UserGuide PDFtextCMD.exe (32 bit)

It's a 32-bit-commandline-tool, completely steered by parameters.
It can be called via shell commands from nearly any programming language or from a bat-file, too. We're offering a dll-version with similar functionalities (PDFtext.dll), too...
But if you have problems working with dynamic link libraries (dll) in your application you can embed this PDFtextCMD.exe and it will work in a proper way nearly everywhere.

You can deliver and distribute this exe as a part of your programming solution without limitations (royalty free). It's forbidden to deliver this exe alone.

You'll find the extracted textcontent optionally in a new textfile (fill out parameter "target") or in the clipboard (to choose clipboard or textfile you have to use parameter "opt").

The exe returns with exitcodes.
These exitcodes can contain value 1 (for success), the pagecount of a document or an errorcode.
There are complete **samples-projects (vb6, vb.net, C# and delphi)** included to show you how to check and catch these exitcodes.
Additionally there are many bat(ch)-files included to show you how to use PDFtextCMD.exe directly on the commandline (cmd).

This is the test version!
This means "try before you buy"!
If you think this exe can be useful for your work please order the unlimited version at:
www.PDF-Analyzer.com or www.IS-Soft.de
This version isn't limited in any cases 'cause you should be able to test the whole functionality.
There are only two differences to the full version:
The message window at programstart and the text "Testversion!" in application windows.

If you still use the testversion you can get the unlimited version from:

Ingo Schmoekel
- Software-Dev. & Distribution -
Zedernstr. 30a
D-28832 Achim
GERMANY

Webmaster@PDF-Analyzer.com
http://www.PDF-Analyzer.com
http://www.IS-Soft.de

Points that should be considered

If you want to do textextraction into file at least the first five parameters have to be used.
If you want to do textextraction into clipboard at least the first four parameters are a must and target should be a short **-**
If you are using only one or two parameters the pagecount will be returned.
If you are using one parameter (the file) pagecount returns as an exitcode-value.
If you are using two parameters the pagecount returns as clipboard-content (opt=2) or file-content (opt=1). The textfile will be the pdf-filename with an additional ".txt" as new suffix.
If you only want to check in a - for example - regularely way if a document was changed in the meantime you can get the hash value (hw=1).
More than five parameters ... up to thirteen are possible.
If you're using more than four parameters but less than all thirteen all unaffected parameters will be set with standard values.
If you're using path- and file-names with spaces you should enclose them with double quotes. Each parameter should be separated by a space.

The parameters

FileName	= The input file for textextraction
opt	= The extract type (file or clipboard)
hw	= Only a hash value will be returned to see if a document was changed in the meantime
fast	= The mode of textextraction
target	= Path and file where the pdf-content shall be extracted as a textfile (**-** means no target)
lspaces	= With leading spaces in the extracted textcontent
ptitel	= A title on each page like "page ... from ..." or for example "Seite ... von ..." (- means no page title)
pos	= With positions (row and column in pixels) for each textstring
page	= A single page number where the textcontent shall be extracted
clock	= If the user shall see a sandclock while extraction
blank	= If more than one blank between words shall be deleted -content
ende	= Break after xxxx seconds 1-1800 or 0
wlist	= The textcontent will be returned as a stringlist

Details regarding the parameters

FileName = If you're already in the relevant path you can take the real filename. That's enough. If it's a document from another path you should use drive, path and filename. If there are spaces inside you should enclose FileName with double quotes. A correct entry (drive/path/file) here is a must.

opt = If you want textextraction into a new textfile you should use 1. If the textcontent shall appear in the clipboard you should use a 2.
Value 1 or 2 is a must.

hw = If you only want to check if a document was changed in the meantime you can set hw to value 1 to get back a unique hash value calculated regarding the filecontent.
Value 0 or 1 is a must.

fast = The textextraction is faster with value 1 but then positions for each string aren't available. Value 0 extracts string by string - so it's a bit more slow.
Value 0 or 1 is a must.

target = If "opt = 1" was used a correct value for target (the textfile) is a must. Drive, path and filename should be used. Drive and path should be already exist. If you don't need/want a textfile you should insert there a single **-**

lspaces = It's for supress leading spaces at the beginning of a string. 1 means no leading spaces - 0 means with leading spaces.
Value 0 or 1 here is a must.

ptitel = To keep the overview in longer textextractions you can insert on each textpage something like "page x from x" or "Seite x von x". If you want this you should insert something like "page" (for english), "Lado" (for spain), "Seite" (for german) or whatever you want.
A value here is a must - If you don't want a title on each page you can insert a single **-**

pos = If you want a textoutput "string by string" with the leading positions (actual page, pagecount, row and column).
Value 0 means no positions - Value 1 means with positions. If you're using value 1 parameter "fast" should be set to 0.
A value here is a must.

page = If you want to extract only the textcontent of a special page you can insert here a value between 1 and 99999.
Value 0 means all pages. A value here is a must.

clock = 0 means no sandclock while the function is working... Value 1 shows the sandclock.
One value 0 or 1 is a must.

blank = There are documents with justification layout. This means that sometimes (to fill each line) there are more than one space between words. If you're using PDFtextCMD.exe to search in a second step through extracted textcontent it could be easier if you know that between words are only one space. If "blank" is set to 1 more than one space between words will be deleted.
A value here is a must.

ende = Sometimes you can get very voluminous documents with much different contents - The extraction can last many seconds... minutes. Sometimes it's enough if you've extracted the first pages... Here you can set a value that means "stop extracting after xxxx seconds".
0 means no limit, 1 until 1800 means from 1 second until 30 minutes (that should be enough).

wlist = If value 1 was set here the textcontent will be returned as a wordlist - each word in a single row. Value 0 or 1 is a must.

Error-messages

The program normally returns with the extracted textcontent as the result. If there's something wrong different error messages are possible via returned exitcode ...

0	was set when the app is starting ... means no success
1	means common success
9	means common error
9001	pdf document wasn't found
9002	first parameter isn't a pdf document
9003	the pdf document is protected with a user-password
9004	the pdf document has a damaged page structure
9005	target path isn't valid
9006	all parameters are missing
9007	source- and target-path are the same
9013	the pdf document has a damaged directory
9015	codepage 1251 not yet supported
9016	codepage CJK not yet supported
9021	parameter filename isn't valid
9022	parameter target isn't valid
9023	parameter value for ptitel isn't valid
9024	the value for parameter page isn't valid
9025	the value for parameter ende isn't valid
9026	parameter opt can only be 1 or 2
9027	valid parameter values are 0 or 1

Using PDFtextCMD.exe with common IDEs

It's very easy using PDFtextCMD.exe with the shell- and/or execute-syntax offered/supported by nearly all main programming languages. Each module returns with exitcodes to the system telling something about the actual status of the module. The problem is that the exitcodes will return as long as the module is running and that the first exitcode will return immediately – so the first exitcode will deliver the status value for "running".

Regarding this fact it's important to enclose the syntax part with the calling module with a loop to get not only the first but all exitcodes until the module has stopped.

How this can be realized you can see detailed in attached complete sample projects with source code for C#, VB6, vb.NET and Delphi.
Using PDFtextCMD.exe with scripting languages like the common bat(ch)-files you can see below …

Using PDFtextCMD.exe on the commandline could look like this ...

"c:\temp\PDFtextCMD.exe" "c:\temp\sample.pdf" 1 0 1 "c:\temp\sample_pdf.txt" 0 page 0 0 0 0 0 0

c:\temp\sample.pdf	the text shall be extracted from document sample.pdf ...
1	into a textfile (target is necessary) ...
0	no hash-value needed
1	the fast method shall be used ...
…\sample_pdf.txt	this is the textfile which will contain the extracted textcontent ...
0	leading spaces shouldn't be deleted ...
page	on the top of each page there shall be a "page x / xx" ...
0	no string positions needed ...

0 not one but all pages shall be extracted ...
0 no sandclock shall appear ...
0 multiple blanks shouldn't be deleted ...
0 the function shall run to it's end - no maximum seconds ...
0 no word-list needed.
The exitcode will contain value 1 (=okay).

"c:\temp\PDFtextCMD.exe" "c:\temp\sample.pdf" 2 0 0 - 0 - 1 0 1 0 0 0

c:\temp\sample.pdf the text shall be extracted from document sample.pdf ...
2 into the clipboard ...
0 no hash-value needed
0 the method "string by string" shall be used ...
- into the clipboard - so no textfile needed ...
0 leading spaces shouldn't be deleted ...
- no page counter on the top of each page ...
1 extract with positions row/column ...
0 not one but all pages shall be extracted ...
1 we want to see a sandclock ...
0 multiple blanks shouldn't be deleted ...
0 the function shall run to it's end - no maximum seconds ...
0 no word-list needed.
The exitcode will contain value 1 (=okay).

"c:\temp\PDFtextCMD.exe" "c:\temp\sample.pdf" 1 1 1 "c:\temp\sample_pdf.txt"

c:\temp\sample.pdf the text shall be extracted from document sample.pdf ...
1 into a textfile (target is necessary) ...
1 hash-value needed
1 the fast method shall be used ...
c:\temp\sample_pdf.txt this is the textfile which will be created with the calculated hash value ...
The exitcode will contain value 1 (=okay).

c:\temp\PDFtextCMD.exe "c:\temp\sample.pdf"

c:\temp\sample.pdf document sample.pdf shall be processed ...
Because there's no further parameter the value for pagecount will be returned as exitcode.

c:\temp\PDFtextCMD.exe "c:\temp\sample.pdf" 2

c:\temp\sample.pdf document sample.pdf shall be processed ...
Because there's no further parameter the value for pagecount will be returned as clipboard value.
The exitcode will contain value 1 (=okay).

Make it easier to get the exitcode

If you're on the commandline (the black window at menu item accessories) running your bat-files with PDFtextCMD.exe you can see the exitcode on the last line of the processing script-output like this sample with exitcode 1:
"Runtime error 1 at 008CBED0"

If you want to use bat-files but also an automatic analysis about the exitcode you can use the pipe-syntax (the pipe-symbol is the greater-sign >
A sample could be like this:

"c:\temp\PDFtextCMD.exe" "c:\temp\sample.pdf" 1 1 1 "c:\temp\sample_pdf.txt" **>**"c:\temp\result.txt"
The pipe-syntax specifies what to do with the output.
In this sample the output (the line with the exitcode) will be saved into file c:\temp\result.txt.
So later it's easy to analyse the filecontent regarding success or error via the file content.

Bat-files for getting the pagecount

Sample 1

The first three lines of text in our sample are only for description of the batchfile:

```
rem *---------------------------------------------------------*
rem PDFtext via commandline
rem *---------------------------------------------------------*
```

The following last line is the only important line:

```
c:\temp\test\PDFtextCMD.exe "c:\temp\test\inputpdf.pdf"
>"c:\temp\test\inputpdf_pagecount.txt"
```

The first part on the commandline (cmd) is always the application itself with drive, path and filename.
The next part is always the pdf-document which shall be processed with drive, path and filename.
If this are all parameters the application knows that the pagecount shall be returned.
Because there's no file- or clipboard-option the value will be returned as exitcode.
In this sample here the string ends with an additional `>"c:\temp\test\inputpdf_pagecount.txt"` … This has nothing to do with the application. It's from the usual script for bat-files and want to say with the so-called pipe-symbol **"**>**"** that the result of this batchfile shall be written into a textfile.
These textfile content will look like this sample:
`Runtime error  28 at 008CBED0` (the 28 is the pagecount value).

Sample 2

```
rem *---------------------------------------------------------*
rem PDFtext via commandline
rem *---------------------------------------------------------*
```

The following last line is the only important line:

```
c:\temp\test\PDFtextCMD.exe "c:\temp\test\inputpdf1.pdf"
```

The first part on the commandline (cmd) is always the application itself with drive, path and filename.
The next part is always the pdf-document which shall be processed with drive, path and filename.
Because there aren't any more parameters the application returns the pagecount as exitcode.
These errorlevel-line will look like this sample:
`Runtime error  28 at 008CBED0` (the 28 is the pagecount value).

Sample 3

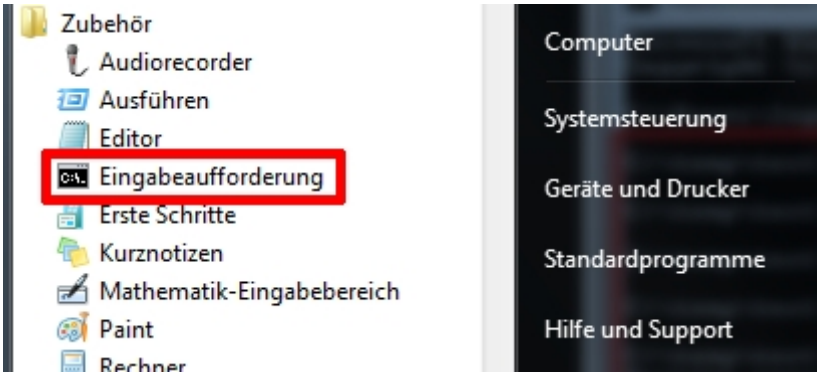
```
rem *---------------------------------------------------------*
rem PDFtext via commandline
rem *---------------------------------------------------------*
```

The following last line is the only important line:
c:\temp\test\PDFtextCMD.exe "c:\temp\test\inputpdf1.pdf" 2

The first part on the commandline (cmd) is always the application itself with drive, path and filename.
The next part is always the pdf-document which shall be processed with drive, path and filename.
The last parameter is the value **2**. This means that the pagecount value shall be put into **clipboard**.
If the last parameter is value **1**, this means **fileoutput**. The pagecount value will be written into a file which has the same name as the processed file with an additional ".txt" as suffix.

Few samples regarding bat(ch)-files illustrated by screenshots

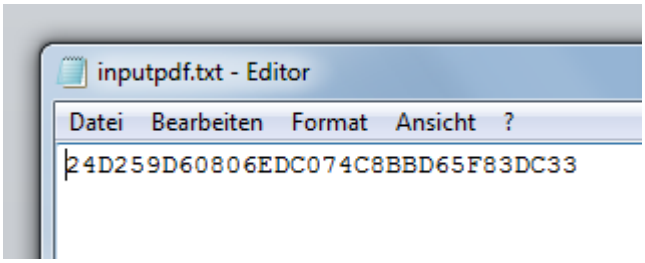
Via "Accessories" and "Command prompt" you can activate the system-window (the command-line)…



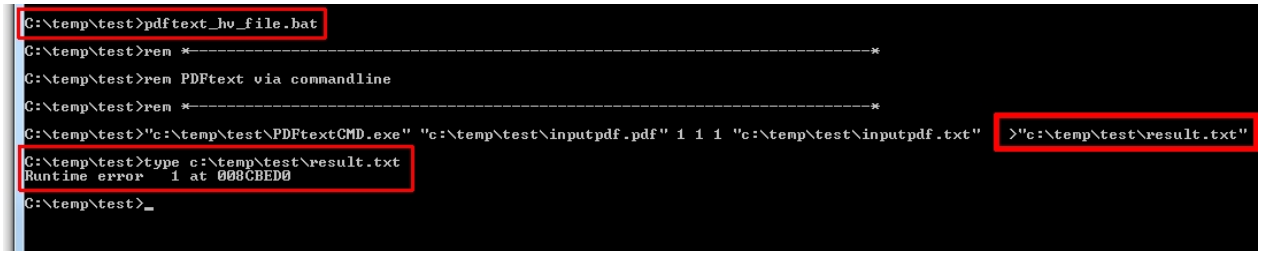
Getting the hash value from a pdf-document …

1. With the cd-syntax you change to the sample-directory c:\temp \test …
2. There you can call the self executing bat-file …
3. The processing rows of the bat-file writes on the screen (rem …) …
4. The long syntax call from the bat-file (with PDFtextCMD.exe) will be executed …
5. The returning exitcode will be displayed as an error level on the screen (Runtime error 1 …) …
6. Exitcode 1 means that the process was executed in a proper way.

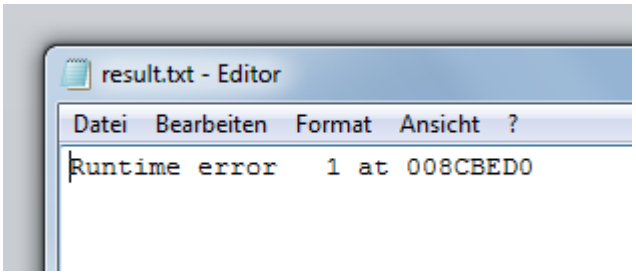
7. And the file inputpdf.txt contains the hash value …



Getting the hash value from a pdf-document and save the process result …



1. We're in the sample-directory c:\temp \test …
2. There you can call the self executing bat-file …
3. The processing rows of the bat-file writes on the screen (rem …) …
4. The long syntax call from the bat-file (with PDFtextCMD.exe) will be executed …
5. The returning exitcode will be redirected and saved with the pipe-symbol (**>**"c:\temp\test\result.txt") into the file result.txt …
6. Showing the content of result.txt with "type" and you can see what's inside: The error level line …



7. And the file inputpdf.txt contains the hash value …