VISION DOCUMENT

FDsys PDF Image Extraction

1.0 Document Revision History

Date	Description of revision
2009-10-19	FDsys PDF Image Extraction Draft
2009-11-15	Milestones and timeline added.

2.0 Project Information

SPONSORING ORGANIZATION (SO): United States Government Printing Office
EXECUTIVE SPONSOR: Selene Dalecky, Scott Stovall
POINT OF CONTACT (POC): Blake Edwards (Liaison), Lisa LaPlant, Deng Wu
PROGRAM/PROJECT/TASK MANAGER: John Hautzinger (Team Co-Leader), Christopher Bly (Team Co-Leader), Dr. Chengyu Sun (Faculty Advisor)
TARGET COMPLETION DATE: May 2010

3.0 Overview of the Proposed Activity

Goal:
Extract images from PDF files and associate the images with HTML pages.

Background:
The United States Government Printing Office (GPO) is currently implementing the Federal Digital System (FDsys). FDsys is comprised of commercial, off-the-shelf (COTS) software, custom-developed software, and hardware platforms that will enable federal content originators to easily submit to GPO the content that can be authenticated, versioned, preserved, and delivered upon request. FDsys will become the content management system for federal publications within the GPO enterprise.

Some of the content that is submitted into the FDsys application is not in a text-searchable format (e.g. TIF, scanned PDF). This software utility will provide the capability to take images that are present in the PDF documents, optically recognize the contained characters, and insert the image and text into the HTML and XML files. These documents will then be stored in FDsys and delivered upon request.

Problem Statement:
There are four types of images that are present in PDF files that are not made available in HTML files:

- Images representing a full or partial page of text
- Tables and charts
- Scientific formulas
- Pictures and other images

Currently, when GPO provides access to the documents where image files are present, the text in those images (or text describing the image) is missing from the HTML and XML files. This software utility will allow GPO to provide access to HTML and XML files that include this text and the image itself.

Objectives:
The tasks required to meet the goal include:

- Conduct market research and identify candidate tools which extract images from PDF files and use optical character recognition (OCR) to convert the images to text.
- Create and populate a decision tool (i.e., Pugh Matrix) with recommendations for the tool to be used.
- Develop software utility based on the selected extraction tool to extract the images and associate them with HTML and XML pages.

The software utility should:

1. Develop reusable images from PDF files.
    - Extract images from PDF files.
    - Generate text from images in PDF files using OCR.
    - Organize images and associated text in a manner to make them reusable.
2. Insert the text and graphic into existing files in line with the text. (Note: For pictures and other images, the text would be the caption for the associated graphic.)
    - Insert the text and graphic into existing HTML files in line with the text.
    - Insert the text and graphic into existing XML files in line with the text.

The deliverable will be the software utility for extracting images from PDF files for HTML and XML files, source code and the associated design documentation. The utility can include the COTS tool with custom integration code if necessary.

4.0 Stakeholders

Key Stakeholders:

- GPO Program Management Office

5.0 High Level Assumptions

- Sample data will be provided by GPO to the CSULA project team.
- Ideally, the OCR engine will have and Application Programming Interface (API).

- Completed code and design documentation will become the property of GPO at the time of delivery. There will be no restrictions on the use of the design documentation or software utility of code by GPO.

6.0 Anticipated Constraints

- Quality of text captured via OCR is of utmost importance.
- Extraction and OCR processing should take less than 2 minutes per package.
- Extracted images should be in JPEG format.
- All programming should be done in Java.
- The software utility should provide Java APIs for integration with other applications programmatically.
- Logs for the software utility should use Apache log4j.
- HTML should be compliant with Section 508 of the United States Code. The areas of this affected by the text include:
    - § 1194.22 Web-based intranet and internet information and applications.
        - (a) A text equivalent for every non-text element shall be provided (e.g., via "alt", "longdesc", or in element content).
        - (g) Row and column headers shall be identified for data tables.
        - (h) Markup shall be used to associate data cells and header cells for data tables that have two or more logical levels of row or column headers.
    - More information on Section 508 can be found at www.section508.gov.
- All students may be required to pass a standard U.S. Government background check.

7.0 High-level Risks

If the project is not implemented or is delayed, GPO will not be able to provide complete XML and section 508 compliant HTML files to the public.

8.0 Budgetary Estimate (ROM)

There is a $20,000 sponsor fee, plus any associated travel costs.

9.0 Project Milestones and Timeline

Phase 1: 10/26/2009 – 11/16/2009

- Initial market research of PDF image extraction tools and OCR tools.
- Discussion of system architecture and requirements.
- Deliverables
    - Vision Document completed with project milestones and timeline.
    - System Requirements document.
    - Report of the initial tool selection based on the system requirements.

Phase 2: 11/16/2009 – 12/14/2009

- Tool acquisition and testing.
- Deliverables
    - Tool evaluation report and decision of tool selection.
    - Concept of Operations document.

Phase 3-5: 1/4/2010 – 1/25/2010

- System and API design.
- Discussion and certification of the design.
- Deliverables
    - Functional Specification document.

Phase 6.1: 1/26/2010 – 2/22/2010

- System implementation.
- Deliverables
    - Initial implementation of the system.

Phase 6.2: 2/23/2010 – 3/8/2010

- System testing.
- Deliverables
    - System Performance Evaluation report.

Phase 6.3: 3/9/2010 – 3/22/2010

- Project presentation and system delivery.
- Deliverables
    - User documentation.
    - The completed system.

10.0 Sponsor Signature

Date:	Signature:

PdfImage Documentation

Installation

PdfImage consists of two components: a C++ component (*ocr.zip*) that provides a command line utility to perform OCR, and a Java component (*pdfimage.zip*) that handles input and output, image/text extraction and insertion, and work flow control.

Prerequisites

Download and install the latest Java SE Development Kit (JDK). Set a `JAVA_HOME` environment variable to the directory where JDK is installed.

Download Ant and unzip it to a local folder, which we will refer to as `$ANT`. Add `$ANT/bin` to the `PATH` environment variable.

Download and install Glassfish Server Open Source Edition 3.0. By default the server will use port 8080, which is the port number we will use in the rest of the document. Note that Glassfish installation does not automatically start the server, so you have to start it manually using Windows Start Menu -> All Programs -> GlassFish v3 -> Start Application Server. After the server is started, open a browser and go to `http://localhost:8080`, and you should see a page that shows "Your server is running".

Install OmniPage Capture SDK 16. It will create a local folder, e.g. C:\Program Files\Nuance\OPCaptureSDK16. In the rest of the document we will refer to this folder as `$SDK`. Follow the SDK documentation to activate the product.

Install Visual Studio 2005 or above (older Visual Studio may work, too).

C++ Component

Unzip *ocr.zip* to a local folder, e.g. C:\ocr. In the rest of the document we will refer to this folder as `$OCR`.

Open `$OCR/ocr.sln` in Visual Studio. The solution file was created using Visual Studio 2010. If your version of Visual Studio cannot open the file, simply create a C++ Win32 Console project yourself, then add the `ocr.h` and `ocr.cpp` to the project.

Right click on the project and select Properties -> Configuration Properties,

- Change Character Set from `Use Unicode Character Set` to `Not Set`.
- Under VC++ Directories
    - Add `$SDK\Include` to Include Directories.
    - Add `$SDK\Lib` to Library Directories.
- Under Linker -> Input
    - Add `KernelAPI.lib;RecApiPlus.lib` to Additional Dependencies.

Build the project use Release configuration. It should produce an executable file `ocr.exe` under `$OCR\Release`.

Java Component

Unzip *pdfimage.zip* to a local folder, e.g. C:\pdfimage. In the rest of the document we will refer to this folder as `$PDFIMAGE`.

Edit `log4j.xml` and `pdfimage.properties` under `$PDFIMAGE/conf` so they match your local setup.

Open a command prompt, change to `$PDFIMAGE`, and type "`ant deploy`" (without the quotes). The build script will perform several operations, including

- Build the project from source and create two files, `pdfimage.jar` and `pdfimage.war`, under `$PDFIMAGE/dist`.
- Generate Java API documentation.
- Deploy `pdfimage.war` to the Glassfish application server so PdfImage can be used remotely via a web service interface.

There will be lots of warning messages during Javadoc generation. You can safely ignore those messages.

A screen capture video demonstrating this installation process can be found at http://sun.calstatela.edu/~cysun/videos/pdfimage.wmv.

Testing

To test if everything is installed properly, create a folder with a PDF file and its corresponding HTML file. Modify the `test.dir` property in `pdfimage.properties` to point to this folder. Under `$PDFIMAGE`, type "`ant test`". After the program finishes, you should see the output files under a subfolder of the folder where the PDF and the HTML file are located.

Usage

To use PdfImage as a local library, simply include the following files in your classpath:

- pdfimage.properties
- log4j.xml
- pdfimage.jar
- All the jar files under `$PDFIMAGE/WebContent/WEB-INF/lib`

The use PdfImage as a web service, you can access the service WSDL at the following address:

```
http://<host>:8080/pdfimage/PdfImageServiceService?wsdl
```

and use your choice of web service tools to generate the client stub.

Input and Output

PdfImage operates in two modes: *Folder Processing* mode and *List Processing* mode. In the Folder Processing mode, it processes every PDF file and its corresponding HTML file (if exists) in the input folders. In the List Processing mode, it processes an ordered list of PDF files and an ordered list of HTML/Text files. In the List Processing mode, the PDF files and the HTML/Text files may not have one-to-one correspondence, but it is assumed that the files in each list combined together will produce the same content.

Folder Processing Mode

The input for this mode are the following four folders:

- Input PDF folder
- Input HTML folder
- Output PDF folder
- Output HTML folder

The input PDF folder contains PDF files to be processed. For example:

```
\pdf
    a.pdf
    b.pdf
```

The input HTML folder contains HTML files to be processed. For example:

```
\htm
    b.htm
    c.htm
    d.htm
```

Note that the HTML files do not need to match the PDF files one by one, but in the Folder Processing mode, only the HTML files that have matching PDF files will be processed.

The output PDF folder will contain two output Image-Over-Text (IOT) PDF files for each input PDF file, and some intermediate files generated during processing. For example:

```
\output_pdf
    \a
        a.1.pdf
        a.2.pdf
    \b
        b.1.pdf
        b.2.pdf
    \c
        c.1.pdf
        c.2.pdf
```

Note that a subfolder will be created for each input PDF file using its base name (i.e. file name without the .pdf suffix).

The output HTML folder will contain the images extracted from the PDF file, some intermediate files generated during processing, and the output HTML file with the images and the OCRed text inserted. For example:

```
\output_htm
    \b
        b.htm
        Page_0001_Image_0001.jpg
        Page_0002_Image_0001.png
```

Note that only HTML files that have corresponding PDF files will be processed, and a subfolder is created for each processed HTML file.

List Processing Mode

The input for this mode are the following four folders and two ordered list of file names:

- Input PDF folder
- Input HTML folder
- Output PDF folder

Output HTML folder
- A list of PDF file names
- A list of HTML/Text file names

The folders in this mode serve the same purposes as in the Folder Processing mode, except that only the files in the two lists will be processed. An example of this mode is shown below:

- Input PDF folder: `\pdf`
- Input HTML folder: `\htm`
- Output PDF folder: `\output_pdf`
- Output PDF folder: `\output_htm`
- List of PDF file names: `{a.pdf, b.pdf}`
- List of HTML/Text file names: `{x.htm, y.htm, z.htm}`

After process, the output PDF folder will contain the following content:

```
\output_pdf
    \a
        a.1.pdf
        a.2.pdf
    \b
        b.1.pdf
        b.2.pdf
```

The output HTML folder will contain:

```
\output_htm
    \x
        x.htm
        a_Page_0001_Image_0001.jpg
    \y
        y.htm
        a_Page_0002_Image_0001.jpg
        b_Page_0001_Image_0001.jpg
    \z
        z.htm
        b_Page_0002_Image_0001.jpg
```

Special Cases

In both processing modes, the input and the output folders do *not* have to be distinct. In particular, there are two special cases worth mentioning.

First, we expect that GPO would want to maintain the current package structure and keep all the PDF file under the /pdf folder and all the HTML files under the /htm folder. In this case, the input and the output PDF folders can be the same, and the input and output HTML folders can be the same.

Second, during testing, it is convenient to drop all test samples in the same folder and process them. In this case, all four folders can be set to the same.

Components

Structurally PdfImage can be divided into four major components:

- Extractor
- Recognizer
- Inserter
- Config

Extractor is responsible for extracting images from PDF. For the purpose of finding the locations in an HTML to insert the extracted images, the Extractor also extract the text on the page before the image, the page after the image, and the page that contains the image.

Recognizer uses the OCR engine to perform the following operations:

- OCR the extracted images into text.
- OCR the input PDF file to produce an Image-Over-Text (IOT) PDF file.
- Split the input PDF file into text pages and image pages, OCR the image pages into IOT PDF files, then combine these files with the text pages to produce an output IOT PDF file.

Inserter inserts the extracted images and the OCRed text into an HTML file.

Config uses a set of parameters to control the behaviors of Extractor, Recognizer, and Inserter.

Extractor

Extractor is responsible for extracting images from PDF. For the purpose of finding the locations in an HTML file to insert the extracted images, the Extractor also extracts the text on the page before the image, the page after the image, and the page that contains the image.

Image Extraction

Images are stored in PDF as objects with type `XObject/Image`. In its raw form, an image is simply a 2-dimensional array of pixels, where each pixel is represented by a number of bits, e.g. 1 bit for black and white images, 8 bits for gray scaled images, and 24 bits for RGB images. Because images in their raw forms can be very large, compression methods are usually used to reduce their sizes. The compression methods are referred to as *filters* in the PDF specification.

Some of the PDF image filters use the same compression algorithms as some common image formats, and we take advantage of this in our Extractor implementation to achieve maximum image extraction performance. In particular, Extractor identifies two filters that are commonly used in GPO's PDF files, *DCTDecode* and *CCITTFaxDecode*, and handles these two filters differently from other filters.

For the DCTDecode filter, the images objects compressed with this filter conform to the JPEG image format, so we extract these image objects directly into JPG files.

For CCITTFaxDecode filter, there are two different cases to handle. In the first case, the image objects compressed with this filter conform to the TIFF image format, except that the image headers required by the TIFF format are missing. In this case, Extractor creates the following TIFF image headers based on image metadata extracted from the PDF:

- ImageWidth
- ImageLength
- BitsPerSample
- SamplesPerPixel
- PhotometricInterpretation
- Compression
- StripOffsets
- RowsPerStrip
- StripByteCounts

These headers and the image object are then written out into a TIFF image file.

In the second case, the image objects compressed with this filter has a property known as *byte aligned*. What this means is that the encoding bits of a row in an image always starts on a byte boundary. For example, if it takes 10 bits to encode the first row of pixels in an image, the bits of the second row will start at bit 17 (i.e. the first bit of the third byte) instead of bit 11. The byte aligned property is supported by the PDF specification but not by the TIFF standard, so to handle this case, we have to convert the image objects from byte aligned to byte unaligned by removing the "gaps" between rows, which requires actually decoding the image object to locate the row boundaries. However, it should be noted that decoding is only used to locate row boundaries - the image object is not decoded into the raw form and then re-encoded, so this process is still very efficient. Once the image objects are converted to byte unaligned, they can be extracted into TIFF images just like in the first case.

For filters other than DCTDecode and CCITTFaxDecode, we use the LGPL version of the JPedal library to first decode the image objects, then re-encode them into PNG images. This process takes considerably longer than processing DCTDecode and CCITTFaxDecode filters.

The image files extracted use the following naming convention:

    Page_####_Image_####.[jpg|tif|png]

where the first 4-digit number is the page number, and the second 4-digit number is the index of the image within a page.

Text Extraction

PDF is a format designed for rendering and publishing a finalized document, not for editing or intermediate processing. What this means is that *perfect* extraction of text from PDF is not possible. Consider a simple example of a PDF file that show the word "Hello". Although from a reader's perspective, the PDF file contains the text "Hello", but if we look inside the PDF file, we may find that:

- The letter H is drawn using two bars and a horizontal line.
- The letter e uses a font that is not available on our computer, and it is drawn after the string "llo".

So in this example, when we extract the text, instead of "Hello" we may end up with "llo?".

For a more in-depth discussion about the problems of extracting text from PDF, please check out Chapter 18.2 of the book *iText in Action* by Bruno Lowagie, the developer of the iText library.

With all that being said, it is still possible to extract *some* text out of PDF. In our implementation we use the PDFBox library, which does an admirable job of extracting text and keep them in order.

The text extracted from a page is stored in a file with the following naming convention:

    Page_####.txt

where the 4-digit number is the page number.

OCR Software Evaluation for the FDsys PdfImage Project

Executive Summary

One of the goals of the FDsys PdfImage Project is to OCR images contained in PDF files. For this purpose we have conducted a thorough evaluation of 40 commercial and open source OCR products currently available on the market.

The evaluation process consists of three rounds.

In the first round, 26 candidate products were eliminated from further consideration based on the following criteria:

- The product is focused on a specific type of data (typically form data).
- The product is focused on a non-English language.
- The product lacks documentation or support.
- The product does not provide an API.

In the second round, we evaluated the OCR accuracy of the remaining 14 candidates on two sample images. Both sample images contain pure text, but one sample has much higher image quality than the other. Out of the 14 candidates, 10 were eliminated in this round due to runtime errors or poor OCR accuracy (i.e. less than 90%) – note that for the OCRed text to have a high degree of readability, accuracy rates must be in the high ninetieth percentile. The four candidates that move to the next round are:

- Nuance OmniPage
- ABBYY FineReader
- Cvision Maestro
- Transym OCR (TOCR)

In the third round, we evaluated the remaining four candidates against a test sample set that consisted of 19 sample images. 11 of these samples represent various types of images such as images that contain pure text, tables, charts, graphics, or maps. The other eight images were provided by GPO and were used in their previous testing.

The results show that both Nuance OmniPage and ABBY FineReader consistently achieve high accuracy on the samples that contain significant amount of text, while Cvision Maestro and Transym OCR perform poorly on some of these samples. Comparing to OmniPage, FineReader has a significant advantage in its Recognition Server technology. ABBYY Recognition Server can OCR the images contained in PDF files, then add the OCRed text back to the PDF files as hidden layers so the images in the PDF files become “searchable”. Using ABBYY Recognition Server not only will make the OCR step of the process efficient and scalable, but also will drastically speed up the development of the PdfImage Project because it is no longer necessary to write Java code to drive a C/C++ OCR engine.

Based on these findings, we recommend Abby FineReader Recognition Server for the FDsys PdfImage Project.

Candidates

Table 1. OCR Software Candidates

Name	API	License(s)	Online	Windows	Mac OS X	Linux	Unix
ABBYY FineReader	Yes	Commercial	No	Yes	Yes	Yes	No
Accusoft.	Yes	Commercial	No	Yes	Yes	No	No
Alt-N Technologies'	No	Commercial	No	Yes	No	No	No
AnyDoc	?	Commercial	?	?	?	?	?
Asprise	Yes	Commercial	No	Yes	Yes	Yes	Yes
Brainware	No	Commercial	No	Yes	No	No	No
EMC Captiva	Yes	Commercial	No	Yes	No	No	No
Clara OCR v1.1	GPL	GPL	No	No	No	Yes	Yes
Computhink's ViewWise	No	Commercial	No	Yes	No	No	No
CuneiForm	DK	BSD	No	Yes	Yes	Yes	No
CVision Maestro	Yes	Commercial	No	Yes	No	No	No
DTK	Yes	Commercial	Yes	Yes	No	No	No
ExperVision TypeReader & RTK	Yes	Commercial	No	Yes	Yes	Yes	Yes
Free OCR	Yes	Freeware	Yes	No	No	No	No
Form Storm	Yes	Commercial	No	?	?	?	?
GOCR	Yes	GPL	No	No	No	Yes	Yes
Google Docs	No	?	Yes	Yes	Yes	Yes	Yes
HOCR	Yes	GPL	No	No	No	Yes	No
img2txt.ru	No	Commercial	Yes	No	No	No	No
Lead Technologies	Yes	Commercial	No	Yes	No	No	No
Microsoft Office Document Imaging	No	Commercial	No	Yes	Yes	No	No
Microsoft Office OneNote 2007	No	Commercial	No	Yes	No	No	No
MoreData	No	Freeware	No	Yes	No	No	No
NEOPTEC	No	Commercial	No	Yes	Yes	No	No
NovoDynamics VERUS	No	Commercial	?	?	?	?	?
Ocrad	GPL	GPL	No	No	No	Yes	Yes
Ocre	GPL	GPL	No	No	No	Yes	Yes
OCRopus	Yes	Apache	No	No	No	Yes	No
OCR Terminal	Yes	Freeware	Yes	Yes	Yes	Yes	Yes
Nuance OmniPage	Yes	Commercial	No	Yes	Yes	No	No
OnlineOCR.net	No	Freeware	Yes	Yes	Yes	Yes	Yes
Recogniform Technologies	Yes	Commercial	No	Yes	No	No	No
Readiris	Yes	Commercial	No	Yes	Yes	No	No
ReadSoft	No	Commercial	No	Yes	No	No	No
Scantron Cognition	No	Commercial	No	Yes	No	No	No
SimpleOCR	Yes	Freeware	No	?	?	?	?
SimpleSoftware	Yes	Freeware	No	?	?	?	?
SmartScore	No	Commercial	No	Yes	Yes	No	No
Tesseract	Yes	Apache	No	Yes	Yes	Yes	No
Transym OCR	Yes	Commercial	No	Yes	No	No	No

Table 1 shows the initial pool of the OCR software candidates, which includes virtually all the products currently available on the market. The question marks in the table mean that we were not able to find publicly available information regarding that product.

Round 1: Preliminary Selection

We conducted some preliminary investigation of the 40 OCR software candidate listed in Table 1 based on publicly available information such as product descriptions and online documentation. We eliminated some of the candidates from further consideration based on the following criteria:

- The product is focused on a specific type of data (typically form data).
- The product is focused on a non-English language.
- The product lacks documentation or support.
- The product does not provide an API.

Table 2 summarizes the 26 candidates eliminated in this round and the reasons why they were eliminated.

Table 2. Candidates Eliminated in Round 1

Name	Reason
Accusoft.	Forms focus
Alt-N Technologies'	No API
AnyDoc	Forms focus
Brainware	No API
Computhink's ViewWise	Data capture focus
DTK	Only 4 font types
ExperVision TypeReader & RTK	Poor support
Free OCR	Tesseract based; not mature
Form Storm	Forms focus
Google Docs	Online only
HOCR	Hebrew focus
img2txt.ru	No support (one page Russian site)
Microsoft Office Document Imaging	Require MS Office & VB
Microsoft Office OneNote 2007	Require MS Office & VB
MoreData	No support
NEOPTEC	Forms focus
NovoDynamics VERUS	Middle Eastern languages focus
Ocre	Spanish focus
OCR Terminal	Online only
OnlineOCR.net	Online only
Recogniform Technologies	Forms focus
Readiris	No support
ReadSoft	Forms focus
Scantron Cognition	Forms focus
SimpleSoftware	Forms focus
SmartScore	Music focus

Round 2: OCR Accuracy of High Quality and Low Quality Text

In this round we evaluate the OCR accuracy of the remaining 14 candidates using two sample images that contain pure text. We refer to the sample that has higher image quality as the "High Quality Text Sample", and the other sample the "Low Quality Text Sample". Table 3 shows the OCR accuracy percentages of each candidate on these two samples.

Table 3. OCR Accuracy on Two Text Samples

Name	High Quality Text	Low Quality Text	Notes
ABBYY FineReader	100%	99%	
Asprise	71%	<70%	
EMC Captiva	-	-	Not able to obtain an evaluation copy
Clara OCR v1.1	-	-	Segmentation fault during testing
CuneiForm	-	-	Program locked up during testing
CVision Maestro	98%	80%	
GOCR	<70%	<70%	
Lead Technologies	97%	90%	
Ocrad	<70%	<70%	
OCRopus	-	-	Failed to OCR the sample images
Nuance OmniPage	100%	99%	
SimpleOCR	<70%	<70%	
Tesseract	97%	82%	
Transym OCR	98%	98%	

Note that for the OCRed text to have a high degree of readability, accuracy rates must be in the high ninetieth percentile. As shown in Table 3, only ABBYY FineReader, Nuance OmniPage, and Transym OCR achieved 98% or above accuracy rates for both test samples. These three candidates, plus CVision Maestro, became the finalists in the last round of accuracy testing. The reason to include CVision Maestro is that it provides a server product that is similar to ABBYY Recognition Server, which is highly desirable for this project. We will discuss ABBYY Recognition Server in more details in the *Non-Accuracy Factors* section.

Round 3: OCR Accuracy of the Full Test Set

In this round we evaluate the four final candidates using a test sample set that consists of 19 images. 11 of these images are selected to represent a wide range of images types:

- High quality image containing primarily text
- Low quality image containing primarily text
- Text in multiple fonts
- Text with graphical formatting
- White text on black background
- Graphics with text overlay
- Tables (without borders)
- Flow chart
- Bar chart with labels
- Pie chart with labels
- Map (rotated)

The other 8 sample images in the test dataset are provided by GPO. These images are characteristic of the volumes of text that are the product of Congress. GPO previously ran test on these images and wanted to compare the results to this set of test.

Table 4. OCR Accuracy of the Four Finalists

No.	Image	OmniPage	FineReader	Maestro	TOCR
1	0001r001	99.66%	100.00%	100.00%	99.76%
2	0002r200	100.00%	100.00%	100.00%	100.00%
3	0003r003	99.01%	99.01%	97.85%	<50%
4	0005r005	99.84%	99.40%	99.80%	87.10%
5	0010r010	99.72%	99.57%	99.48%	86.02%
6	06780646	99.94%	98.81%	99.83%	<50%
7	19130047	99.96%	97.27%	99.75%	97.84%
8	19150049	100.00%	99.19%	99.57%	95.93%
9	High quality text	99.60%	99.87%	97.67%	97.54%
10	Low quality text	98.59%	99.44%	79.68%	98.17%
11	Multiple fonts	88.21%	91.73%	77.64%	92.65%
12	Black background	99.50%	99.50%	100.00%	99.50%
13	Table without borders	95.69%	91.65%	96.84%	99.82%
14	Text with graphical formatting	93.71%	98.26%	69.52%	77.98%
15	Graphic with text overlay	93.05%	70.04%	91.15%	83.76%
16	Flow chart	91.09%	95.81%	93.29%	78.44%
17	Labeled bar chart	86.00%	58.67%	80.00%	67.33%
18	Labeled pie chart	57.25%	79.00%	61.25%	74.25%
19	Rotated map	54.09%	51.78%	50.53%	95.91%

The accuracy results of the four finalists on the test dataset are shown in Table 4. To better understand these results, we can roughly divide the test samples into two categories: images that contain primarily text (samples 1 to 14 in Table 4), and images that contain primarily graphics (samples 15 to 19 in Table 4).

For images containing primarily text, OmniPage and FineReader clearly outperform Maestro and TOCR. For OmniPage and FineReader, the worst result in this category is 88.21% on the sample with multiple fonts by OmniPage. On other hand, both Maestro and TOCR dip below 80% (and in the case of TOCR, 50%) on multiple occasions. Based on these results and the assumption that the majority of the images to be OCRed contain primarily text, we can remove Maestro and TOCR from further consideration.

For images containing primarily graphics, the performance of OmniPage and FineReader are comparable with OmniPage being slightly better.

Overall, OmniPage and FineReader are the winners of this round.

Non-Accuracy Factors

Table 5. ABBYY FineReader vs. Nuance OmniPage

Name	API	Windows	Mac OS X	Linux	Unix	Speed
FineReader	Yes	Yes	Yes	Yes	No	Less than 1 minute per sample
OmniPage	Yes	Yes	Yes	No	No	Less than 1 minute per sample

Table 5 summarizes some non-accuracy factors of FineReader and OmniPage. We can see that again they are very similar, except that FineReader supports Linux while OmniPage does not. Since Linux is the preferred target platform, this gives FineReader a significant advantage over OmniPage.

There are two other factors that are in favor of FineReader. First and foremost, ABBYY provides a server version of FineReader called ABBYY Recognition Server. This product can OCR the images contained in PDF files, then add the OCRed text back to the PDF files as hidden layers so the images in the PDF files become “searchable”. ABBYY Recognition Server is designed to be scalable and is well suited for a production environment where large number of PDF files need to be processed efficiently. Furthermore, using ABBYY Recognition Server will drastically speed up the development of the PdfImage Project because it is no longer necessary to write Java code to drive a C/C++ OCR engine. Secondly, we are informed that ABBYY Recognition Server is already being used by GPO, so choosing ABBYY Recognition Server over equivalent products by other vendors is likely to incur lower software and maintenance costs.

We did notice during our testing that OmniPage did a better job to preserve the original layout and formatting in the OCRed text.

Final Recommendation

Based on the accuracy results and the non-accuracy factors, we recommend ABBY Recognition Server for the FDsys PdfImage Project.

File	NumPages	NumImages	NumText	ExtractT	OCRT	
BUDGET-2010-BUD	146	9	16	6	221	
BUDGET-2010-TRANSMITTAL	3	3	3	0	8	
CDIR-2000-10-01-CAPITOL	21	5	11	4	49	
CDIR-2000-10-01-STATEMAP-CA	1	1	1	0	3	
CDIR-2000-10-01-STATEMAP-VI	1	1	1	0	2	
CDIR-2000-10-01	1213	62	73	44	2475	
CDOC-111hdoc11	1081	1079	1080	4	15791	
CDOC-111hdoc17-pt1	1548	1546	1547	1999	9131	
CDOC-111hdoc17-pt2	1345	1343	1344	1685	19688	
CDOC-111hdoc7	38	35	36	1	459	
CFR-2009-title49-vol7	666	129	176	11	1298	
CHRG-106shrg10636166	78	78	78	0	270	
CHRG-110hhrg11046861	169	83	97	4	453	
CHRG-111hhrg11147258	125	75	78	2	466	
CHRG-111hhrg11148055	74	12	17	0	148	
CHRG-111hhrg11151898	69	33	46	1	197	
CHRG-111shrg37-pt1	58	2	4	0	82	
CHRG-111shrg370	193	161	162	5	687	
CPRT-108SPRT90655	134	10	14	0	201	
CPRT-110HPRT44807-Part 3	29	4	6	2	40	
CPRT-110HPRT44807-Part 4	518	243	313	8	1436	
CPRT-110HPRT44807-Part 5	132	13	14	8	227	
CPRT-110HPRT44807-Part 6	113	24	25	14	253	
CPRT-111HPRT54329	366	285	301	26	2634	
CPRT-111SPRT47215	66	2	6	2	78	
CPRT-111SPRT48170	269	263	264	236	1295	
CPRT-111SPRT51207	20	2	3	0	32	
CPRT-111SPRT51233	22	0	0	0		0
CREC-1996-01-22	538	132	168	275	3136	
CRECB-2001-pt20-issue-2001-12-19	706	252	275	712	2755	
CRECB-2001-pt20	1284	262	287	753	4767	
CRPT-108hrpt490-pt1	45	8	12	0	101	
CRPT-108hrpt490-pt2.pd		3	1	3	0	5
CRPT-110srpt251	234	54	80	3	809	
ECONI-1910-02-Pg1	1	1	1	0	11	
ECONI-1910-02-Pg37	1	1	1	0	8	
ECONI-1995-04-Pg1	1	1	1	0	10	
ECONI-1995-04-Pg13	1	1	1	0	16	
ERP-1996-other-2	3	3	2	0	3	
ERP-2009-chapter1	29	10	20	36	73	
ERP-2009-chapter2	36	9	20	32	74	
ERP-2009-chapter3	30	6	15	19	57	
ERP-2009-chapter4	24	5	12	22	45	
ERP-2009-chapter5	23	4	11	14	39	
ERP-2009-chapter6	21	4	10	14	38	
ERP-2009-chapter7	20	4	9	14	40	

ERP-2009-chapter8	22	4	10	14	38
ERP-2009-chapter9	22	4	11	13	37
ERP-2009-frontmatter	15	4	7	0	13
ERP-2009	419	55	127	6	914
ERP-2010-chapter10	26	10	3	0	35
ERP-2010-chapter3	31	1	3	0	44
ERP-2010-chapter9	24	1	3	0	34
ERP-2010-frontmatter	13	6	6	0	17
FR-1996-02-12	243	1	3	5	536
FR-2008-01-29	347	2	6	0	834
GAOREPORTS-GAO-08-384	32	2	6	0	47
GAOREPORTS-GAO-08-685T	17	2	4	1	34
GAOREPORTS-GAO-08-876R	24	6	11	0	61
PPP-2004-book1-frontmatter-pgiii	12	1	3	0	11
STATUTE-117	3158	14	24	2	6718
STATUTE-118-FrontMatter-Pgi	31	1	2	0	34
WCPD-1998-02-09-FrontMatter	3	1	2	0	5