If you only wish information about a subset of the pages of a PDF, you can use the `-f` and `-l` options to specify the first and last page (respectively) to be processed. For example, to get information about only the second page of the PDF, we might say:

```
pdfspy -f 2 -l 2 -o output.xml somedoc.pdf
```

If you would like more information about what actions the software is performing and to follow its progress in doing so, you can add `-progress` to the command line. PDFspy also supports the writing of this information to a log file `(-log filename)`, which you could use as follows:

```
pdfspy -log log.txt -progress somedoc.pdf
```

If you need to process password protected PDFs, you can use the `-opw` and `-upw` options to specify either the document’s owner or user password. If you attempt to process a secure PDF without providing this information, PDFspy will return an error. You can also take advantage of “wildcards” to provide PDFspy with a list of files to be processed and then specify a directory for the output. For example, to export information

about all of the PDF files in the current directory and place them in an existing directory called outputDir, you could use:

```
pdfspy -o outputDir *.pdf
```

Another unique feature of PDFspy is the ability to use it as a PDF validation tool. When the `-validate` option is added to the command line, PDFspy will perform many of the same operations that Acrobat will in order to render each page to screen including checking font and image data. Any errors and issues discovered will be reported to the log. The validate option does not perform a comprehensive verification against the Adobe PDF Specification. The primary difference between validate and non-validate modes is that PDFspy will 'execute' the drawing operations for each PDF page. Think of it as a dynamic analysis of the PDF file instead a static analysis. The validate option is useful for finding PDF files that have invalid commands or have been corrupted in some way, eg. truncated image data.

```
pdfspy -validate -log report.txt -o output.xml somedoc.pdf
```

You can always get help from PDFspy by using any of the following command line options.

```
-v      print copyright and version info
-h      print usage information
```

PDFspy also supports remembering user preferences. These preferences are stored in an optional PDFspy.setup file. On Windows and OS X systems this file is stored in the same directory as the executable, on Unix systems it is stored in /usr/apago/setups/PDFspy.setup. You may edit this file with any standard text editor (notepad, vi, ...). The format is a simple "option = value" format where the option is the command line parameter without the leading '-' and the value is either the command line value if the argument takes a value or "true" or "false" if it is a boolean parameter. This file is completely optional and the pdfspy program will operate without it.

4. PDFspy Output

PDFspy writes its output as an XML documents using a custom grammar optimized for storing information about the attributes/features of a PDF file. This section will document that grammar to enable programmatic and human understanding of the contents.

PDFspy output files will be valid XML documents with a root element of `<pdfattrs>`. The `<pdfattrs>` element will have a series of attributes that reflect some general information about the document. An example might look like:

```
<?xml version="1.0" encoding="UTF-8"?>
<pdfattrs encrypted="false" filesize="423948" linearized="false"
objects="85" pages="6" tagged="false" version="1.3" >
</pdfattrs>
```

The complete list of attributes that might appear on the <pdfattrs> element, the potential values and a description of each can be found in the following table.

path	string	full path to the file in the local file system
version	string	PDF version (eg. 1.1, 1.3, 1.7, etc.)
compressed-objects	true/false	does the PDF contain PDF 1.5 compressed objects
xref-streams	true/false	does the PDF contain PDF 1.5 xref streams
id or id1 & id2	hex string	one or more unique ID's for the PDF
linearized	true/false	is the PDF prepared for "fast web viewing"
tagged	true/false	does the PDF contain tags/structure for accessibility
pages	integer	number of pages in the PDF
objects	integer	number of objects in the PDF
filesize	integer	size of the PDF on disk
encrypted	true/false	is the PDF encrypted/secured
usage-rights	true/false	does the PDF contain Acrobat Reader Permissions

4.1 <info>

Almost every PDF will produce an <info> element as a child to <pdfattrs>. This element will contain all the information from the original PDF metadata - called the "Info Dictionary". It is a set of "key/value" pairs of strings. The name of each child of <info> is the "key" and its contents are the value.

Example:

```
<info>
   <Creator>Word</Creator>
```

```
    <Producer>Mac OS X 10.2 Quartz PDFContext</Producer>
    <CreationDate>09/10/02 04:05:10</CreationDate>
    <ModDate>09/10/02 04:05:10</ModDate>
</info>
```

4.2 <Extensions>

In 2007, Adobe submitted the PDF file format to the standards body ISO. The ISO published ISO 32000-1:2008 which documented the PDF 1.7 file format as an open international standard. Any additions to PDF 1.7 are considered "extensions." The <Extensions> element contains the information about any extensions in the PDF.

Example:

```
<Extensions>
    <ADBE BaseVersion="1.7" ExtensionLevel="3" />
</Extensions>
```

4.3 <permissions>

If the PDF document is encrypted, then PDFspy will add a <permissions> element as a child to <pdfattrs>. This element will contain children elements that specify which features of the PDF have been enabled or not.

Example:

```
<permissions>
    <printing>yes</printing>
    <copying>yes</copying>
    <changing>yes</changing>
    <commenting>yes</commenting>
    <form-filling>yes</form-filling>
    <screen-reading>yes</screen-reading>
    <document-assembly>yes</document-assembly>
    <high-quality-printing>yes</high-quality-printing>
</permissions>
```

4.4 <ViewerPreferences>

A PDF document may contain some information in it that tell Acrobat how to display it to the user with respect to the Acrobat user interface. For example, the PDF may not want Acrobat to display its toolbar or menubar when it is open. This element contains a series of attributes that document which items are enabled or not.

Example:

```
<ViewerPreferences CenterWindow="true" DisplayDocTitle="false"
HideMenubar="true" HideToolbar="true" HideWindowUI="false"/>
```

4.5 <OutputIntent>

A PDF document that has been prepared according to the international standard (ISO) PDF/X or PDF/A standards will contain a special set of objects called an OutputIntent.

Example:

```
<OutputIntents>
   <v0 OutputConditionIdentifier="CGATS TR 001"
RegistryName="http://www.color.org" S="GTS_PDFX"
Type="OutputIntent"/>
</OutputIntents>
```

4.6 <OCGs>

Acrobat 6 (PDF 1.5) introduced a new feature to PDF called Optional Content Groups (OCGs), commonly know as layers, in the user interface. If a PDF contains any of these OCGs, then their names and some basic attributes will be exported in a <OCGs> child element.

Example:

```
<OCGs>
   <ocg name="GraphicsOC"/>
   <ocg name="TextOC"/>
</OCGs>
```

4.7 <action>

When a PDF document is first opened, Acrobat can perform a series of actions from going to a particular page (and view) to playing a sound or running a JavaScript. Any actions found will be output in an <action> child element whose attributes provide details about the action itself.

Examples:

```
<action page="1" view="Fit"/>
<action key="OpenAction" page="1" type="GoTo" view="XYZ" view-
left="-32768" view-top="-32768" view-zoom="0.25"/>
```

4.8 <names>

PDF documents may contain a series of objects that will be referred to indirectly via their names instead of their object numbers. These objects are organized here, but their subtypes (which become child elements to <names>).

Examples:

```
<names>
   <JavaScript>
```

```
        <script name="Calculator">var
display=this.getField(&amp;quot;Display&amp;quot;); var
func=this.getField(&amp;quot;Func&amp;quot;); all_cancel();</script>
    </JavaScript>
    <AP>
        <appearances name="Draft"/>
        <appearances name="FacesGrumpy"/>
        <appearances name="FacesSmirk"/>
        <appearances name="FacesSurprised"/>
    </AP>
    <EmbeddedFiles>
        <file name="Sample.joboptions">Sample.joboptions</file>
    </EmbeddedFiles>
</names>
```

4.9 <AcroForm>

One common type of PDF is a "form" - a PDF that contains a variety of elements where users can enter data or interact just as that might with a web browser or electronic version of a paper form. In PDF "lingo" there are called "AcroForms", and so information about them is placed into an <AcroForm> element. The children elements will always contain a <fields> (which is the list of individual <field> elements with attributes and children that tell you all you need to know about the fields), but may also contain other information.

Example:

```
<AcroForms>
        <fields>
            <field background-color="[0 0 0]" border-
color="[0.50195 0.50195 0.50195]" border-style="B" border-
width="2" default-appearance="/Helv 21 Tf 1 1 0 rg" default-
value="0" height="47.7612" name="Display" spell-check="true"
text-alignment="right" type="text" value="0" width="204.478"
x="197.393" y="518.905">
                <actions>
                    <action key="K"
type="JavaScript">AFNumber_Keystroke(8, 0, 0, 0, &amp;quot;&amp;quot;,
true);</action>
                    <action key="F"
type="JavaScript">AFNumber_Format(8, 0, 0, 0, &amp;quot;&amp;quot;,
true);</action>
                </actions>
            </field>
            <field default-appearance="/Helv 0 Tf 0 1 1 rg"
name="Nine" type="push-button">
                <kids>
```

```
                        <field background-color="[0 0 0]" border-
color="[0.50195 0.50195 0.50195]" border-style="B" border-
width="2" caption="9" caption-alternate="9" default-
appearance="/Helv 0 Tf 1 1 1 rg" height="28" hilight-mode="P"
width="37" x="274" y="441">
                              <actions/>
                        </field>
                    </kids>
                </field>
    </fields>
</AcroForms>
```

4.10 <bookmarks>

Bookmarks or outlines are the elements of a PDF usually displayed on the left side of Acrobat in a hierarchical fashion to enable you to navigate around the PDF. Each bookmark may be the parent of other bookmarks and have similar attributes to <actions>.

Example:

```
<bookmarks>
    <bookmark page="3" title="Table des matières" type="GoTo"
view="XYZ" view-top="756"/>
    <bookmark page="9" title="Préface" type="GoTo" view="XYZ"
view-top="786">
        <bookmark page="10" title="Aperçu du système" type="GoTo"
view="XYZ" view-top="733"/>
        <bookmark page="10" title="Aperçu du document"
type="GoTo" view="XYZ" view-top="501"/>
    </bookmark>
</bookmarks>
```

4.11 <page>

For every page in a PDF, there will be a <page> element. This element is the root of a series of child elements and attributes that will provide all the information that is page-specific.

The attributes for the <page> element are:

width	integer	width of the page, in PDF units
height	integer	height of the page, in PDF units
Id	integer	page number

rotate	integer	page rotation, if any
user-unit	integer	PDF 1.6 scaling factor
thumbnail	true/false	does the page have an embedded thumbnail
label	string	the page's textual label (if any)
separation	string	name of the color in a pre-separated PDF

4.11.1 <boxes>

Every page in a PDF must have at least one "box" that define the physical size of the page - the MediaBox. It may, in addition, have up to 4 other boxes that define other areas of the page of importance to other processes.

Example:

```
<boxes>
   <MediaBox height="1224" width="792" x="0" y="0"/>
   <CropBox height="688.65" width="464.6" x="13" y="523.35"/>
   <BleedBox height="557.813" width="331.25" x="129.688"
y="613.062"/>
   <TrimBox height="529.688" width="293.75" x="148.438"
y="622.437"/>
</boxes>
```

4.11.2 <fonts>

In order to place text on a page, there must be a font associated with it - and this child element lists the id's of each font that is referenced/used on this page. The ids can be used to find the matching <font> child element of <pdfattrs>.

Example:

```
<fonts>
   <font id="20"/>
   <font id="16"/>
   <font id="17"/>
   <font id="18"/>
</fonts>
```

4.11.3 <annots>

One of the features of PDF being used more and more each day is the ability to comment and markup a PDF with strikeouts, notes, and such. In addition, multimedia such as sounds and movies can be embedded. These elements call fall under the category of an "Annotation" and all such annotations are listed here in the <annots> element.

Here is an example of some <annot> elements:

```
<annots>
   <annot color="[1 1 0]" colorspace="DeviceRGB" height="22"
modification-date="02/13/01 12:00:43" name="3900012" type="Text"
width="18" x="72" y="631">This is a note!</annot>
   <annot border-width="0" color="[]" colorspace="None" default-
appearance="[1 0 0] r /HeOb 18 Tf" height="69" modification-
date="02/13/01 12:01:33" name="3900018" type="FreeText"
width="94" x="24" y="545">This is a simple text
annotation</annot>
   <annot color="[0 1 0]" colorspace="DeviceRGB" height="20"
modification-date="02/13/01 12:02:12" name="390002c" type="Sound"
width="20" x="22" y="529">Silly sound</annot>
   <annot color="[1 1 0]" colorspace="DeviceRGB" height="97"
modification-date="07/13/05 14:01:03" name="3900039"
opacity="0.505005" type="Stamp" width="97" x="512" y="693"/>
</annots>
```

4.11.4 <links>

It is very common to have a PDF documents with active links/hyperlinks to other parts of the same PDF, another PDF or any arbitrary URI/URL. These links are listed here in the <links> element, with individual <link> elements with attributes similar to <action> for all the details of the link.

```
<links>
   <link border-style="S" border-width="1" height="14.9998"
hilight-mode="I" page="2" type="GoTo" view="Fit" width="13.2351"
x="449.112" y="33.5374">
         <actions/>
   </link>
   <link border-style="S" border-width="1" height="15.8822"
hilight-mode="I" href="http://www.thelivingshadow.com/"
type="URI" width="299.997" x="110.293" y="33.5374">
         <actions/>
   </link>
   <link height="45.9249" name="ÓÏÉªÇGríjKX?Main" page="1"
type="GoTo" view="XYZ" view-top="748.042" width="180.704"
x="41.4327" y="431.56">
         <actions/>
   </link>
```

```
   <link height="47.9216"
href="http://www.lazerware.com/software.html" type="URI"
width="182.701" x="39.4359" y="359.678">
        <actions/>
   </link>
</links>
```

4.11.5 <actions>

In addition to document level actions (see 4.6), each page my have a series of Actions associated with it. If present, an `<actions>` element will be present along with each `<action>` child element.

Here is an example of a page's open action that will start playing a movie:

```
<actions>
   <action key="O" title="Sample" type="Movie"/>
</actions>
```

4.11.6 <vectors>

A page of a PDF document may contain three types of data - vectors (lines, paths, etc.), text and raster/bitmap images. If the document contains vector data, then this element will be present with two attributes - `paths` & `subpaths`. Each number represents a count of the total number of each on that page.

```
<vectors paths="57" subpaths="57"/>
```

4.11.7 <text>

When a page of a PDF contains text, the `<text>` element will be present along with two attributes. The `characters` attribute is a count of the number of characters/letters present on that page - whether they are visible or invisible. If a page is a scan, but contains "hidden text" as would be present if the scan had been through an OCR process, then the `hidden-text` attribute will have a value of true, otherwise it will be false.

```
<text characters="2781" hidden-text="false"/>
```

4.11.8 <images>

When there are raster/bitmap images present in a PDF, each one will be listed under the `<images>` element an element of type `<image>` which can contain a lot of attributes with information about the image.

Some example images.

```
 <images>
   <image bpc="8" colorspace="DeviceGray" columns="648"
filter="DCTDecode" height="416" id="32" rows="1083" width="249"
x="158" xres="187.373" xscale="38.4259" y="81" yres="187.442"
yscale="38.4118"/>
```

```
   <image BlackIs1="false" bpc="1" columns="63"
filter="CCITTFaxDecode" height="5" id="21" invert="false"
mask="true" rows="34" width="8" x="176" xres="567"
xscale="12.6984" y="463" yres="489.6" yscale="14.7059"/>
</images>
```

The attributes that can appear on an `<image>` element include:

bpc	bits per component (1, 2, 4, 8, 16)
colorspace	colorspace of the image (RGB, CMYK, etc.)
columns	how many pixels wide is the image
filter	compression type (eg. DCTDecode==JPEG)
height	actual height of the image (in PDF units)
id	object ID of the image
rows	how many pixels high is the image
width	actual width of the image (in PDF units)
x	horizontal location where the image is drawn (in PDF units)
xres	horizontal resolution (dpi) of the image
xscale	horizontal scaling factor of the image
y	vertical location where the bottom of the image is (in PDF units)
yres	vertical resolution (dpi) of the image
yscale	vertical scaling factor of the image

4.11.9 <colors>

In addition to fonts, one of the most common things that users wish to know about their PDFs is what colorspaces are used in the document. It is only Black & White, was RGB used, etc. This section provides the list of colorspaces found on each page. If the colorspace is a "complex" one, such as Indexed or DeviceN, then the details of that space are present as child elements to that.

Here is an example of simple and complex colorspaces:

```
<colors>
   <color base="DeviceGray"/>
   <color base="DeviceRGB"/>
   <color base="DeviceCMYK" used="CM"/>
   <color base="CalRGB"/>
   <color base="Indexed">
        <color alt="DeviceRGB" base="ICCBased" comps="3"
name="IEC 61966-2.1 - sRGB"/>
   </color>
   <color base="DeviceN" alt="DeviceCMYK"
            simulation="cmyk(0,0,0,1.0)">
        <color base="Separation" alt="DeviceGray">Black
        </color>
   </color>
</colors>
```

For CMYK colors, PDFspy will report with channels are actually used via the "used" attribute. In the example above, the DeviceCMYK colorspace used the Cyan and Magenta channels. The Indexed colorspace had an associated sRGB ICC profile. The DeviceN colorspace was composed of a single separation named "Black" which had an alternate colorspace of DeviceGray.

4.11.10 <opi>

In the prepress community, some users continue to use a technology called OPI (Open Prepress Interface), which enables them to use low-resolution images in their documents and have the higher resolution versions replaced at print/RIP time. If a document contains such info, PDFspy will report the details of the OPI objects in this section.

Example:

```
<opi>
   <opidict>
        <_1.3 ColorType="Process" Type="OPI" Version="1.3">
             <Size v0="2100" v1="2550"/>
             <F F="SERVER BETA:Film Scans:JONAH - JO:JONAH CAMP
A:JONAH CAMP A ELEMENTS:VEGGIExa B&amp;amp;W Comp only2.tif"
Type="Filespec"/>
             <Color v0="0" v1="0" v2="0" v3="1" v4="Black"/>
```

```
            <CropRect v0="538" v1="331" v2="1831" v3="2493"/>
            <CropFixed v0="538.003" v1="331.418" v2="1830.58"
v3="2493.3"/>
            <Position v0="172.244" v1="714.865" v2="172.244"
v3="1129.97" v4="420.501" v5="1129.97" v6="420.501"
v7="714.865"/>
            <Resolution v0="150" v1="150"/>
            <ImageType v0="1" v1="8"/>
        </_1.3>
    </opidict>
</opi>
```

4.11.11 <patterns>

PDF documents may contain patterns - either of the tiled variety (which are the ones which consist of a set of elements repeated multiple times to fill an area) or shaded (otherwise known as smooth shades or gradients). If either are present, PDFspy will report on it via the <patterns> element with attributes for each type and the number of occurrences of that type.

```
<patterns tiled="1" shading="96"/>
```

4.11.12 <transparency>

Introduced to PDF in version 1.4, the ability to have objects be transaparent has caused a lot of stir in the PDF world - both good and bad, esp for folks trying to output such documents on older RIPs/printers. To be able to check if a document uses this feature, PDFspy will report if there is any transparency present via the <transparency> element with a count of how many objects use it.

```
<transparency objects="2"/>
```

4.12 <colors>

As mentioned in section 4.10.9, each page of a document will contain a list of colorspaces used on that page. However, sometimes you just want a list of ALL colorspaces used in a document. The <colors> child element of <pdfattrs> is where you will that. It has the same syntax as the page-based <colors> element, but is a list of unique colorspaces for the entire document.

4.13 <fonts>

As mentioned in section 4.10.2, the list of fonts on each page is just the id of the font object in the PDF, but doesn't include the details of that font. Instead the <fonts> child element of <pdfattrs> includes the details of each font used in the document.

Example:

```
<fonts>
   <font embedded=”false” hasToUnicode=”false” id=”121”
name=”Times-Roman” subset=”false” type=”Type 1”/>
   <font embedded=”false” hasToUnicode=”false” id=”122”
name=”Times-Bold” subset=”false” type=”Type 1”/>
   <font embedded=”false” hasToUnicode=”false” id=”125”
name=”Helvetica” subset=”false” type=”Type 1”/>
   <font embedded=”true” hasToUnicode=”false” id=”15”
name=”HelveticaNeue-Condensed” subset=”false” type=”Type 1C”/>
   <font embedded=”true” hasToUnicode=”true” id=”16”
name=”ZapfDingbats” subset=”false” type=”Type 1C”/>
</fonts>
```

4.14 <shadings>

The <shadings> child element of <pdfattrs> contains one attribute, objects, which is the number of shading objects in the PDF file.

5. Frequently Asked Questions

What is PDFspy?

PDFspy is a tool for extracting useful information out of a PDF file such as the used fonts and colors. It can also report on a variety of other information including image attributes, use of transparency, annotation and form field settings, etc.

Why do I need PDFspy?

1) You are building your own custom “PDF Preflight” tool and don’t wish to learn the details of PDF.
2) You are building a PDF reporting tool, such as annotation or form field usage
3) You need a PDF validation tool and wish to use PDFspy’s validation feature
4) You want to extract text from a PDF file.

What type of PDF files does PDFspy support?

PDFspy is compatible with all types of PDF files - from version 1.0 to the current 1.7.

What OS platforms is PDFspy available for?

PDFspy is available for Mac OS X, Windows, Linux and Solaris.

What are the system requirements to run PDFspy?

For Mac OS you will need an Apple Macintosh Intel-based computer running Mac OS 10.6 or later with at least 1GB of RAM.

For Windows you will need a Intel or AMD based computer running Windows XP or later with at least 1GB of RAM.

Does PDFspy require Adobe Acrobat?

No.

Can I try PDFspy before purchasing it?

Yes, contact Apago for a trial serial number.

Where can I purchase PDFspy?

You can purchase PDFspy at Apago's store @ http://www.apagostore.com

How do I report a bug or submit a suggestion?

Please send an email to support@apago.com or call +1 770 619-1884.

6. Legalese