Text2PTO Proof of Concept White Paper

Version 1.0

24 March 2015

American Environmental and Engineering Consultants (AEEC, LLC)
1925 Ballenger Ave, Suite 450,
Alexandria, VA 22314
http://www.americanconsultants.com
Phone: (703) 317-0800

Table of Contents

List of Figures

List of Tables

1 Introduction

This white paper presents details about a Text2PTO prototype for extracting text, layout, and formatting information from PDF files that have text behind them. A proof-of-concept (POC) was conducted to design a solution that could accept incoming PDF files and extract text content along with formatting and layout information. This document is presented by AEEC's Application Architecture Software Engineering Team (AASET) to the United States Patent and Trademark Office (USPTO).

1.1 BACKGROUND

USPTO currently accepts patent applications through its Electronic Filing System (EFS). EFS accepts PDF documents during application submission. Applicants can use various tools to create PDFs for EFS submission. More than 45% of these submitted PDFs have text behind them. Due to the differences in the COTS or open source tools used by the applicant to generate the PDFs, the format and structure of the PDFs differ. Currently, USPTO relies on OCR to extract text from TIFF representations of these submitted PDFs. The OCR-extracted text and layout information is used to generate XML4IP documents.

2 Problem and Scope

USPTO's current approach of using OCR to extract text does not produce fully reliable results. There would be great advantages, including an increase in reliability, if it were possible to use the applicant submitted text contained within the PDFs for generating XML4IP documents.

2.1 Requirements for Prototype

USPTO requirements for the proposed Text2PTO prototype are as follows:

- The system shall support monitoring incoming files to the INPUT Folder
- The system shall check if the given input file is a scanned or text based PDF file
- The system shall capture the file metadata that can be used to analyze file formats and identify their success rate for extracting text
- The system shall handle exceptions and capture the cause of exception
- The system shall extract the text and formatting information from the PDF file and generate a valid XML file
- The system shall support batch processing

2.2 Scope and Assumptions

- All PDF files to be extracted will be present in a designated folder.
- Prototype testing will be done on samples provided by the USPTO team.
- Performance and scalability is not a primary concern while designing the prototype
- There is no pre-defined format for the generated output XML. The prototype team can define their own format.
- The prototype team is not responsible for generating XML files in XML4IP format.

2.3 High Level Objective

Research the PDFs from EFSWeb submissions that contain text to determine if the text in the PDFs can be extracted to an XML format that may eventually be converted to XML4IP documents.

Conduct market research and develop a prototype to identify a tool that can extract text, format, and layout information consistently and reliably from PDFs that have text behind them.

3 Solution Analysis

3.1 PDF Brief Introduction

A PDF file encapsulates a complete description of a fixed-layout flat document, including the text, fonts, graphics, and other information needed to display it. The appearance of everything that each page contains is completely specified. The structure of a PDF file does not match the structure of the PDF document it describes. PDF documents are display-oriented and the specification is not built to address text extraction concerns. The underlying structure of a PDF is multi-layered and very complex. Elaborating on the intricacies of extracting text from a PDF is outside the scope of this document. However, additional details can be found by referring to the following articles:

http://partners.adobe.com/public/developer/tips/topic_tip31.html

http://www.planetpdf.com/developer/article.asp?ContentID=navigating_the_internal_struct&page=0

3.2 Selecting a Tool

As discussed in the previous section, text / format extraction from a PDF is a complex task. There are several tools in the market that have sought to solve this problem with varying degrees of success. Most of these tools do a good job extracting text from PDFs, but they lack reliability and consistency in extracting the document's format, layout, and structures. Developing a tool from scratch to extract text, format, and layout information from PDF documents would be a very complex task, and may not produce an acceptable output. Therefore, this prototype effort focuses instead on identifying a tool that could get as close as possible to our objective.

3.3 Solution Overview

We followed a 3-step process to identify and develop a solution:

1. Identify format, layout, and structural elements of PDF documents to be extracted.
2. Conduct market research on tools that can extract the identified elements from PDFs.
3. Develop a Java-based solution around selected tools in order to:
    a. Integrate the tool in the USPTO environment
    b. Support batch processing
    c. Output an XML document
    d. Address shortcomings of the tool

4 Tool Selection

This section discusses the criteria and the process used to select the tool that will be used to extract text, metadata, format, and layout information from PDF files.

4.1 Criteria for Tool Selection

A following set of criteria were defined to compare and select tools for this prototype:

Extraction Capabilities: The primary criterion for tool selection is to extract text along with layout, format, and structure information. With that in mind, the following list of elements that the selected tool needed to extract from PDFs was created and includes:

- Text
- Underlines
- Strikethrough
- Bold and Italics
- Lists
- Tables
- Images
- Mathematical Formulas
- Chemical Formulas
- Indentation

Extensibility: It is very difficult for any of the tools to produce a result that exactly matches USPTO's requirement. Therefore, it is important that the selected tool provide some means of extending / customizing / refining its features.

Maturity: The tool should already be in the market and should be in use by at least a few large customers.

Integration: We should be able to integrate the tool into an existing or custom application built for this purpose. As a result, it should offer an API or a service.

Price: The overall cost of deployment for the tool needs to be considered.

Deployment Environment: The tool should be able to deploy with ease and on different environments.

Support: Support should be available in the form of a license or active user community.

4.2 First Cut

A market research study was conducted to identify tools that could be used for the purpose of this prototype. After market research, 13 tools were selected for further analysis in order to identify the best tool for the prototype. The results of this study comparing each of the 13 tools to our selection criteria are provided in **Table 1**.

Name	API/Tools	Open Source	License	Support	Environment	Features
Acrobat Pro	PDF Library SDK	No	Contact Adobe for price	Yes	Windows, UNIX	- Support for conversion to PDF/X-1a and PDF/X-3 standards - Support for Mac 64-bit platform - PDF Library SDK includes 17 completely functional solution samples - Complete documentation, including an application programming interface (API)
PDFBox	Java Library	Yes	Apache License 2.0	Its own issue tracker. Attach PDF file to get support	Windows, UNIX	More Command Line Tools
JPedal	- jPedal Java Library- Extracts text and images - PDF to HTML5 library	No	Server License Java Library--$699/per year PDF Conversion- $5000+$0.20 per page	Yes (available from IDR Solutions)	Windows, UNIX	- Available as Web Service API - Uses as Swing Application - Integration with Server Applications
jPDFText	Java Library	No	jPDFText CPU-Pair License(s) /$400	Yes (Qoppa Software)	Windows, UNIX	
AbleToExtract	PDF to HTML SDK Developer Tool	No	1 Developer License / $2,500	Yes (Investintech.com)	Windows only (standalone)	Converts PDF file into different formats Excel, MSWord etc.

Name	API/Tools	Open Source	License	Support	Environment	Features
PDF2XML	Command line tools only	Yes	GNU General Public License (GPL), v2	No	Windows, UNIX	- PDF to XML conversion - Text extraction - Vectorial instruction extraction
PoDoFo	Portable C++ library	Yes	GNU Library or Lesser General Public License version 2.0	No	All	Extracts images only. Does not support extraction of text, font and table.
PDF-Parser	Standalone PHP library http://www.pdfparser.org/	Yes	GPLv3 License	No	Requires PHP 5.3	- Extracts metadata - Extracts text from ordered pages
PDFExtract	https://github.com/CrossRef/pdfextract	Yes		No	Requires Ruby 1.9.1 and above	- Open issues with extracting references - It extracts text
veryPDF	Multiple tools available to extract the data	Yes	Commercial License	Yes	All	- User friendly tools - Supports Mac OS
PDFMiner	Command line tools and PDFMiner library supports Python	Yes	MIT/X License	No	All	- Written entirely in python - Supports various fonts - PDF to HTML Conversion - Tagged content extraction
LibreOffice	Java and C++ APIs	Yes	Mozilla Public License v2.0	Community Support & Professional support from Collabora	Windows, Mac, Linux	
PDFxStream	Java Library	No	$5000 per server	$1500 for 2 year support	Windows, Unix, Mac	- Text, Image and Form extraction - Support for Unicodes, embedded and standard fronts - Table extraction utilities - Handlers to customize output format

Table 1: Comparison of PDF Text Extraction Tools

The following five (5) tools were eliminated from further consideration because of their inherent limitations:

jPDFText

- Extracts only text
- Unable to extract font, tables, strikethrough lines, etc.

PDF2XML

- No text format information
- Does not provide Developer Libraries
- Extracts only plain text
- No professional support and community not active

PoDoFo

- Accepts only simple PDF files to extract text
- No Commercial Support available for this product

PDFExtract

- Does not extract all required format information
- Not mature enough
- No product support

PDF-Parser

- Only simple text extraction
- No product support

4.3 SECOND CUT

Each of the eight (8) remaining tools was installed in the development environment and tested against sample PDF documents that contained all the elements needed for extraction. The results of this analysis for the 8 tools are shown in **Table 2**.

Criteria	Acrobat Pro	Libre Office (SVG)	PDFxStream	PDFBox	veryPDF	AbleToExtract	JPedal	PDFMiner
Text	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Underlines	Yes	Yes	Yes	No	Capture as images	Yes	No	No
Strike-throughs	Yes	Yes	Yes	No	Capture as images	No	No	No
Bold and Italics	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Lists	No	No	No	No	Yes	Yes	No	No
Tables	Yes	Yes	Yes	No	Yes	Yes	Each cell as table	No
Images	Inconsistent	Inconsistent	Inconsistent	Inconsistent	Inconsistent	Inconsistent	No	No
Mathematical Formulas	No	No	Inconsistent	No	No	No	No	No
Chemical Formulas	No	No	No	As Images	As Images	As Images	No	No
Indentation	Yes	No	Yes	No	Yes	No	No	No

Table 2: Comparison of PDF Text Extraction Tools

Based on the above results, three (3) tools were selected for further analysis: Acrobat Pro, LibreOffice, and PDFxStream.

4.4 Final Cut

In this phase, Acrobat Pro, PDFxStream, and LibreOffice were further analyzed to determine which would be chosen as the final product. A Java application was developed to interface with these tools for further testing.

4.4.1 Acrobat Pro

Prior to this phase, 'Acrobat XI Pro' software was used to test the conversion capabilities of this tool. Acrobat also provides an SDK called 'Adobe PDF Library SDK' which purports to provide

the same programmatic capabilities of the GUI tool. Upon experimenting with the SDK, it was determined that the conversion feature is packaged into another product called 'Adobe Livecycle'. Adobe Livecycle Enterprise Suite is an enterprise document and form platform that helps to capture and process information. One of the modules in the suite is an 'Export Service' that takes PDF as an input and generates HTML.

Limitations:

- The 'Export Service' is only available for the Windows environment.
- Livecycle is a complete suite that comes with an integrated J2EE server, making deployment and maintenance more difficult.
- The 'Export Service' is a very small component of the suite and is not sold independently, resulting in a high licensing cost.
- The service is like a black box and it cannot be customized or extended.
- Results were not consistent.
- It does not provide error notifications when elements are missing in the extracted document. This forces all extracted documents to be compared manually with the original PDFs for errors.
- There is no support for extracting lists.
- Image retrieval is inconsistent and cannot retrieve vector images
- There is no support for mathematical formula retrieval.
- There is no support for chemical formula retrieval.

4.4.2 LibreOffice

LibreOffice is a complete office suite for creating documents, spreadsheets, presentations, drawings, etc. The steps for extracting from text using Libre Office are as follows:

- Open Libre Office Writer
- Use the 'Open' menu option to open up a PDF file
- Use the 'Save As' option to save the PDF file as an '.fodg' document
- The saved '.fodg' document is essentially an Open Document XML-based file format for representing graphics.

Limitations:

- The main drawback of Libre Office is the generated output format. Though '.fodg' files are XML-based, they are more suitable for representing graphics than text content. Parsing and retrieving meaningful content from '.fodg' files would be a challenging task.
- Results were not consistent.
- It does not provide error notifications when elements are missing in extracted documents. This forces all extracted documents to be compared manually with the original PDFs for errors.
- There is no support for extracting lists.
- Image retrieval is inconsistent and cannot retrieve vector images
- There is no support for mathematical formula retrieval.
- There is no support for chemical formula retrieval.

- There is no support for extracting indentations.

4.4.3 PDFxStream

PDFxStream is a Snowtide product. It is written in 100% pure Java and is developed specifically to extract text and metadata from PDF documents.

How PDFxStream works:

Figure 1 shows how PDFxStream can be integrated into our application to extract text and metadata. As shown in the figure, PDFxStream parses the PDF file and generates events. The application output handler can listen to these events and perform an appropriate action (this is similar to XML parsing using a SAX handler). PDFxStream gives the application access to content and related metadata at each stage of PDF processing. This gives the application a lot of flexibility to customize the output at each stage of processing. The application may raise errors when the tool cannot process any of the elements present in the PDF. This can prevent manual inspection of output documents.

Limitations:

- There is no support for extracting lists.
  - PDFxStream can be extended through a custom Java application to extract list information from the original PDF documents.
- Image retrieval is inconsistent and cannot retrieve vector images
  - PDFxStream partially fixed the vector image retrieval issues. PDFxStream is planning to provide a fix for vector image retrieval.
- There is no support for mathematical formula retrieval.
  - Currently PDFxStream can extract a partial set of special characters. They are planning to provide a fix for special characters. The other option would be to extract mathematic formulas as an image by extending the application.
- There is no support for chemical formula retrieval.
  - The application can be extended to extract chemical formulas as an image.

As shown above, there are possible solutions that may resolve PDFxStream's limitations to some extent by extending the application. The capability of PDFxStream to raise errors when it cannot process any of the elements will prevent the need for manual inspection of all extracted documents.

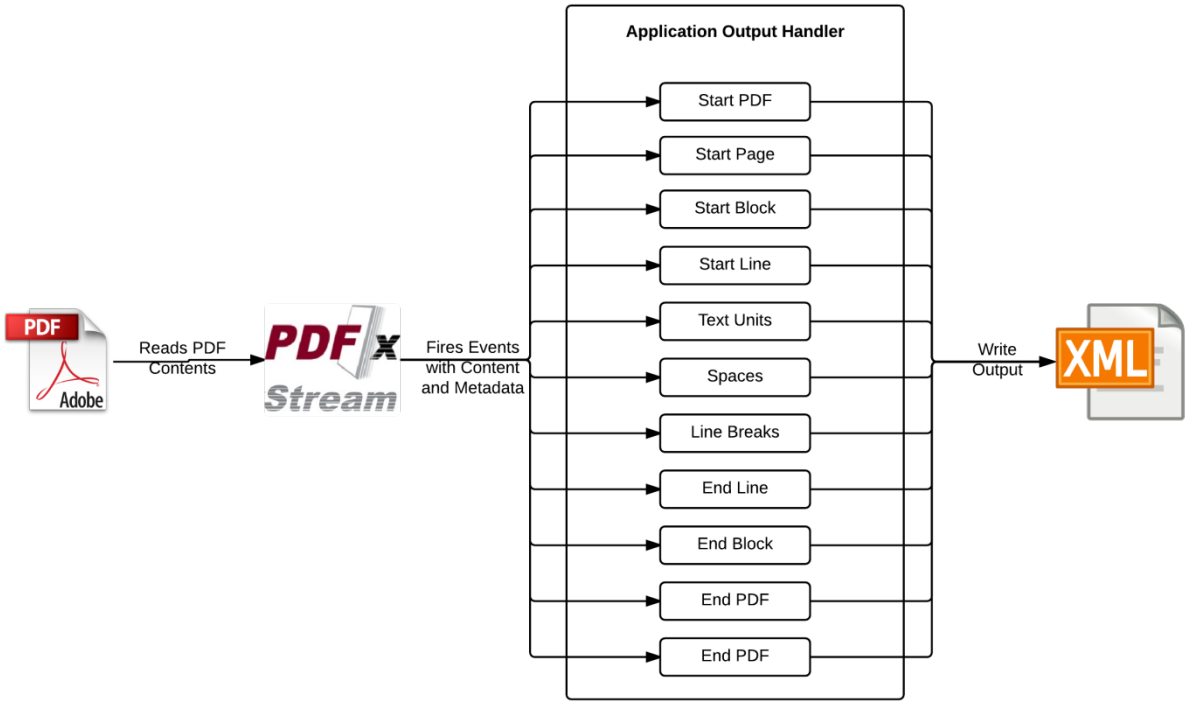

Figure 1: PDFxStream - Content Parsing

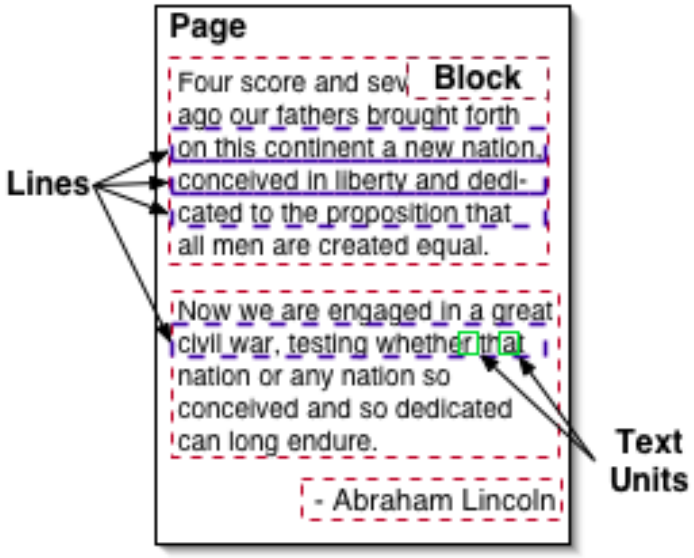

Figure 2: PDFxStream - Data Elements

4.5 RECOMMENDED PRODUCT

After analyzing the features and limitations of the 3 products, PDFxStream was chosen because it brings us closest to achieving USPTO's requirements. PDFxStream is superior for the prototype because it:

- Prevents the need for manual inspection of all output documents
- Consistently extracts elements from PDF documents
- Provides possible alternative solutions to address its limitations

The advertised list of PDFxStream features includes:

- Unicode text extraction, including support for Chinese, Japanese, and Korean (CJK) in both horizontal and vertical writing modes
- Efficiently customizing PDF text extract formatting
- Complete support for embedded and standard fonts and character encodings
- Automated layout processing, providing a traversable PDF document model including inferred block, line, column, and table structure
- Support for extracting text from "searchable image" PDFs
- Support for all varieties of rotated text
- Basic detection and inference of tabular data and a set of table-extraction utilities
- Decompression and decoding of dozens of PDF image types
- Automatic stitching of image tiles and strips

5 Prototype Design

This section talks about the design, limitations, and results of the prototype.

5.1 Components and Flow

Figure 3 shows a high level view of the various components in the prototype and flow of data.

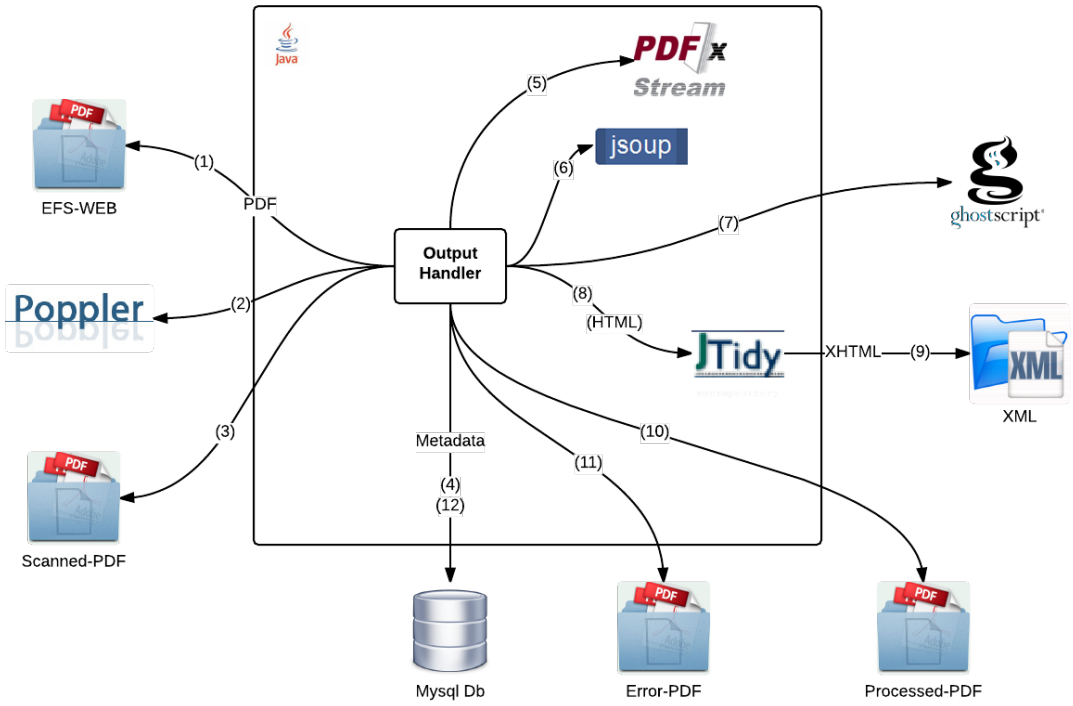

Figure 3: Component and Flow Diagram

The following is an explanation of the various steps in the process:

1. The application continuously monitors the 'EFS_WEB' folder. If a new PDF, the file is placed in the directory and picked up for processing. For prototype purposes, only one PDF file is processed at a time. If multiple PDF files are placed in the EFS-WEB folder, they are processed sequentially. This is because the trial version of 'PDFxStream' limits the application from processing files simultaneously.
2. The files being placed in the EFS-WEB folder could be one of two types:
    a. Scanned PDFs
    b. Text-based PDFs

The prototype is only concerned with PDFs that are text based and does not process scanned PDFs. To detect if a PDF is scanned or text-based, one could open the properties of the PDF file and check for the presence of fonts. If a PDF does not have any fonts, it means it does not contain any text. The application uses a third party open source tool called Poppler to verify if a PDF file has fonts.

3. Scanned PDF files (as determined in step 2) are copied from the 'EFS-Web' folder into the 'Scanned-PDF' folder.
4. During this step, metadata related to the scanned PDF (as identified in step 2) is extracted and stored in a MySQL database. The metadata includes:
    a. File name
    b. Generated By (the tool used to originally create the PDF)
    c. Scanned Flag (set to true)
    d. Created Date
5. In this step, the text-based PDF is parsed using PDFxStream. This generates a stream of events that the application can listen to.
6. This is the step where the bulk of the application logic resides. The application listens to the stream of events generated by PDFxStream and builds an HTML model using the JSoup library. The reasons for using HTML encoding are:
    a. HTML already has pre-defined elements and attributes to represent all the text and metadata that needs to be represented in the output
    b. Being such a ubiquitous language, HTML has several tools and libraries to parse it as needed. This would make it easier to later convert into XML4IP format.
7. Step 7 is an optional step and is used to process formulae. This is discussed in detail in later sections.
8. The HTML generated in step 6 is not a valid XML. In order to generate a valid XML as the final output, a library called jTidy is used. jTidy cleans up malformed and faulty HTML and generates a valid XML. The XML output can then be parsed using either an XML parser or an HTML parser in further stages.
9. The generated XML from step 8 is copied in to 'XML-Output' folder.
10. The processed PDF file is moved to the 'Processed-PDF' folder.
11. If there are any exceptions while processing a PDF file, the file is moved to the 'Error-PDF' folder.
12. During this final step, metadata related to the processed PDF is extracted and stored in a MySQL database. The metadata includes:
    a. File name
    b. Generated By (the tool originally used to create the PDF)
    c. Scanned Flag (set to false)
    d. Message (Holds the exception message in case the PDF failed to process)
    e. Created Date

5.2 Output Format

The following table presents a mapping between the elements in the PDF and the corresponding XHTML tags in the final output

PDF Element	XHTML Elements and Attributes
Textual content	
Underlines	<u>
Strikethrough	<s>
Bold	<b>
Italics	<i>
Table	<table><tr><td>
Images	<img>
Line Breaks	
Page	
Formula	<img class=”formula”>

Table 3: Mapping between PDF and XHTML Elements

5.3 Custom Processing

5.3.1 Image Processing

Support for image extraction has only recently been added in PDFxStream. PDFxStream identifies all images and places them at the start of the page. It does not provide events for identifying images in the order they appear on the page. As a result, if we do not add custom logic, all images in the final output will be placed at the beginning of the page.

As part of the additional logic:

- The application stores the images and their locations when a page starts
- While processing other elements on the page, the application compares the position of an element against the position of the stored image to determine if an image needs to be placed before or after the element.

5.3.2 Table Processing

PDFxStream has partial support for table processing. It identifies tables, the number of rows in each table, and the number of columns in each row. For symmetric tables, where each row has the same number of columns and each column has the same number of rows, this information is good enough.

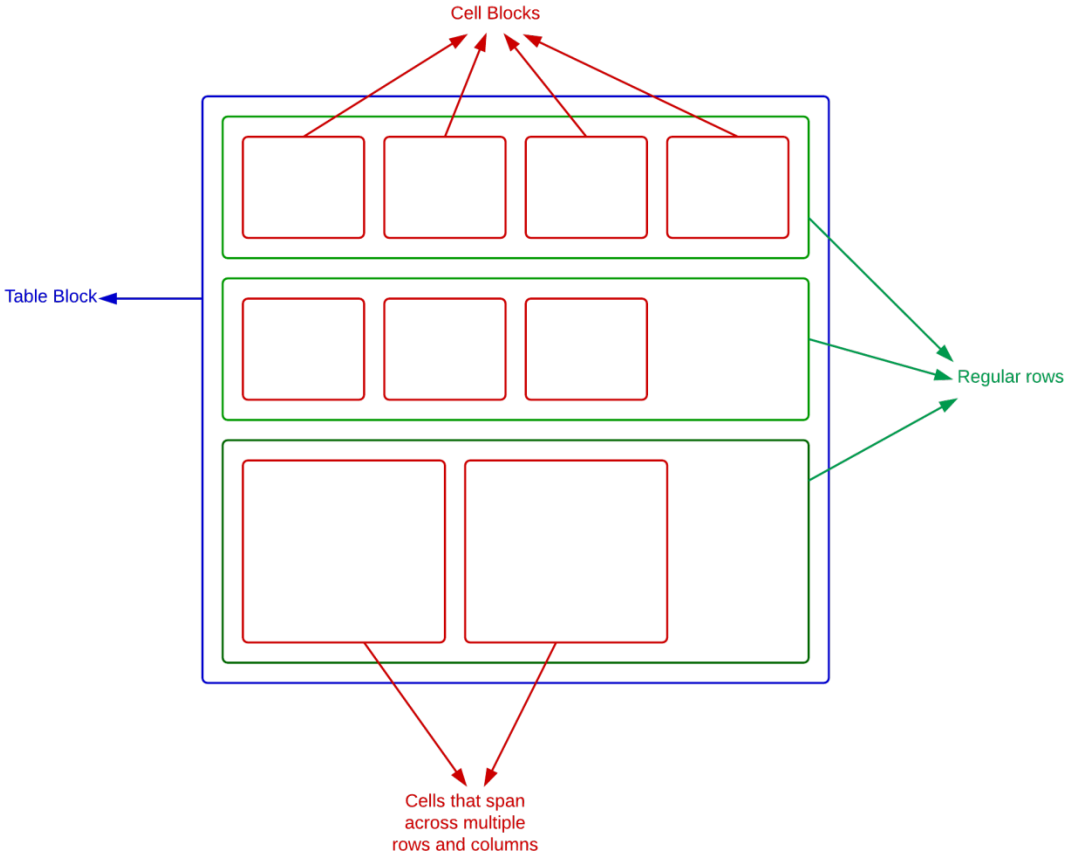

Figure 4: PDFxStream - Table Structure

However, when cells span across multiple rows or columns (as shown in **Figure 4**), the output does not match the input unless some custom logic is added.

To address this situation, the prototype has extra logic built in to determine the column span of each cell. The logic is explained below:

- For each table, identify the row with the maximum number of columns. Make a note of the cell width ('minimum cell width') and the number of columns in this row.
- While processing each row in the table, if the number of columns is less than the maximum (as determined in previous step), for each cell in the current row, calculate the width of the cell as a multiple of the 'minimum cell width'. This multiple gives the column span of the current cell.

Similar logic needs to be applied to calculate row span.

5.3.3 Formula Processing

Mathematical and Chemical formulae are challenging to extract. Formulae could be embedded in PDF documents as text using a mix of regular / special characters and vector graphics as needed to convey the information.

It is challenging to identify formulae because the formula can:

- Span multiple lines
- Have special Unicode characters (which cannot be consistently identified)
- Have graphics

In default mode, the application processes formulae as regular text. As a result, the output does not identify the formulae with any special tags and may sometimes look like garbage text.

The application has to be run by setting a system property called '-Dmode=line' to identify formulae. The following logic is used to identify a formula and extract it when this mode is used. This approach is neither consistent nor complete. However, it is something that could be improved upon in future iterations to achieve consistency and reliability.

- While processing each line, determine the number of special characters in the line. For the purpose of the prototype, any Unicode character with value greater than 256 is considered a special character. (This assumption may not be correct and what should constitute a special character is debatable, but the logic used to identify whether a particular character is special is abstracted and could be adjusted as needed.)
- If the number of special characters in a line compared to the total number of characters in the line is greater than a certain percentage (in our case this percentage is set to 10%), the line is marked as being part of a formula.
- The above logic is applied for successive lines until a non-formula line is detected. At this point, all previous lines are considered part of one single formula.
- Once a formula is identified, the coordinates of the bounding box for all combined lines are calculated.
- The bounding box is then extracted as an image using Ghostscript.
- The extracted image is written to the XHTML output using Base64 encoding and tagged with a class called 'formula' as shown below:
    - <img class='formula' src='…' ></img>

The following table shows snapshots taken under 3 different scenarios:

- Img1: Snapshot from original PDF
- Img2: Shows the output if no processing is applied for formulae
- Img3: Shows the output when a formula is detected and extracted as an image

Img1	$S_{j,K} \equiv max_{k\in[1,K]} \sum_{i=1}^{j-1} x_{i,k}\, v_{(i)} + \sum_{i=j}^{K} x_{i,k}\, v_{(i+1)}$, wherein $\mathbf{x_{j,k}}$ represents a measure
Img2	E ∑E E? Ú E∑EwE ≡ E EEo∈E>Ú,E?w E,EoE:; E,EoE: ,;wherein **x** represents a measure E,Ew E @ Ú @E > Ú **j,k**
Img3	$S_{j,K} \equiv max_{k\in[1,K]} \sum_{i=1}^{j-1} x_{i,k}\, v_{(i)} + \sum_{i=j}^{K} x_{i,k}\, v_{(i+1)}$, wherein $\mathbf{x_{j,k}}$ represents a measure

Table 4: Formula Extraction Comparison

5.4 Exception Handling

Exceptions could happen in the application for several reasons:

- The PDF document is corrupted
- When decrypting an encrypted PDF document
- When the database server is down
- Some unexpected error in code

All the above exceptions and any other unexpected errors are handled by the system and appropriately logged for further investigation. Furthermore, the PDF that failed processing is moved to the error folder.

5.5 Performance

The primary focus of the prototype was to extract data consistently. There was no effort spent on increasing the performance of the system. Also, the trial version provided by PDFxStream allows only synchronous processing, thus preventing us from building a prototype that could be used for benchmarking.

5.6 XML4IP Format

The goal of the prototype was to generate a standard XML document, which could be used in further stages to generate an XML4IP compliant document. In this regard, the prototype generates XHTML output, which is a standard format.

5.7 RESULTS

For testing the prototype, 150 client-provided PDF files were used. The prototype was used to extract XML from all 150 files. The extracted content was manually compared to the original PDF files. Below is a summary of the findings:

Sample Files Tested	150 files
Successful	121 files
Failed	29 files

Table 5: Test Results

Observations:

A failed file is one in which none of the content was extracted or part of the content was not properly extracted.

It should be noted that the 121 successful files were mostly text-based with very limited special characters.

The 29 files that failed had a combination of:

- Special characters
- Vector graphics
- Math and chemical formulae

5.8 LIMITATIONS

The following are some of the limitations of the prototype:

Vector Graphics

Vector Graphics is the use of geometrical shapes (such as points, lines, curves and shapes) to represent images in computer graphics. As of this writing, PDFxStream can extract regular images, but cannot extract vector graphics consistently. Extraction of vector graphics is a complex process and requires advanced processing to extract individual images and stitch them together. According to the team at PDFxStream, they are working on a solution for extracting vector graphics and plan to release a patch in the first quarter of 2015.

Special Characters

PDFxStream extracts the Unicodes that correspond to each of the characters found in a PDF. These Unicodes represent text in the extracted output. However, PDFxStream fails to retrieve the Unicodes consistently for special characters. This could be because the PDF uses:

- Character sets which map to Unicodes from private user area

http://en.wikipedia.org/wiki/Private_Use_Areas

- A custom character set and does not provide mapping between the character codes and Unicodes
  http://blogs.adobe.com/insidepdf/2008/07/text_content_in_pdf_files.html

At this point, it is not clear if there is a resolution for the two scenarios above. The PDFxStream team has been notified about this and they are looking into it.

Math and Chemical Formulae

Though the prototype attempted to detect and extract math and chemical formulae, the solution is not complete. Please refer to section 5.3.3 for details.

List Processing

List detection is not built into PDFxStream and there are no indications it will be supported in the future. This needs to be custom built into the application.

5.9 CONCLUSION

Although the presented solution has some limitations, it takes the right approach towards achieving the objectives. Limitations may be addressed in the long term by extending the Java application and using future releases of PDFxStream to customize / enhance the product to suit USPTO's needs.

Recommendations:

- Obtain support from PDFxStream to fix issues with Vector Images and Special Characters
- Implement additional logic to handle Chemical / Math formula
- Implement custom logic to extract List and Superscript / Subscript information