

**CONTENT EXTRACTION FROM PDF INVOICES
ON BUSINESS DOCUMENT ARCHIVES**

R.M.C.V. Bandara

168208N

Master of Science in Computer Science

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

January 2020

**CONTENT EXTRACTION FROM PDF INVOICES
ON BUSINESS DOCUMENT ARCHIVES**

R.M.C.V. Bandara

168208N

Thesis submitted in partial fulfilment of the requirements for the
Degree Master of Science in Computer Science

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

January 2020

DECLARATION

I declare that this is my own work and this thesis does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my thesis, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works.

.....

R.M.C.V. Bandara

.....

Date

The above candidate has carried out research for the Master of Science in Computer Science thesis under my supervision.

.....

Dr. Indika Perera

.....

Date

Abstract

Archiving documents is a crucial part on information management, and it will give an organization better control over their information processes. When a business expands, more documents will be produced, and it needs to be carefully handled and tracked to make good use of. Output management systems that are working with ERP systems contains thousands of business documents and Portable document format (PDF) is the common output format for these types of documents. These systems need to execute documents search operations frequently. PDF documents Indexing is a critical part in this context. It will boost document search engine efficiency by cutting search space. Content extraction from PDF documents goes a step further and it will allow more structured search queries.

Extracting the document content from a PDF file is a very important. But this is a very challenging task because PDF is a layout-based format that defines the fonts and locations of the individual character as opposed to the semantic units of the text and their role within the document. In this research I have developed a technique to extract content from a PDF file. We can use it for allow more structured search queries on large document archives in output management systems typically work with world leading ERP systems.

On this research mainly considered on four aspects which are correctly identifying words, word order on a paragraph, clear separation of paragraph boundaries and semantic roles of each word. After extracting content from the PDF file, extracted texts content written to an xml document. XML file contains tags to recognize the pages and rotation angle and number of images on each page. Sample set of PDF invoices extracted and calculated the extracted word percentage to evaluate the accuracy of this technique. This tool hits 94.27% accuracy rate according to the results.

ACKNOWLEDGEMENT

First and foremost, I am deeply grateful for the continuous support, insight, and patience of my supervisor, Dr. Indika Perera: without his invaluable support, this thesis would not have been completed.

I thank Mr. Ishara Yatawara - software architect at Creative Software, who provided insight and expertise that greatly assisted the research.

Finally, I present my appreciation to my family and my friends who were behind me, encouraging and directing me towards the success of my project.

TABLE OF CONTENTS

DECLARATION.....	i
Abstract.....	ii
ACKNOWLEDGEMENT	iii
TABLE OF CONTENTS.....	iv
LIST OF FIGURES	vi
LIST OF TABLES.....	viii
1. INTRODUCTION.....	1
1.1 Overview.....	1
1.2 Problem.....	3
1.3 Motivation.....	4
1.4 Objective	5
2. LITERATURE REVIEW	6
2.1 Automatic indexing of PDF documents with ontologies.....	6
2.2 Automatic Indexing of Scanned Documents - a Layout-based Approach.....	8
2.3 Content Extraction of PDF Documents	10
2.4 Diagram Extraction of PDF documents.....	14
2.5 Benchmark and Evaluation for Text Extraction from PDF	15
3. METHODOLOGY	19
3.1 PDF Components.....	19
3.1.1 Objects	22
3.1.2 File Structure.....	29
3.1.3 Document Structure.....	32
3.1.4 Content Stream.....	34
3.2 Text in a PDF file	34

3.3	Graphical effect on Text	36
3.4	Text state parameters and operators	38
3.5	Text Objects	42
3.6	Extraction of text content.....	43
4	IMPLEMENTATION.....	45
4.1	High level architecture.....	45
4.2	Getting the last cross reference table (xref) offset.....	46
4.3	Creating the cross-reference table	47
4.4	Finding the root and creating the page tree.....	47
4.5	Tokenizing	48
4.6	Token Handling.....	51
4.7	Object Builder	51
4.8	Reading the content objects	52
4.9	Character Mapping	52
4.10	Character Mapping	54
4.11	Creating Text Rendering Matrix (TRM)	54
4.12	Decomposing Text Rendering Matrix	55
4.13	Reading the virtual coordinate plane	55
4.14	Extracting text to an xml document.....	55
5	RESULTS AND EVALUATION	56
6	CONCLUSION	65
6.1	Summary	65
6.2	Future Works.....	68
6.3	Limitations	69
7	REFERENCES.....	70
	APPENDIX	74

LIST OF FIGURES

Figure 1: A sample business document	2
Figure 2: Typical page from a technical manual after AIDAS analysed it.....	7
Figure 3: Same template and nearly constant positions of index data.....	9
Figure 4: Extraction document through the information extraction system.	10
Figure 5: overview of the processing pipeline	11
Figure 6: System processing steps.....	12
Figure 7: The flowchart for text segmentation.....	13
Figure 8: An 8x8 SPAS structure for spatial indexing	15
Figure 9: Output file with 3 paragraphs and ground truth file with 1 paragraph	18
Figure 10: three assignments to evaluation criteria in order to assess O against G ...	18
Figure 11: PDF Components.....	19
Figure 12: Initial structure of a PDF document.....	29
Figure 13: Structure of a PDF document.	32
Figure 14: Glyphs painted in 50% gray	37
Figure 15: glyph outline treated as stroked path	38
Figure 16: Character spacing in horizontal writing	40
Figure 17: Word spacing in horizontal writing	40
Figure 18: Horizontal scaling	40
Figure 19: Leading.....	41
Figure 20: Text Rising	42
Figure 21: High level architecture.....	45
Figure 22: Flow Diagram.....	45
Figure 23: Getting the last cross reference table offset	46
Figure 24: Page Hierarchy	47
Figure 25: Tokenizers	48
Figure 26: Tokenizes the xref objects.....	49
Figure 27: Tokenizes the objects.....	49
Figure 28: Tokenizes the decoded texts.....	50
Figure 29: Tokenizes the Character mappings.....	50

Figure 30: Decoded stream	52
Figure 31: Decoded to Unicode stream	53
Figure 32: Coordinate plane	54
Figure 33: sample input 1.....	58
Figure 34: Extracted word percentage	64

LIST OF TABLES

Table 1: White-Space characters	20
Table 2: Delimiter Characters	21
Table 3: Entries on stream dictionaries.....	27
Table 4: Entries in the file trailer dictionary.	31
Table 5: Text state parameters.....	38
Table 6: Text state operators	39
Table 7: Text rendering modes.....	41
Table 8: Latin-text encoding	44
Table 9: Extracted word percentage	60
Table 10: Words precentage vs Number of files.....	63

1. INTRODUCTION

1.1 Overview

Portable Document Format (PDF) is a file format that based on PostScript language. The significance of the PDF is it present documents that are independent from intended document reader, hardware and type of operating system. “Each PDF file encapsulates a complete description of a fixed-layout flat document, including the text, fonts, vector graphics, raster images and other information needed to display it” [1].

A PDF file is a "read only" document that cannot be altered without leaving an electronic footprint. Furthermore, they can be signed electronically. The PDF is now an open standard, maintained by the International Organization for Standardization (ISO) [2].

In early times, business communication was limited to paperwork. But now PDF documents heavily support business communication. Internal records is one of the major category on business communication. Internal record is a record maintained by a firm which consisting various documents. Almost all organization and business firms are maintaining their record in PDF format. It provides speed and effeminacy in providing information to the decision makers [3].

Most of the business applications including ERP systems generates its outputs as PDF documents. ERP systems may contain millions of PDF documents on its databases. These are can be invoices, receipts and other relevant documents.

PDF file format facilitate business applications to easily design and produce documents, data or reports from their back-end business systems.

accure
Genius Output. Simplified.
www.accure.eu

FAKTURA

sid 1(1)

Mottagare
Finnvedens Lastvagnar
Mörrums gräv
korpädalsvägen 2
374 51 ASARUM
Sverige

Ordernr
0004234653

Kund
10024

Fakturanr
000203825

Orderdatum
100519

Betalare 10024
MÖRRUMS GRÄV AB
BRÄKNEBODAVÄGEN 61-8
373 91 MÖRRUM

Fakturanr 008203825 **Ordernr** 0004234653 **Kund** 10024 **Lev nr** 900

Ert ordernr **Er referens** EC290CL/110007 **Referens** **Vår referens** H Andersson

Betalin villkor 30 dagar netto **Leveransvillkor** Ex works **Leveranssätt** 001 **Ert momsreg nr** 350373039501

RADER

Rad	Artikelnr	Pris	Kvantitet	Belopp
001	VOE9022851149 STARTER	7.168,00	1	6.451,20

MOMS

Momskod	Moms (%)	Momsgrund	Momsbelopp
01	25 % moms	25,00	6.451,20
10	25 % moms	25,00	0,00

TOTAL

ATT BETALA	SEK	8.064,00
-------------------	-----	-----------------

Direktiva anmärkningar mot denna faktura: Innehåll skall vara giltiga inom 10 dagar från fakturadatum

Huvudkontor	Kontakt	Web	Bank	Betalning
Dan Sverige AB	+46 21 600290	www.dan.se	Fig. 5877-8777	Orgnr. 350867-06
Box 67	+46 21 600291	finance@dan.se	Fig. 212-4578	VAT 1255666706
177 22		info@dan.se	SWIFT: DASAZ22X	

Figure 1: A sample business document

One of the main challenges is for PDF document management system is for querying document based on key words. Document management systems may contain some information about PDF documents but not data about its contents.

Users typically need to retrieve document archives by searching its contents. As an example, user can search by client's name, address, order details, etc. To support such an operation document management system, have to index PDF document's content. Manual document content indexing is quite expensive. So, it is better to have a tool that can extract PDF document's contents [4].

1.2 Problem

World large ERP systems like Infor, produce millions of PDF documents as its outputs. These are can be invoices, receipt, quotations or other business documents. Most of the time these documents are saved on a relational database in binary format with limited relation data like created date and time.

Since the large number of outputs generated by ERP system, managing output is an extra effort. So, These ERP systems using third party output management systems to manage those documents. Output management systems are import outputs that are generated by ERP system. Users can manage millions of output records through this system.

Most of the time users of the ERP system (or Output Management System) need to retrieve archived business documents by searching its content. Output Management Systems need to efficiently execute those queries and retrieve appropriate documents. But one of the major problems facing by output management system is to querying documents by its content. Because these systems have no idea about the contents of documents. When documents generating, ERP system may produce some indexes, but it is not enough to support advanced features in output management system.

Manually creating indexing for PDF business documents are time consuming task. When it comes to ERP system generated outputs which are included thousands of PDF documents, manual indexing is almost impossible. SO, we must find a solution for create appropriate index entries for document archives automatically.

Typically, PDF document consists with encoded binary stream. When extracting data within the PDF document we need to decode that data by considering font type, style, orientation, etc...

There are few commercial tools available for automated indexing for PDF documents. But some of them are not available on this region and others are quite expensive. It's better to have an in-house product to automatically extract PDF content and indexing appropriately.

1.3 Motivation

The current move towards a paperless office through the digitalization of existing paper records and the digital exchange of new records enables new ways of managing and storing the wealth of information in businesses. The indexing of digital and digitalised records plays a significant role in this area. Tagging a document using a predefined vocabulary allows the grouping of document sets with similar correspondence in smaller subsets. These subsets can be used to boost the efficiency of document search engines by decreasing the search space.

Extracting information is more advanced feature. To extract a particular term from a document, need more structured queries. If we are able to extract information from documents automatic document processing becomes possible, like sending documents to responsible persons.

1.4 Objective

- Extracting appropriate text content from the given PDF document.

The proposed tool can scan a given PDF document and extract its text content. This content is typically in form of encoded binary stream. This tool can identify text content in any format regardless of its font, colour, style or orientation.

Extracting text content from the PDF document is the main objective of this tool. I'm planning to use Adobe specifications for PDF documents as primary guideline for this project.

- Support for extensive querying on PDF content.

User can search a document by using key words. System can execute a query against extracted data that are stored on a database.

2. LITERATURE REVIEW

2.1 Automatic indexing of PDF documents with ontologies

A paper published by Anjo Anjewierden and Suzanne Kabel in 2001 explains how AIDAS tool splits PDF document into chunks automatically and how that chunks indexes automatically and save them on a database for reuse [5]. They have used automated document analysis combined with ontology indexing technique to divide large bodies of data into reusable chunks. Then these documents support for efficient and effective search results.

Authors distinguished four stages in document analysis and indexing process.

- i. Interpreting PDF
Source is taken to be available in PDF format.
- ii. Discovering the logical document structure
The segments to save correspond to the logical structure of the document. At this stage, in order to discover the logical structure, the layout structure is incrementally analysed.
- iii. Indexing and fragmenting the logical structure
Indexing picture and text fragments using ontologies that are created by document analysis.
- iv. Storing the document in a multi-media database
Fragments saved with their indexes saved in a multimedia database.

Documents that are to be index need to discover their “document style” and for that AIDAS can use its’ AI methods. As a first step PDF instructions transform to graphics, image objects and text. Set of objects that created from this transform together to form layout structure and that structure used by AIDAS. [6].

Transforming a set of layout objects to a one hierarchical structure will referring as a logical structure discovery. Physical structure of the document is represented by using the set of layout objects and document organization illustrate by the logical structure.

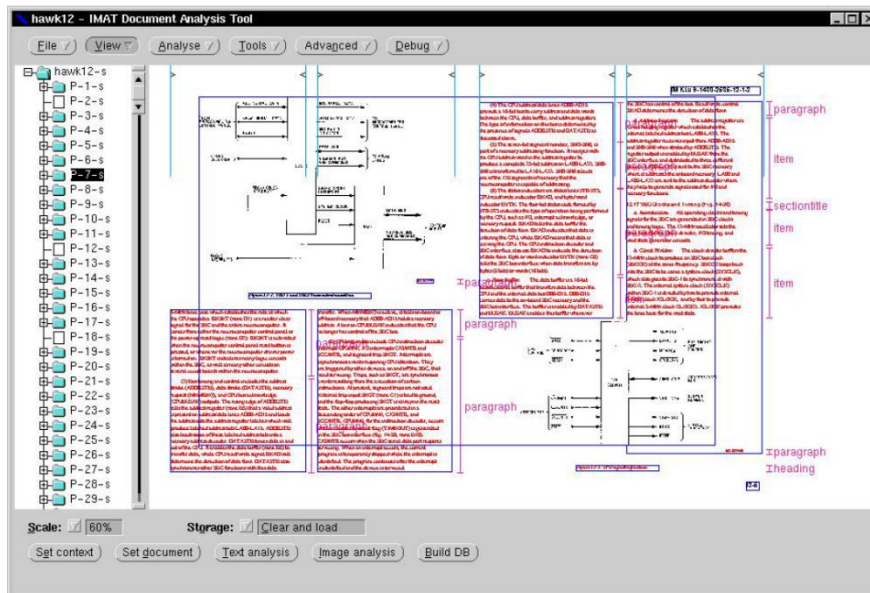


Figure 2: Typical page from a technical manual after AIDAS analysed it

Classify all layout objects is the first step AIDAS performs. This will take into account every object and adds abstractions based on rules which depend on the document's global characteristics.

The second step is to determine whether there is text in a layout object that can indicate elements of the logical structure. This can be determined by an ontology which generally maps concepts of a document onto the tokens in the text.

Take order into account is the final step. This is done by creating a grammar for all potential elements of the logical structure. Before a conclusion is drawn, the grammars limit the importance of the layout abstractions. To conclude, AIDAS uses classification and grammars to identify specific characteristics of layout objects [5].

Indexing a document using various points of view is the main concept of this paper. An ontology was established for each clearly defined perspective on the document's content:

- general and syntactical viewpoint: fragment ontology
- domain viewpoint: domain ontology
- semantic viewpoint: description ontology
- instructional viewpoint: instructional ontology

Indexing produces better search results is the expected advantage [7]. Except the domain ontology, above mentioned all ontologies are generic.

Document analysis and the development of ontologies are two existing techniques. This will be used to index source data by various perspectives. Compared to standard keyword search, they expected above two techniques will greatly increase the efficiency of the search results.

2.2 Automatic Indexing of Scanned Documents - a Layout-based Approach

Daniel Esser, Daniel Schuster and Klemens Muthmann published a paper in 2012 by describing a layout-based approach for automatic indexing of scanned documents [8]. They proposed a novel approach to capture index terms from the document by considering positioning data.

Authors proposed a new graphical method by using index data locations to get quality extraction from digitalized documents. Documents that are generated by template will identify and indexing. Clusters are created based on them and new documents are assigned to those clusters. Index data positions of cluster documents are used to capture the data from new documents [8].

In this paper authors considered about...

- A template clustering and detection method for large sets of business documents able to be trained fast and continuously by ordinary users.
- A robust and fast data extraction based on template detection that delivers even good results if data sources are potentially incorrect due to OCR errors.
- Evaluation results showing the effects of the proposed method on a large corpus of business documents.

Documents used on business domain typically have a pre-defined template. It is act as a blueprint and describe characteristics of the layout. This template is filling with relevant details [9].

Followings are three documents generated on similar template. The frequency of specific index data is almost constant, allowing for a layout-based extraction using its position.

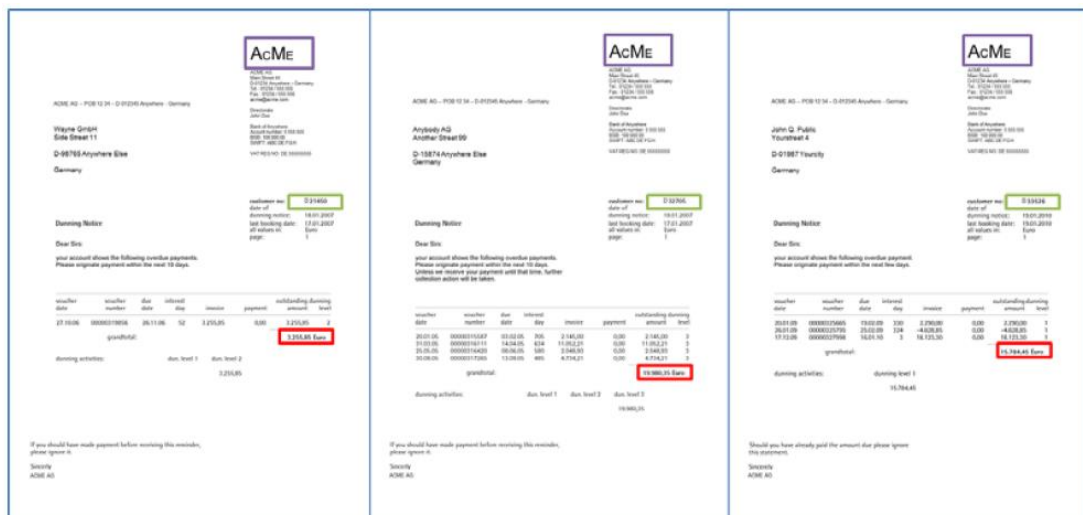


Figure 3: Same template and nearly constant positions of index data

To process a given document it will identify current documents that indexed and with same template on the archive. Then using selected document, it will generate positional extraction rules. Then it will calculate and apply to the extraction document [8].

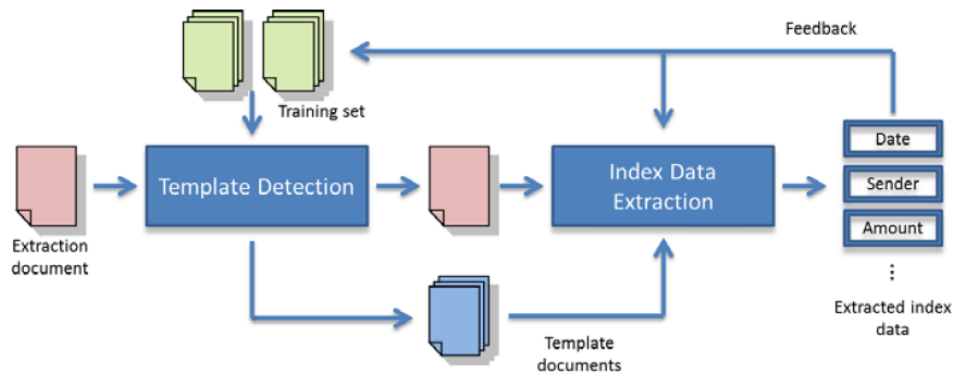


Figure 4: Flow of an extraction document through the information extraction system

2.3 Content Extraction of PDF Documents

To identify logical components of a PDF document, Hui Chao and Jian Fan published a research paper in 2004 by developing techniques. Using these techniques, they tried to extract contents, outlines and style attributes from the logical components and expressed that extracted data in an XML format [1]. Such approaches may encourage the modification and reuse of a PDF document page's layout and its content.

Researchers introduced techniques to automatically segment a PDF document into separate logical structures. These structures are identified as text blocks, images blocks, vector graphics blocks, and compound blocks and logical component of the document are represented by them. Then they specified each component's geometrical outline, extracted the content and style attributes of each component.

Content stream of a PDF document include all the page objects. Those are not present the logical structure of a page. To identify the logical components, the layouts and characteristics of the page objects were analysed and interpreted. Then they group them into different logical components.

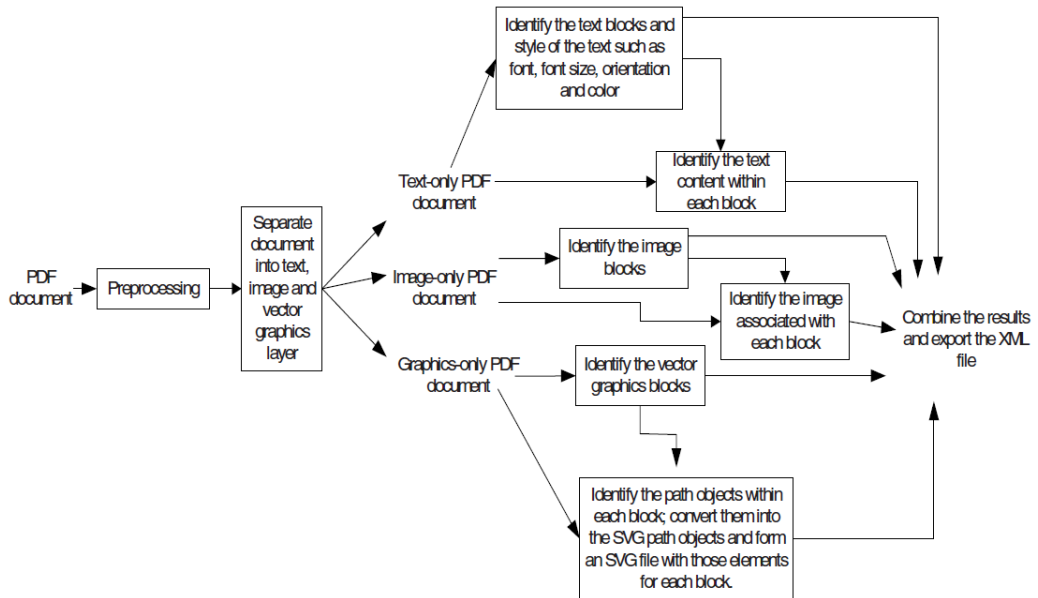


Figure 5: overview of the processing pipeline

Above figure shows their processing pipeline. They have divided PDF page into layers to prevent the overlay of logical component and the intervening within logical components. There're three different layers named text layer, image layer and vector graphics layer. Every layer represented as separated document. "To specify the logical structural component blocks in each layer, they transform the PDF files to bitmap images, and do image segmentation. Or else they have directly analysed the three PDF documents" [1]. Then they have extracted the content and style attributes of each component.

- Pre-processing

Page objects can be simple and can be compound types. All compound objects will extract on this stage.

- Splitting the document

Document will split into text, image and vector graphic layers.

- Text segmentation and extraction

A PDF document provide information regarding text on the page such as characters used in the text, their attributes and positions. Text segmentation was performed directly on PDF document.

- Image Detection & Extraction Components

They have converted image only PDF to a bitmap image to detect images on the page.

- Vector Graphics Segmentation and Extraction

Researchers have applied a document image segmentation method to locate the regions that defined various components. To find the path objects within each component on the PDF page, the outline of each region was used and the SVG representation for the logical component was generated with those path objects [1].

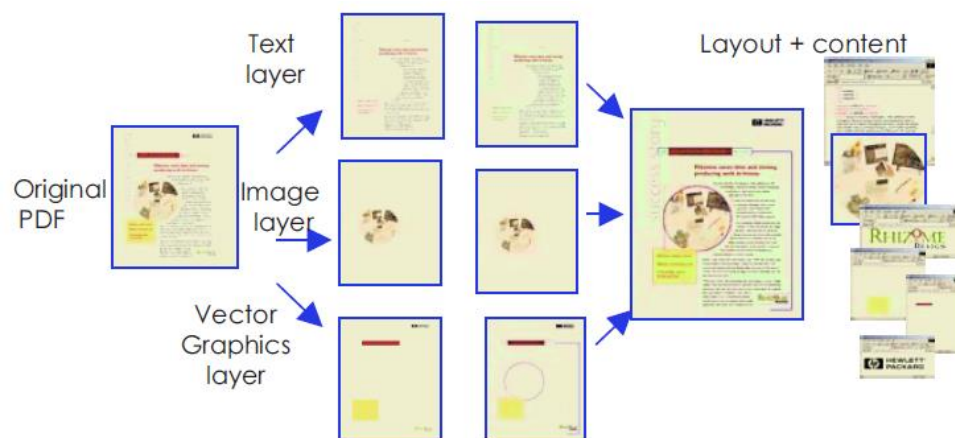


Figure 6: System processing steps

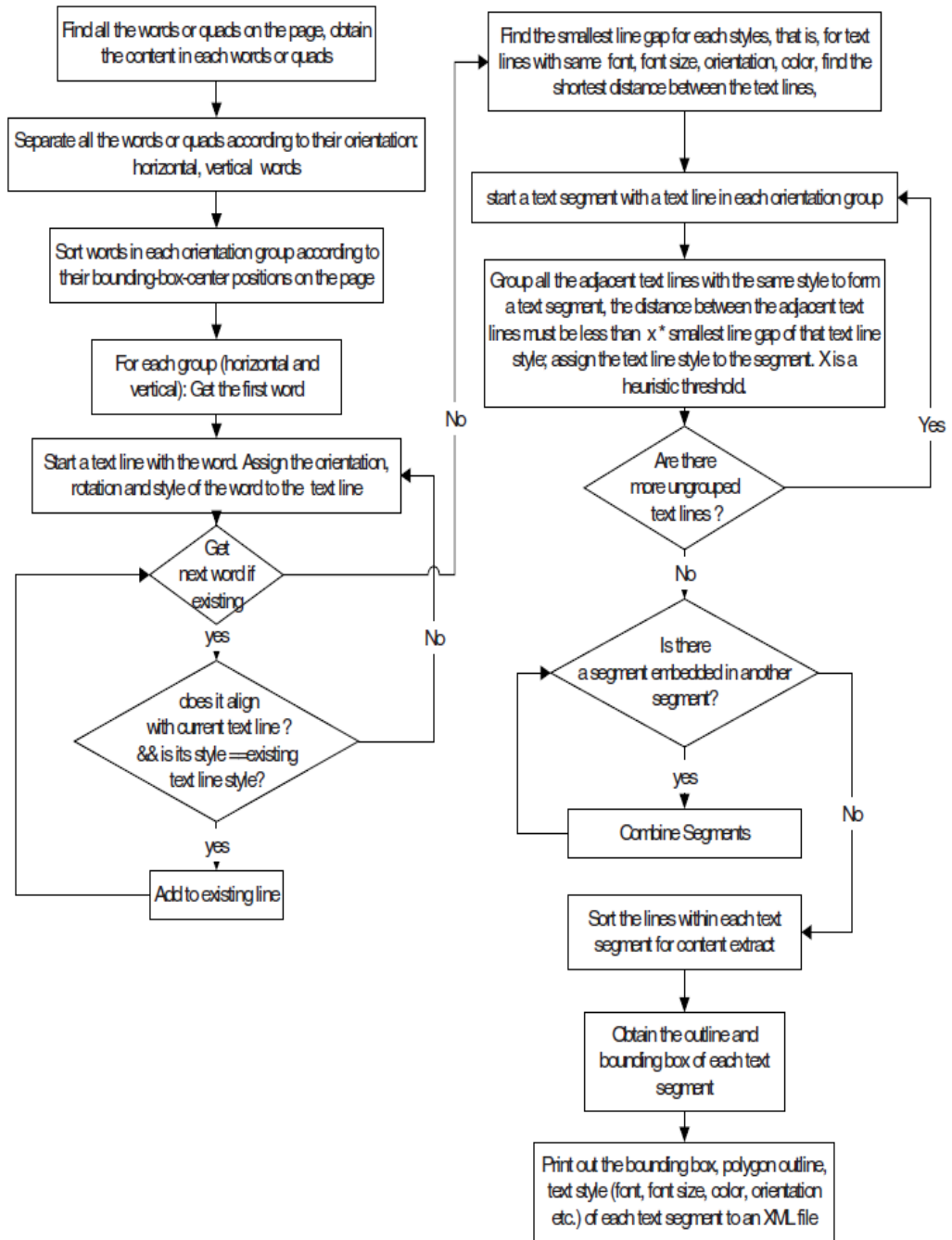


Figure 7: The flowchart for text segmentation

2.4 Diagram Extraction of PDF documents

Robert P. Futrelle, Mingyan Shao, Chris Cieslik and Andrea Elaina Grimes presented a paper on 2003 based on diagrams classification and extraction of PDF documents. For a document management system, diagram analysis is an important feature. Researchers investigate about vector images that are in research papers [10].

Analysis of the internal structure of the diagrams will be very useful in extracting its' data and it will use to indexing for data retrieving [11]. Documents include images with vector or raster format. Researchers focused only on vector-based diagram classification.

For selected PDF documents, PDF commands are parsing to generate a vector of graphics objects. Then those objects will transform to their own graphic object definition. Below is a sample representation.

```
172.443 482.809 m
172.443 483.756 l
172.885 483.775 173.187 483.805 173.365 483.844 c
173.646 483.906 173.871 484.027 174.049 484.215 c
```

Which gives four GOs,

- m- corresponding to a move of the drawing position
- l- drawing a line from there to the position given
- c- drawing two curved lines in succession

To detect pages with graphics, they have count the GOs. If the number of GOs are more than 20, it is likely to be a diagram and was analysed further.

“A spatial index is used to analyse the spatial layout structure of the graphics in a page. The one used is Spatially Associative Substrate (SPAS) that developed for diagram parsing” [12].

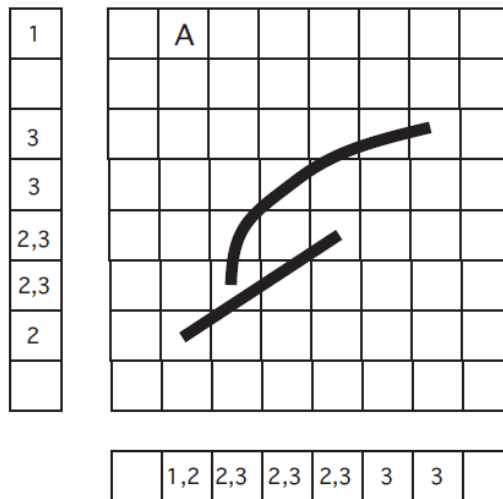


Figure 8: An 8x8 SPAS structure for spatial indexing

2.5 Benchmark and Evaluation for Text Extraction from PDF

“Extracting text from PDF file is a difficult task because PDF is a layout-based format which specifies the fonts and positions of the individual character rather than the semantic units of the text and their role in the document” [13]. There’re many content extractions tools but their quality and functionality hard to determine.

In 2017, Hannah Bast and Claudius Korzen presented a paper by showing how to develop a good benchmark from TeX and PDF data. They established a set of standards for a clean and independent evaluation of a given extraction tool’s semantic abilities [14].

They have investigated following semantic information in their paper to construct a high-quality benchmark.

- Word identification

This is a very important for an application like search. Because if word is not identified correctly that word is not found.

- Word order

This specifies the correct order of words to read.

- Paragraph boundaries

This is also important for reflow applications or for plain text reading. This specifies the start and end of a paragraph.

- Semantic roles

This defines the different roles of the text appearing on the document. As an example, text from the document body and the text from other document elements.

To construct the benchmark researchers used more than 12000 scientific articles. The benchmark includes the PDF file and a ground truth file for each article. Ground truth files contain article's title, section titles and body text as a plain text.

Benchmark PDF files were processed by each tool in batches. After converting the output to plain text, they have selected semantic parts that also appear on the ground truth file.

The main purpose of this evaluation is to use a collection of easily interpretable and independent metrics to compare output files from each tool with the ground truth files [13].

- Establishing the evaluation criteria

They have specified four aspects for evaluating the quality of an output file.

- i. paragraph boundaries
- ii. distinction of body text and non-body text
- iii. reading order
- iv. word boundaries

Three categories of metrics were defined that measure the differences between the output file and the relevant ground truth file.

- i. Newline differences

NL^+ : the number of spurious newlines in the output file.

NL^- : the number of missing newlines in the output file.

- ii. Paragraph differences

P^+ : the number of spurious paragraphs in the output file.

P^- : the number of missing paragraphs in the output file.

$P\updownarrow$: the number of rearranged paragraphs in the output file.

- iii. Word differences

W^+ : the number of spurious words in the output file.

W^- : the number of missing words in the output file.

$W\sim$: the number of misspelled words in the output file.

- Measuring the evaluation criteria

For a particular output file and a ground truth file may have many ways to set values to these criteria [13].

output file <i>O</i>	ground truth file <i>G</i>
<i>Text Extraction PDF.</i> <BLANKLINE> <i>A Benchmark and</i> <BLANKLINE> <i>Evaluation for</i>	<i>A Benchmark and</i> <i>Evaluation for Text</i> <i>Extraction from PDF.</i>

Figure 9: An output file with 3 paragraphs and the ground truth file with 1 paragraph

Assignment 1: $P^+ : 3, P^- : 1$	Assignment 2: $P^+ : 1, NL^+ : 1, W^- : 4$	Assignment 3: $P^{\downarrow} : 1, NL^+ : 2, W^- : 1$
<i>Text Extraction PDF.</i> <BLANKLINE> <i>A Benchmark and</i> <BLANKLINE> <i>Evaluation for</i> <i>A Benchmark and</i> <i>Evaluation for Text</i> <i>Extraction from PDF.</i>	<i>Text Extraction PDF.</i> <BLANKLINE> <i>A Benchmark and</i> <BLANKLINE> <i>Evaluation for Text</i> <i>Extraction from PDF.</i>	<BLANKLINE> <BLANKLINE> <i>A Benchmark and</i> <BLANKLINE> <i>Evaluation for</i> <BLANKLINE> <i>Text Extraction from</i> <i>PDF.</i>

Figure 10: three different assignments to the evaluation criteria in order to assess *O* against *G*

To solve that problem researchers introduced an equation to compute ‘*Z*’ which is needed to minimize as much as possible.

$$Z = (NL^+ + NL^-) + (W^+ + W^- + W^{\sim}) + c \cdot (P^+ + P^- + P^{\downarrow}),$$

‘*c*’ is defined to raise the weight for paragraph variation more than word or new line variations.

3. METHODOLOGY

The main goal of Portable Document Format is to support view and exchange e-documents independent from the platform that they were created. PDF defines more structured format to improve performance for interactive viewing. To achieve high performance in interactive viewing PDF used optimized structured binary file format.

3.1 PDF Components

PDF consists with four components which are Object, File Structure, Document Structure and Content Stream. Collection of data objects composing a data structure as PDF document [15]. The way objects locating on a file determines by PDF file structure. It describes how objects are updated and how objects are accessed. Fundamental object types such as pages, fonts, annotations, and so forth are used to represent components of a PDF document. The PDF document structure specifies how they are doing that. Appearance of a page describe by sequence of instructions that are includes on the Content stream [9].

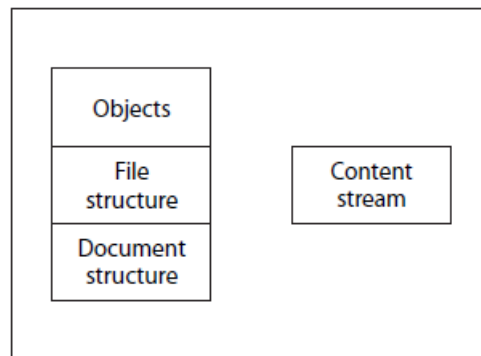


Figure 11: PDF Components

PDF file is a sequence of bytes that are grouped into tokens to form objects. These objects are the fundamental data values used to construct a PDF document. If a PDF file is a non-encrypted file, it can be fully represented using byte values that are mapped with characters defined with 'ANSI X3.4 -1986' and with white space characters. But PDF is not only including ASCII characters, it can include arbitrary bytes also [16].

- PDF Characters

The PDF characters are classified into three parts. These are regular, delimiter and white-space characters.

White space characters are used to separate words. PDF treats those types of adjacent characters as single character except in comment, string, and stream. White space characters are shown on Table 1.

Table 1: White-Space characters

Name	Decimal	Hexadecimal	Octal
Null (NUL)	0	00	000
Horizontal Tab (HT)	9	09	011
Line Feed (LF)	10	0A	012
Form Feed (FF)	12	0C	014
Carriage Return (CR)	13	0D	015
Space (SP)	32	20	040

The delimiter characters are used to delimit syntactic entities (names, arrays, comments, etc.). These are allowed within the string scope if it is following the rule for composing strings [17].

Table 2: Delimiter Characters

Glyph	Name	Decimal	Hexadecimal	Octal
(Left Parenthesis	40	28	50
)	Right Parenthesis	41	29	51
<	Less-Than Sign	60	3C	60
>	Greater-Than Sign	62	3E	62
[Left Square Brackets	91	5B	133
]	Right Square Brackets	93	5D	135
{	Left Curly Brackets	123	7B	173
}	Right Curly Brackets	123	7D	175
/	Solidus	47	2F	57
%	Percent Sign	37	25	45

All characters are defined as regular characters except delimiter characters and white space characters. A token comprises by a sequence of consecutive regular characters. PDF is case sensitive and upper case and lower case shall be considered distinct.

Any ‘%’ sign that are placed outside of a string is represent a comment. All characters that placed after the ‘%’ sign define as the comment. The PDF readers are considering comments as one white space characters.

Example: Following PDF fragment equivalent to token cse and 456.

```
cse% comment (/%) something
456
```

PDF readers may not consider abouts comments, because comments have no semantics.

3.1.1 Objects

PDF include following types of objects.

- Strings
- Numeric
- Names
- Boolean Values
- Arrays
- Dictionaries
- Streams
- Null Objects

Objects may be labelled and those are called as indirect objects. So that they can be referred to by other objects.

- Strings Objects

A String objects consist with series of zeros or more bytes. More compact format used to store string objects and string length may leads to implementation limits.

- Literal Strings

Literal string is a sequence of characters that are wrapped in quotation marks.

Example: (I am a string)

(This string includes
A new line.)

(String can include balanced parenthesis () and
It may include special characters (!* & ^}% and etc..))

(below string is empty)
()
(string length is zero (0))

Arbitrary 8-bits values represent encrypted strings on encrypted documents values. When encrypted string presented by literal string form, relevant application must follow some rules. That is ‘\’ character must use as an escape to represent unbalanced parenthesis or ‘\’ character itself. ‘\’ character may use to represent other arbitrary 8-bits values.

- Hexadecimal strings

Hexadecimal strings are useful for use in a PDF file with arbitrary binary data. This is provided as a series of hexadecimal digits within ‘<>’.

Example: <756e6976657273697479206f66206d6f726174757761>

One byte of the string is represented by pair of hexadecimal digits. White space characters are ignored. If the hexadecimal string end digit is missing it is presumed to be 0.

- Numeric Object

This type of objects representing as either integer or real numbers. An integer represents by one or many decimal digits with a sign preceding it.

Example: 123 +54 -65 0

Real numbers should express as one or many decimal digits, with an optional sign and a decimal point.

Example: 34.5 +52.5 -6.35 0.0

- Name Objects

Name object created with the same set of characters denoted the same object and it has no internal structure. This is a unique symbol identified by a sequence of any other characters other than null.

When writing name on a PDF file '/' sign is used to start a name but it is not representing the name. white spaces used as a segment of a name.

Example:	/Name1	Name1
	/1.2	1.2
	/lime#20Green	lime Green

Unlike keywords such as true, false and object, in PDF file literal names introduced by the '/'. Typically name objects never treated as text but occasionally there're situations that is name objects are treat as text like when represent a font name or a colorant name in a separation.

- Boolean Objects

The Boolean object represent true and false logical values. In a PDF file these are represent by using keywords 'true' and 'false'.

- Array Objects

This is a sequentially ordered, set of objects. These arrays may be heterogeneous unlike many other computer languages. That mean these arrays can contain string, numbers, dictionary or any other objects. Array may have a zero element.

Example: [123 /name (test1)]

These types of objects represent within a PDF document as a sequence of objects enclosed within '[]'. PDF directly support only for one-dimensional array. Higher dimensional arrays can construct by using nested arrays.

- Dictionary Objects

This is an associative table with couple of objects in it. First is key and second is value. Key is the name, and any form of object can be value. There is no common key for different entries in the same dictionary. Dictionaries represent within a PDF file by sequence of key value pairs enclosed in '<< >>'

Example: << /Type /Example
 /Version 1.2
 /Subdictionary << >>
 >>

these objects are the foundation of a PDF file. Typically, these are used to gather and keep attributes of a specific entity.

- Stream Objects

This is a series of bytes, which can have an unlimited length stream. This object is like string object, but string object may have implementation limit. Since stream may contain data with unlimited length, large data objects representing as stream.

Stream objects consist with zero/more bytes within tags ‘stream’ and ‘endstream’ which are followed a dictionary.

Example: dictionary
 stream
 ... zero / more bytes as the content ...
 endstream

Common properties for stream dictionaries are listed below.

Table 3: Entries on stream dictionaries

Key	Type	Value
Length	Integer	Required - Number of bytes the PDF file is used for the stream's data. (within 'stream' and 'endstream' keywords)
Filter	Name or Array	Optional – Name of a filter that shall be applied in processing the stream data found between the keywords 'stream' and 'endstream'.
DecodeParams	Dictionary or Array	Optional – Parameter dictionary or array of such dictionaries that is used by the filters specified by filter.
F	File Specification	Optional – File containing the stream data.
FFilter	Name or Array	Optional – Name of a filter to be applied in processing the data found in the stream's external file.
FDecodeParams	Dictionary or Array	Optional – Parameter dictionary or array of such dictionaries that is used by the filters specified by FFilter.
DL	Integer	Optional – Non-negative integer representing the number of bytes in the decoded stream.

- Null Objects

Null objects contain a value and type that are not match with any other objects. These objects referred by keyword 'Null' and there can be have only one object with type null. An indirect object that is reference to non-existent object shall be treated like null object. Defining the null object as a dictionary entry value will be the same as omitting the entry completely.

- Indirect objects

All objects in a PDF file can be classified as indirect objects. That mean each object has unique identifier so, other object can refer it. This compose with a object Number and a generation number.

Indirect object can refer by using object number and generation number. Indirect object appearing on a PDF file by proceeding object and generation number which is divided by white space. And then there is an object value surrounding by 'obj' and 'endobj' tags.

Example: 12 0 obj
 (Brillig)
 endobj

any object can refer from elsewhere in the file by using object number, generation number and 'R' (keyword) that are divided each part by using white space.

Example: 12 0 R

3.1.2 File Structure

File structure described how objects are arranged in a PDF file. Basically, a PDF file created using four elements [18].

- A header includes the possible version
- A body with objects
- A cross reference table includes the information about indirect objects
- A trailer has the pointer to the cross-reference table

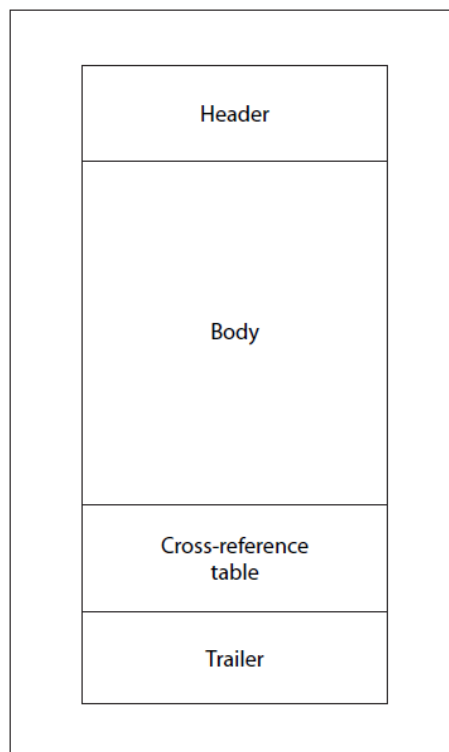


Figure 12: Structure of a PDF document

- File Header

Header is the first line of a PDF file with five characters %PDF- followed by a version number (Example: %PDF-1.7).

if a PDF file contains binary data it will indicate just after the header line and using that PDF reading application can determine whether file needed to treat like binary file or text file.

- File Body

PDF file body containing sequence of indirect object which are represent the content of PDF file. Objects which are represent component of the document like sampled images, fonts and pages. Body may also include streams of objects which each contain a sequence of indirect objects.

- Cross-reference Table

This include entries for indirect objects on a PDF file that are allow random access. Since this table having information about those indirect objects it does not need to read the entire file to locate a specific object. Every indirect object in this table represents a single-line entry with byte offset of the object. This table is the only part with fixed format on the PDF file which allows for random access to table entries [19].

A cross-reference table can include one or many sections. Every section will start by line that contain 'xref' term. After that line there can be more 'cross reference' sub sections.

Example: xref
 0 6
 0000000003 65535f
 0000000017 00000n
 0000000081 00000n
 0000000000 00007f
 0000000331 00000n
 0000000409 00000n

- File Trailer

PDF file trailer enables PDF reading app to locate ‘cross-reference table’ and special objects quickly. Trailer section starting with the term ‘trailer’ and contain key-value pairs within ‘<<’ and ‘>>’.

```
Example:  trailer
          << Key1      Value1
           Key2      Value2
           ...
           Keyn     Valuen
          >>
```

Table 4: Entries in the file trailer dictionary.

Key	Type	Value
Size	Integer	Total number of entries in the PDF file’s cross-reference table.
Prev	Integer	Byte offset in the decoded stream from the beginning of the file to the beginning of the previous cross-reference section.
Root	Dictionary	Catalog dictionary for the PDF document contained in the file.
Encrypt	Dictionary	Document’s encryption dictionary.
Infor	Dictionary	Document’s information dictionary.
ID	Array	A two-byte string array which constitutes a file identifier for the file.

3.1.3 Document Structure

A PDF document consists of object hierarchy found in a PDF file's body section. On the root there is Catalog dictionary of the document [20].

A page object indicates a page of a PDF file. Page object is a dictionary which includes a reference to the content of the page and to other attributes such as its thumbnail image and associated annotation. Individual page objects bound in a structure called the page tree.

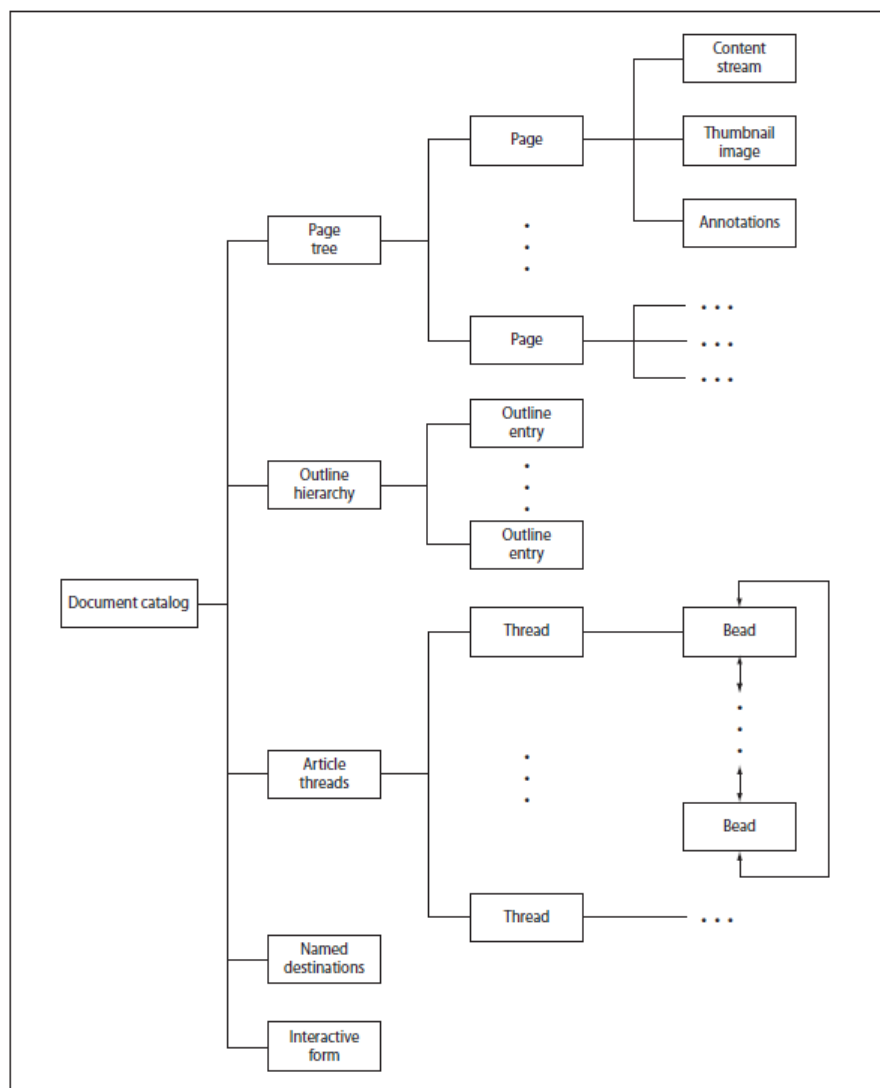


Figure 13: Structure of a PDF document.

- Document Catalog

The catalog dictionary placed on the root of the PDF document. It contains the references for other objects which define document's content, outline and other attributes. Also, it includes other information such as how the document needs to display on the screen.

Example: following shows sample document catalog

```
1 0 obj
  <<  /Type /Catalog
      /Page 2 0 R
      /PageMode /UseOutlines
      /Outlines 3 0 R
  >>
endobj
```

- Page Tree

Page tree defining the order of pages in the PDF document. Using tree structure applications can easily open a thousand-page text. The tree is composed of two node forms.

- Intermediate nodes: page tree nodes
- Leaf node: page objects

The basic structure will contain one-page tree node with direct references to all page objects. But a writer application can create trees of a particular type, known as balanced tree, to optimize the performance of the application.

3.1.4 Content Stream

Content streams are the main element that defines the page's and other graphical element's appearance. Resource dictionary contains information that are related to content stream [21].

Every document page is defined by content streams. It is also used to bundle instruction set as self-contained graphical elements like certain fonts, forms, patterns and annotation appearance.

3.2 Text in a PDF file

PDF having special facilities to working with text. Specially, defining character using glyphs. It is a graphical element that can do any graphical processing like transformation of coordinates. PDF offers high facilities for convenient and efficient describe, selection and rendering of glyphs [22].

- Fonts

Fonts represent glyphs for the character set.

Example: A, **A** and *A* are glyphs for 'A'.

Fonts are representing as a program so conforming reader can use that as a program. Typically, font program represents as a separate file. This file can obtain from an external source or may be contained in PDF document's stream object. Font program consists with glyph definitions that generate glyphs.

A content stream paint glyph using a font dictionary.

Example: in this example text 'XYZ' is using 12-point Helvetica and locate 10 in from bottom and 4 in from left.

```
BT
    /F13 12 Tf
    288 720 Td
    (XYZ) Tj
ET
```

Following is the steps that describe above example.

- Begin Text.
- Set font & size to be used.
- Specify a page start location.
- Render the glyphs at that place for character start.
- End Text.

First content stream needs to detect the font that need to use. The operator 'Tf' will define the name of a font resource. It placed on current resource dictionary as an entry on font sub dictionary.

The font dictionary will itself include a definition of the font program. It defines the externally recognized name of the font and some other details required to applications to draw different glyphs.

Glyphs are defined in one standard size by a font. The default font size is then scaled to be usable. Second operand is scale factor of the 'Tf' operator, so setting the parameter 'text font size' in the graphic state.

It can be used to paint glyphs after selected and scaled font. 'Td' operator adjusts the Text Matrix translation component. 'Td' determines the text location in the current user coordinate system when executing for the first time after 'BT.' This shows the place on the page where glyphs are to start painting.

Using font and other parameters in the graphic state, 'Tj' paints the respective glyphs after taking string operand. Operator 'Tj' treat every string element as a character code.

Glyph descriptions need to execute to render a glyph. This is the flow for simple fonts but for composite fonts interpreting the string as a sequence of character code is more complicated.

3.3 Graphical effect on Text

Typical use of 'Tj' and other operators allows painting of black filled glyphs. Font operators and general graphic operators combine and provide other effects to a text.

Current colour in the graphic state use to pain glyphs. Based on the text rendering mode it can be either stroking color and/or non-stroking color. Default colour is black and to get other colours need to execute colour setting operator or operators before generating the glyphs.

Example: following example use operator g and text rendering mode 0 to fill the glyph in % grey.

```
BT
  /F13 48 Tf
  20 40 Td
  0 Tr
  0.5 g
  (ABC) Tj
ET
```



Figure 14: Glyphs painted in 50% gray

Many graphic styles can be accomplished by using glyph outline as a path. Glyph outline can fill, stroke, etc. and this will be defined in the graphics state by a text-rendering mode parameter.

Example: On the following example text rendering mode limit to 1. Operator 'w' limits the line width to 2 units. By using these parameters 'Tr' operator stokes outlines of the glyph with 2 points thick line.

```

BT
  /F13 48 Tf
  20 38 Td
  1 Tr
  2 w
  (ABC) Tj
ET

```

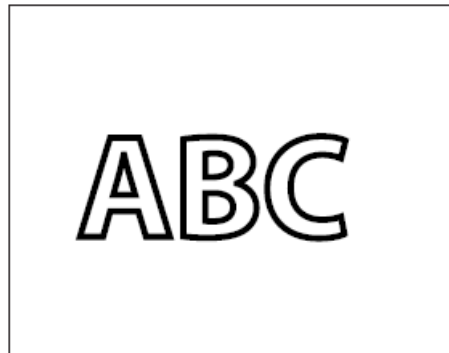


Figure 15: glyph outline treated as stroked path

3.4 Text state parameters

Text state can contain 9 parameters.

Table 5: Text state parameters

Parameter	Description
T_c	Character spacing
T_x	Word spacing
T_h	Horizontal scaling
T_l	Leading
T_f	Text font
T_{fs}	Text font size
T_{mode}	Text rendering mode
T_{rise}	Text rise
T_k	Text Knockout

Values that set by text state operators are persisted in a single content stream across text objects. These parameters initialize at start of the page to their default values like other parameters in the graphics state.

Table 6: Text state operators

Operands	Operator	Description
charSpace	T _c	set the character spacing. This is a number expressed in unscaled text space unit. Initial value: 0
wordSpace	T _w	Set the word spacing. This is a number expressed in unscaled text space unit. Initial value: 0
scale	T _d	Set the horizontal scale. Scale is a number specifying the percentage of the normal width. Initial value: 100 (normal width)
leading	T _l	Set the text leading. This is a number expressed in unscaled text space unit. Initial value: 0
Font size	T _f	Set the text font. There is no initial value either for font or size. They specify explicitly by using 'T _f ' before any text is shown.
render	T _r	Set the text rendering mode. Initial value: 0
rise	T _s	Set the text rise. This is a number expressed in unscaled text space unit. Initial value: 0

In this table state that some parameters defined in unscaled text space unit.

- Character Spacing

$T_c = 0$ (default)	Character
$T_c = 0.25$	C h a r a c t e r

Figure 16: Character spacing in horizontal writing

- Word Spacing

$T_w = 0$ (default)	Word Space
$T_w = 2.5$	Word Space

Figure 17: Word spacing in horizontal writing

- Horizontal Scaling

$T_h = 100$ (default)	Word
$T_h = 50$	WordWord

Figure 18: Horizontal scaling

- Leading

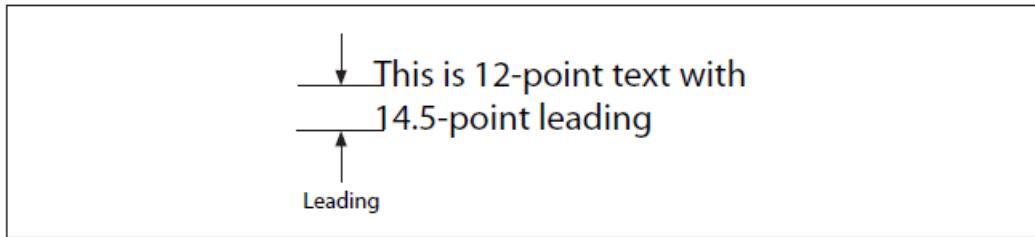








Figure 19: Leading

- Text rendering mode

Table 7: Text rendering modes

Mode	Example	Description
0		Fill Text
1		Stroke text
2		Fill, then stroke text
3		Neither fill nor stroke text (invisible)
4		Fill text and add to path for clipping
5		Stroke text and add to path for clipping
6		Fill, then stroke text and add to path for clipping

7		Add text to path for clipping
---	---	-------------------------------

- Text Rise

(This text is) Tj 5 Ts (superscripted) Tj	This text is ^{superscripted}
(This text is) Tj -5 Ts (subscripted) Tj	This text is _{subscripted}
(This) Tj -5 Ts (text) Tj 5 Ts (moves) Tj 0 Ts (around) Tj	This _{text} ^{moves} around

Figure 20: Text Rising

3.5 Text Objects

This will contain operators that are capable of displaying a text string, moving the location, setting text state & some additional params. Other than that, inside the text object it defined three parameters and it only within the object & not retained from one to another.

- T_m : Text matrix
- T_{lm} : Text line matrix
- T_{rm} : Text rendering matrix

BT operator defines start of the text object and ET operator defines end of it.

Example: BT
 ... Zero to many text operators or other permitted operators ...
 ET

Following operators can include on a text object:

- Text state operators
- Text positioning operators
- Text showing operators

Text space is the coordinate-system where text displayed that denoted by T_m . Text state params (T_{fs} , T_h and T_{rise}) are decide the transformation from text space to user space. The operators for text positioning shall appear only inside text objects. Text showing operators will show text on the page.

In conceptual terms, a text rendering matrix, T_{rm} , that reflect the text space to device space transformation.

$$T_{rm} = \begin{bmatrix} T_{fs} \times T_h & 0 & 0 \\ 0 & T_{fs} & 0 \\ 0 & T_{rise} & 1 \end{bmatrix} \times T_m \times CTM$$

3.6 Extraction of text content

While extracting text from a PDF file, an app transform text to Unicode. To do that application must detect font characters. This recognition will happen if the characters in the font are defined by standard character names or the font uses CIDs in a well-known collection.

A PDF reader can use following methods to map character code to Unicode value.

- If a ToUnicode CMap is contain on the font dictionary use CMap.
- If the font using predefined encodings;
 - a) Map character code to character name using following table & font's difference array.

Table 8: Latin-text encoding

Encoding	Description
StandardEncoding	Adobe standard Latin-text encoding.
MacRomanEncoding	Mac OS standard encoding for Latin text in Western writing system.
WinAnsiEncoding	Windows Code Page 1252, often called the 'Windows ANSI' encoding

- b) To obtain the corresponding Unicode value check character name on adobe glyph list.

4 IMPLEMENTATION

4.1 High level architecture

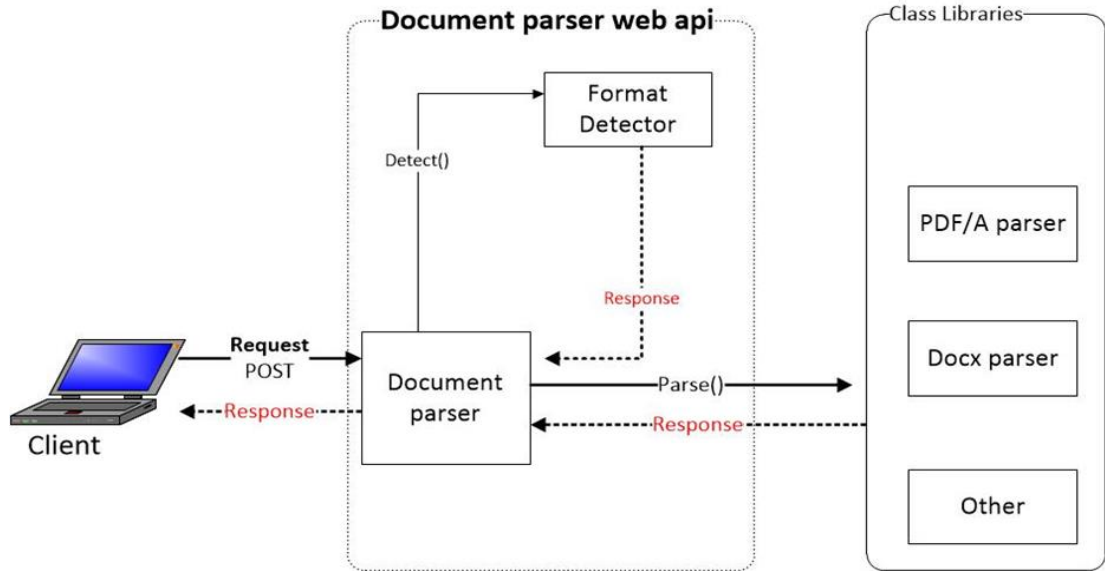


Figure 21: High level architecture

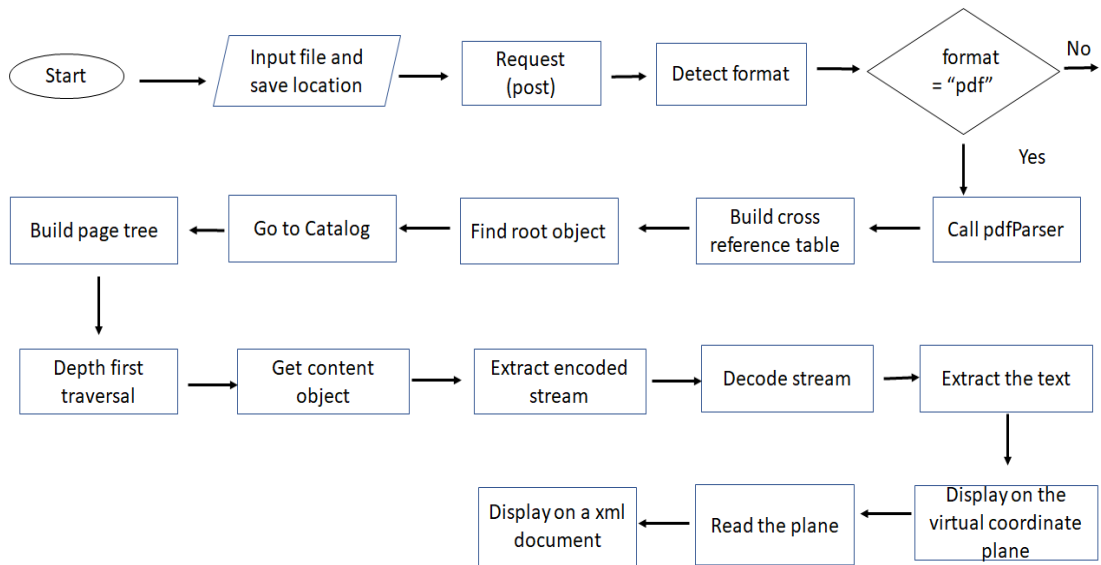


Figure 22: Flow Diagram

4.2 Getting the last cross reference table (xref) offset

The last xref offset is the starting offset of the last modified cross reference table. First, I extracted the last xref offset in order to build the cross-reference table. This was embedded at the end of the pdf in between “”startxref”” and “%%EOF”. There can be spaces and carriage return in before, after or in between the above. I handled all those cases and extracted the offset. After extracting the offset, we could read the xref segment through a byte array segment reader.

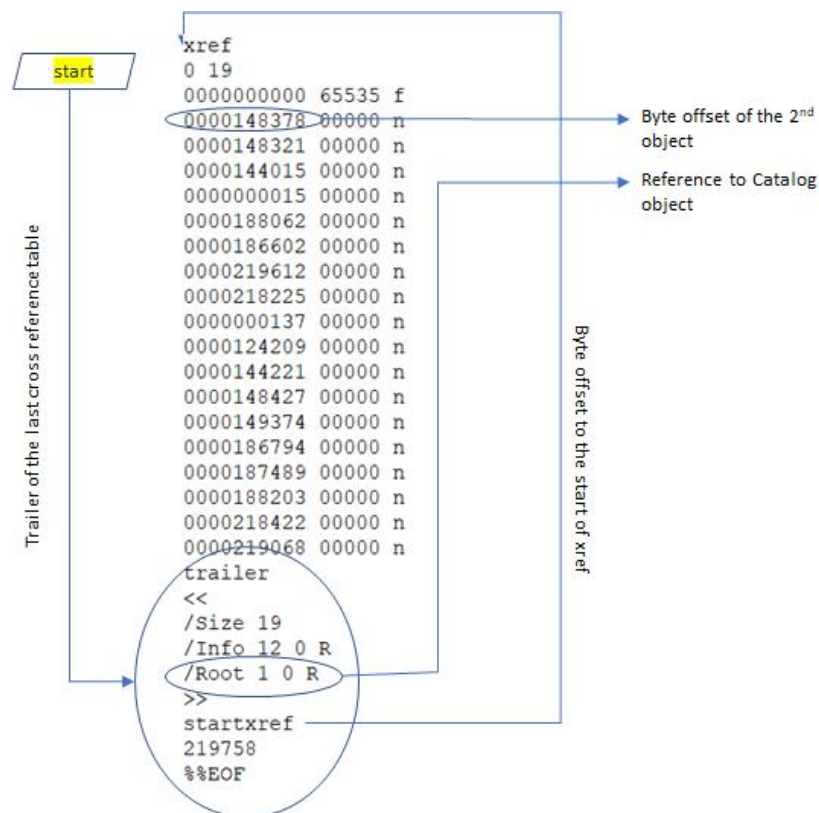


Figure 23: Getting the last cross reference table offset

Then implemented a tokenizer to tokenize the xref objects and through that the xref was tokenized and the object ids and the relevant offsets were extracted for implementing the cross reference table which contains the starting byte offset of each and every object embedded in the pdf .

4.3 Creating the cross-reference table

A Cross Reference Table contains all the object ids and their relevant offsets. To build this I used the string tokens sent by the xref tokenizer and built the cross-reference table using a dictionary with object id as the key and the offset as the value [24].

Whenever the pdf is updated or modified a new xref with only the changed object ids and offsets is built. We can find the offset of the previous xref that was built before the modification at the trailer the trailer entry as shown in the **Figure 23**. In such instances all the xref tables are merged and the updated version of the cross-reference table is built. Then that can be used to refer the offset whenever an object needed to be read. This made it possible to read only the relevant section of the byte array rather than reading the whole array.

4.4 Finding the root and creating the page tree

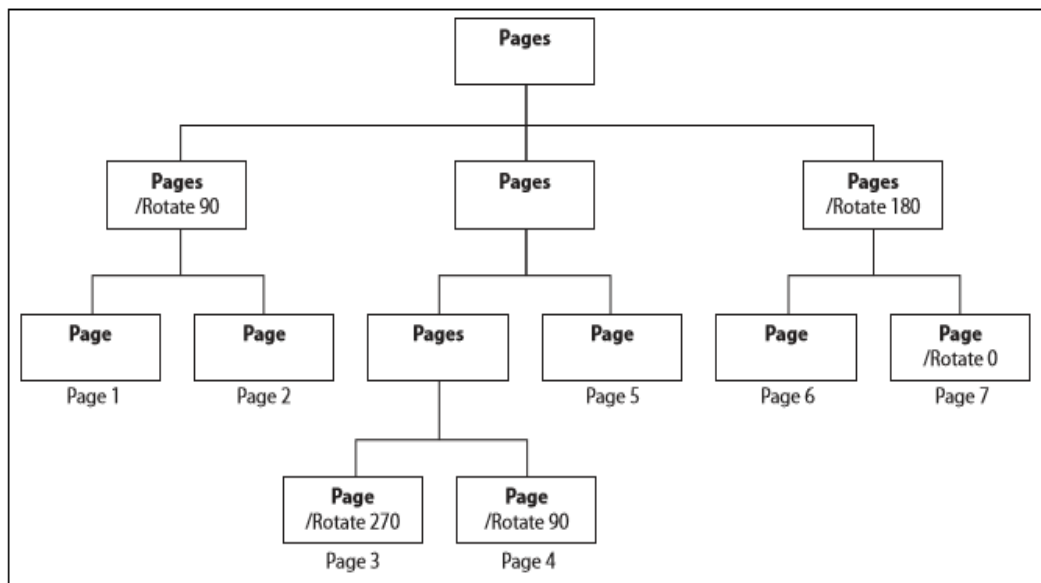


Figure 24: Page Hierarchy

Root object is the root of the pages tree which gives reference to a Pages object. To build the pages tree I extracted the root object from the trailer. A page object may contain references to other pages objects or page objects as shown in the above diagram. A page object contains references to content objects where the text contents are stored. By checking all those object references I created a tree data structure for processing each page. In order to make sure the order of pages and contents I used Depth First Traversal method to process the tree and built a list of content object ids in the correct order for further processing.

4.5 Tokenizing

The only resource we had to deal with was the pdf byte array. But this is not programmable. To make it programmable we had to tokenize the text stream of the byte array into several tokens to match each object to a fixed pattern. I caught the starts and ends using the delimiters and characters and using many regex tokenized them into many tokens such as Begin text token, end text token, dictionary start token(<), dictionary end token(>), numeric token, object reference token, offset token, stream token, string token, trailer token, start object etc. For building these tokens I used many tokenizers such and xref tokenizer to tokenize the xref objects, Object tokenizer to tokenize object tokens, Text tokenizer to tokenize texts, Cmap tokenizer to tokenize cmap streams [25].

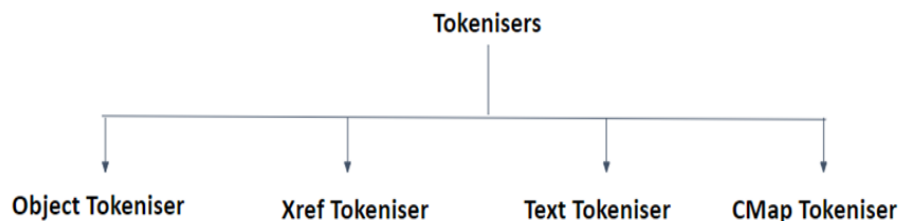


Figure 25: Tokenizers

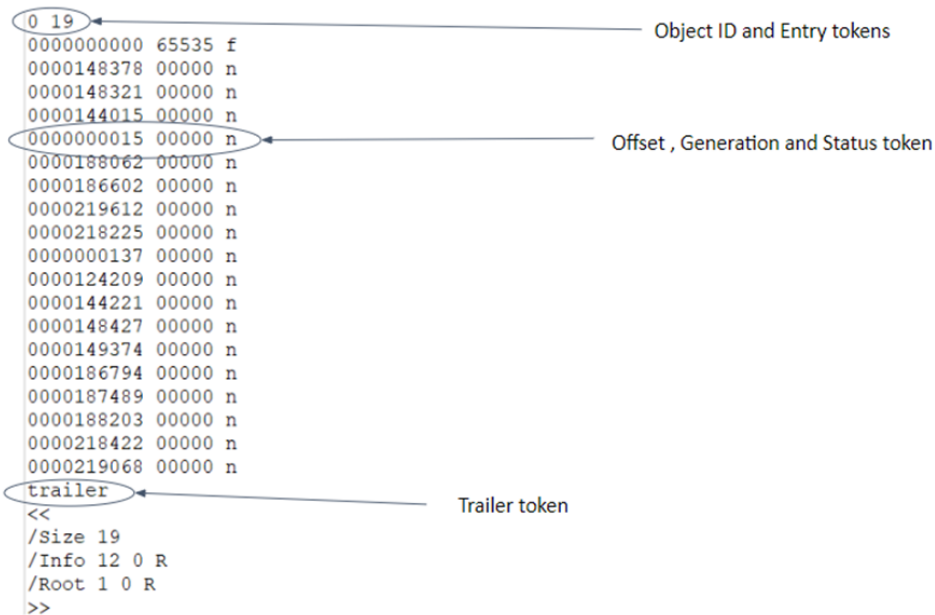


Figure 26: Tokenizes the xref objects

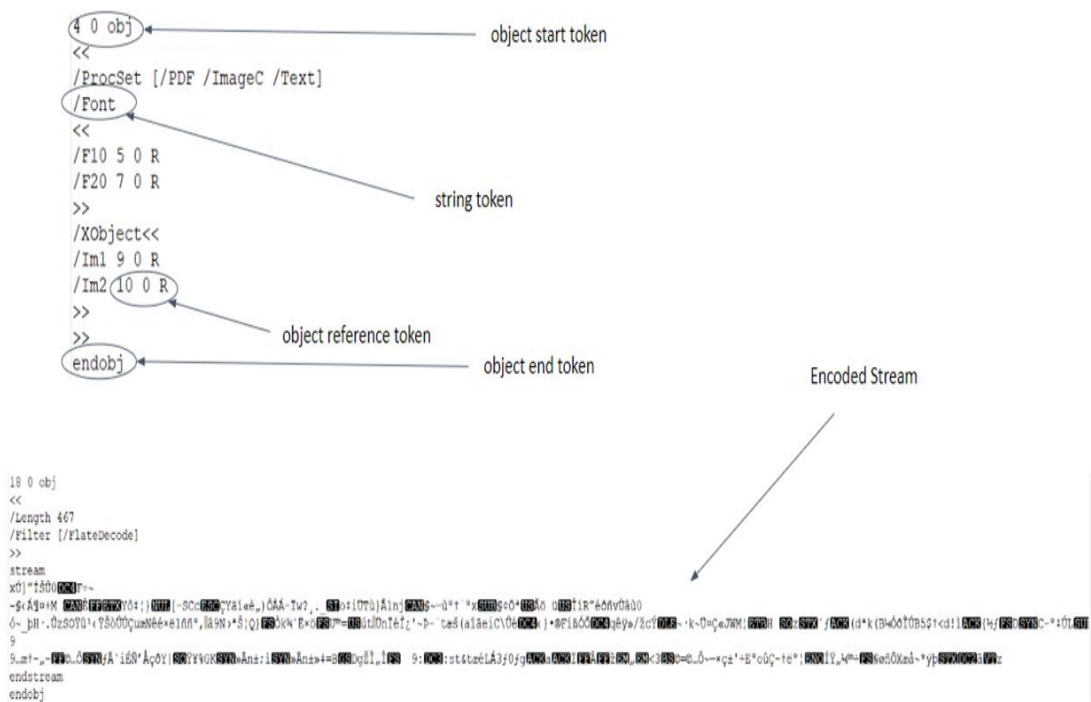


Figure 27: Tokenizes the objects

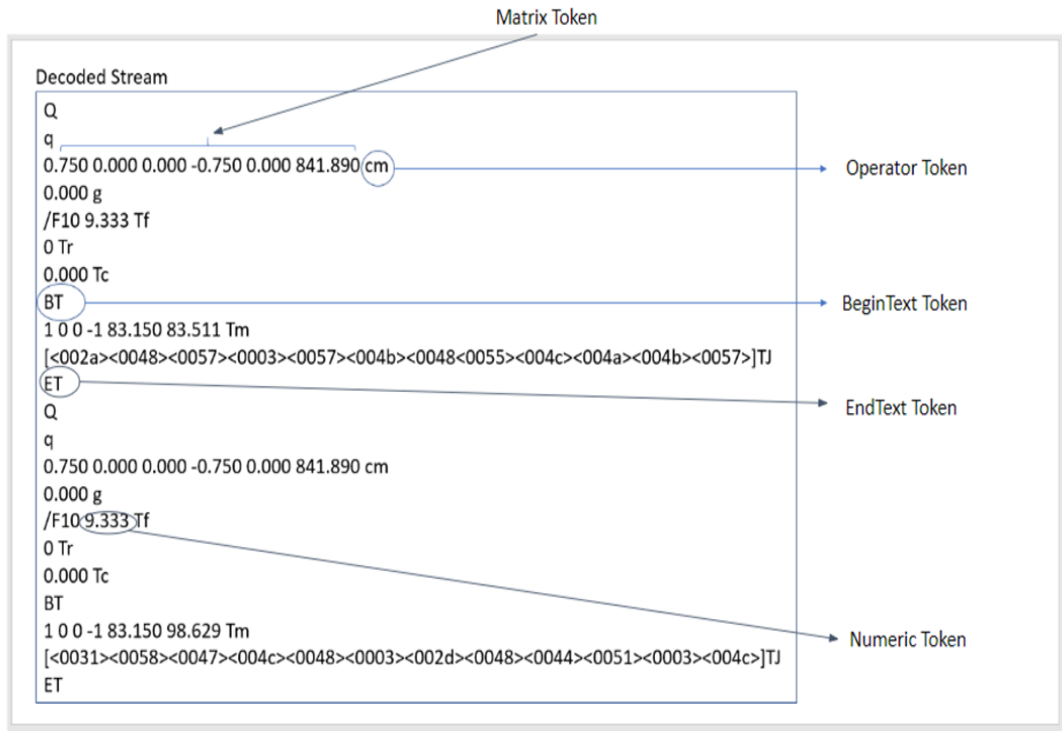


Figure 28: Tokenizes the decoded texts

```

/CIDInit /ProcSet findresource begin ← Operator token
12 dict begin
begincmap
/CIDSystemInfo<</Registry (Adobe)
/Ordering (UCS)
/Supplement 0
>> def
/CMAPName /Adobe-Identity-UCS def
/CMAPType 2 def
1 begincodespacerange
<0000> <FFFF>
endcodespacerange
68 beginbfchar
<0003> <0020>
<0009> <0026>
<000e> <002b>
<000f> <002c>
<0010> <002d>
<0011> <002e>
<0013> <0030>
<0014> <0031>
<0015> <0032>
endbfchar
endcmap
CMAPName currentdict /CMAP defineresource pop
end
end

```

Operator token

Figure 29: Tokenizes the Character mappings

4.6 Token Handling

The binary code of a pdf is always object oriented. These objects needed to be constructed. Problem was that binary code was not programmable and there wasn't any fixed pattern in the code. Hence, the need for tokenizer. Tokenizer was responsible for providing predefined token by token, as for the need of token handler. The token handler had to process the receiving tokens and use the object builder factory to construct corresponding objects. For example, there may be arrays, dictionaries, keywords, strings, numeric values etc. in a pdf object. The tokenizer can identify each as a predefined token. The Token Handler will be asking for tokens for a specific object starting from the "Start Object Token". A start object token can be defined as a token with the object id, generation number and "obj" keyword (1 0 obj). There are several predefined tokens like the one mentioned. If the object contains a dictionary, TokenHandler will identify the dictionary start by "DictionaryStartToken" which appears like "<<". An array start can be shown like "[". A dictionary or an array may contain a sub array or dictionary as well. Therefore, the implementation of TokenHandler has a recursive approach.

4.7 Object Builder

As stated above, the pdf binary code is object oriented and these objects needed to be constructed as per need of the pdf parser. "Factory method" is implemented for object builder. There exists a builder factory for providing the specific object builder for creating pdf objects. Token Handler was used by the object builders. Some objects of a pdf can be listed as follows.

Catalog, Page, Pages, Stream, Font, XObject, etc. After the tokens are being handled, the object builder class will analyze the token list and build the particular object by using a builder of the Object Builder Factory.

4.8 Reading the content objects

To process content objects each content object is read using object reader by traversing the page tree in depth first traversal methodology. Then the streams which are encoded are tokenized and stored in stream tokens objects. Then they are decoded using the relevant decoder. Some of the mostly used decoders are FlateDecode, RunLengthDecode, ASCII85 Decode. In some cases, the same stream is encoded by more than one encoder. In such instances they are decoded in the order of decoders under the filter's entry in the content object. After decoding the streams, the extracted content is as below [26].



Figure 30: Decoded stream

4.9 Character Mapping

The **Figure 30** shows that a TJ entry may not provide only actual text. It could be either hexadecimal string or octal string. These values need to be converted to actual readable standard text.

There are many ways in which these hexadecimal strings or octal strings can be converted into Unicode text. It differs considering the font type and font dictionary operations. For instance, if the font type is Type 0, the font program may contain a ToUnicode entry. This entry will be directing towards a cmap; which is also encoded, to map the characters. Or else, if the font is of type 1 or true type, the “Encoding” entry will be referencing a predefined character map for the conversion.

Figure 31 shows a decoded Cmap where character mapping dictionaries are available. The dictionaries may be in many forms. The diagram shows a single format, where characters are mapped by “bfchar” dictionary. Here each hexadecimal string is coupled with the adjacent unicode character in hexadecimal. In addition to bfchar dictionaries, dictionary formats like bfrange, cid range could be available. A bfrange dictionary differs from a bfchar dictionary only when the value of the dictionary is in the form of an array. A mapper class is defined for this character mapping.

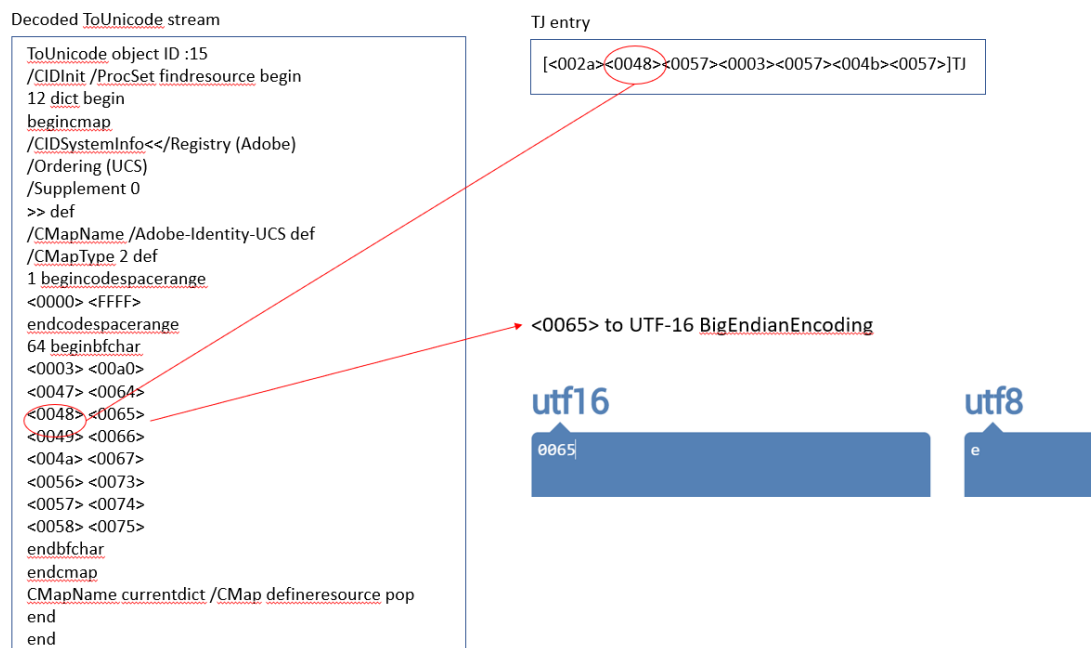


Figure 31: Decoded to Unicode stream

4.10 Character Mapping

After extracting glyphs and relevant coordinates we had to develop a virtual coordinate plane to place them. For this I designed a custom data structure with linked lists where a page is a node and I used glyph objects to store the glyph details and stored them in a sorted dictionary where the y coordinate is the key and another sorted dictionary with x coordinate as the key and the glyph object as the value as the value of the main sorted dictionary [27].

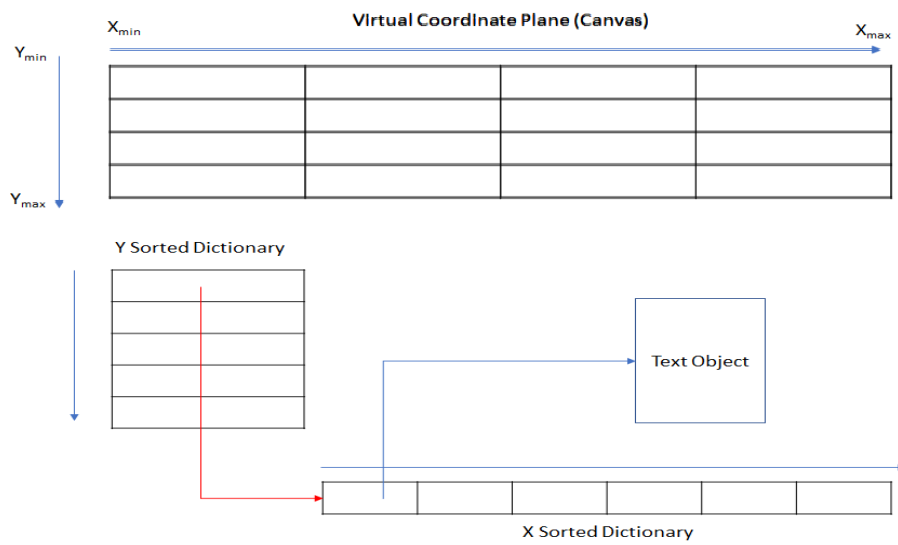


Figure 32: Coordinate plane

4.11 Creating Text Rendering Matrix (TRM)

To find the coordinates of each glyph TRM for each glyph had to be calculated. For this I used formulas stated in the adobe specification.

$$T_m = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} \times T_m$$

$$T_{rm} = \begin{bmatrix} T_{fs} \times T_h & 0 & 0 \\ 0 & T_{fs} & 0 \\ 0 & T_{rise} & 1 \end{bmatrix} \times T_m \times CTM$$

$$t_x = \left(\left(w\theta - \frac{T_j}{1000} \right) \times T_{fs} + T_c + T_w \right) \times T_h$$

$$t_y = \left(wI - \frac{T_j}{1000} \right) \times T_{fs} + T_c + T_w$$

4.12 Decomposing Text Rendering Matrix

Some texts have a rotation and to find the rotation we had to decompose the TRM (Text Rendering Matrix) matrix. For that I decomposed the TRM matrix as Transition matrix, Scale matrix and Rotation matrix. For the rotation I rounded off all the angles in to four nearest angles as 0 degrees, 90 degrees, 180 degrees and 270 degrees. For these four planes I built four separate sorted dictionaries and placed them in the virtual coordinate plane.

4.13 Reading the virtual coordinate plane

Before extracting the actual text into a document, we had to read them to get the texts in the correct order. For that I read the texts from top to bottom left to right and separated them into lines as in the original pdf. I ensured every space as it is and sometimes, we had to do a calculation to detect spaces between Tj s that cannot be recognized by the reader itself and recognized them through a calculation and added them to the extracted text.

4.14 Extracting text to an xml document

Finally, I wrote the extracted texts content to an xml document using xml writer. Here I added tags to recognize the pages and rotation angle and no of images on each page [28] [29].

5 RESULTS AND EVALUATION

When we evaluate a PDF extraction tool, we must focus about some properties of PDF document. Detecting text from PDF file is difficult task. Because unlike other digital documents, PDF is a layout-based format that defines the fonts and locations of the individual characters instead of the text semantic units and their role in the file. As an example, conforming PDF reader need to determine whether the given text is from body or from footer of the PDF file.

In this evaluation I have primary focused about following semantic information for the outputs that generated by proposed system.

- Word identification

This is a very crucial aspect for this application. Because this tool is extracting text content for optimize search queries on ERP system's document archives. If a word does not detect accurately means it cannot find in searches.

- Word order

This means the right reading order of the words. In the future this tool planning to integrate with document mapping tool. For that application word order is a very important aspect.

- Paragraph boundaries

This means the ability to identify beginning and end of a paragraph. This feature is also very important for the document mapping tool which is planning to develop in near future.

- Semantic Roles

PDF file texts representing various semantic roles such as title, body text, formulas, figures. For the search application this tool going to integrate, it might also be useful to know whether a searched word occurs in the body text or in the caption of a figure [30].

By considering above semantic information, to assess the output file on this tool I have identified four aspects:

1. Paragraph boundaries
2. Distinction of body text and non-body text
3. Reading order
4. Word boundaries

After evaluating multiple output files against above four aspect, we can consider this tool as a successful method to extract text content from a given PDF file.

Get the balance right
Nudie Jeans is based on an idea. This idea is in turn composed of several concepts, beliefs, and a good portion of old fashioned fighting spirit.

Invoice

Order no 1000031729
Customer 10000584
Invoice no 101904954
Order date 19-11-09



Reverse charge, Intra-Community supply of goods.

Invoice no 101904954	Order no 1000031729	Customer 10000584	Delivery no 1001025788
Your order no	Your reference Simone Pleus	Our reference 1744012	Sales representative Mesut Anilciak
Payment terms 30 Days net	Delivery terms DAP	Delivery method 21	Your VAT reg no

Style no	Name	Qty	Price	Discount	Amount
131613	Daniel Logo Tee				
Taric Co	6109100010				
	Made In	IN			
	XXS XS S M L XL XXL	00001	16,00	0,00	192,00
Turquoise					
	1 2 2 1				
Turmeric					
	1 2 2 1				



186,24
-5,76

VAT

VAT code	Description	VAT (%)	Tax base	Tax amount
30		0,00	186,24	

TOTAL

Item value 186,24
TOTAL EUR 186,24



Figure 33: sample input 1

```

<?xml version="1.0" encoding="utf-8"?>
<pages>
  <MetaData>
    No MetaData was found.
  </MetaData>
  <page pageNumber="1">
    <content contentId="11">
      <Section Rotation="0">
        Page1(1)
        Invoice
        Order no Invoice no
        Get the balance right
        1000031729 101904954
        Nudie Jeans is based on an idea. This idea is in turn
        Customer Order date
        composed of several concepts, beliefs, and a good portion of
        10000584 18-11-09
        old fashioned fighting spirit.
        10000584
        Gruene Wiese-Wittenbrink Pleus
        Gruene Wiese-Wittenbrink Pleus
        Sppiekerhof 29
        Sppiekerhof 29
        48143 MÜNSTER
        48143 MÜNSTER
        Reverse charge, intra-Community supply of goods.
        Invoice no Order no Customer Delivery no
        101904954 1000031729 10000584 1001025788
        Your order no Your reference Our reference Sales representative
        Simone Pleus 1744912 Mesut Anliacik
        Payment terms Delivery terms Delivery method Your VAT reg no
        30 Days net DAP 21

        Style no Name Qty PriceDiscountAmount
        131613 Daniel Logo Tee
        Taric Co 6109100010 Made in IN 00001 16,00 0,00 192,00
        XXS XS S M L XL XXL
        Turquoise 1 2 2 1
        Turmeric 1 2 2 1

        186,24
        -5,76
        VAT
        VAT codeDescription VAT (%) Tax basis Tax
        amount
        30 0,00 186,24

        TOTAL
        Item value 186,24
        TOTAL EUR 186,24
        Nudie Marketing Jeans AB Nudie Marketing Jeans AB IBAN: SWIFTCODE:
        Västra Hamngatan 6 Västra Hamngatan 6 SE8960000000000037561898 HANDSESS
        411 17GÖTEBORG 411 17GÖTEBORG Account no. VAT no
        Org no. Tel. +46 10 15 15 600 SE556628927701
        55662892 Email.info@nudiejeans.com BG
      </Section>
      <Section Rotation="90">

    </Section>
    <Image ImageNo="0">
    </Image>
    <Image ImageNo="1">
    </Image>
  </content>
</page>
</pages>

```

I have extracted the contents of 100 sample pdf invoices to visualize the accuracy of this tool. I have got word count from the base file and from the output file. Table 9 shows the word count comparison between the base file and output file.

Table 9: Extracted word percentage

File name	Base file word count	Output file word count	Extracted words percentage
Sample File 001	140	133	95%
Sample File 002	142	129	91%
Sample File 003	279	265	95%
Sample File 004	468	429	92%
Sample File 005	372	372	100%
Sample File 006	178	174	98%
Sample File 007	145	131	90%
Sample File 008	172	157	91%
Sample File 009	165	153	93%
Sample File 010	374	369	99%
Sample File 011	686	678	99%
Sample File 012	206	199	97%
Sample File 013	133	128	96%
Sample File 014	102	97	95%
Sample File 015	159	157	99%
Sample File 016	335	322	96%
Sample File 017	267	240	90%
Sample File 018	212	191	90%
Sample File 019	336	336	100%
Sample File 020	156	145	93%
Sample File 021	418	376	90%
Sample File 022	134	129	96%
Sample File 023	260	252	97%
Sample File 024	249	244	98%

Sample File 025	404	368	91%
Sample File 026	384	342	89%
Sample File 027	413	396	96%
Sample File 028	154	136	88%
Sample File 029	357	332	93%
Sample File 030	214	201	94%
Sample File 031	175	165	94%
Sample File 032	111	103	93%
Sample File 033	376	372	99%
Sample File 034	351	351	100%
Sample File 035	382	363	95%
Sample File 036	389	350	90%
Sample File 037	249	244	98%
Sample File 038	342	318	93%
Sample File 039	162	159	98%
Sample File 040	184	164	89%
Sample File 041	340	313	92%
Sample File 042	208	187	90%
Sample File 043	396	368	93%
Sample File 044	192	182	95%
Sample File 045	220	220	100%
Sample File 046	324	295	91%
Sample File 047	169	162	96%
Sample File 048	268	261	97%
Sample File 049	395	356	90%
Sample File 050	185	179	97%
Sample File 051	382	336	88%
Sample File 052	398	366	92%
Sample File 053	298	289	97%
Sample File 054	134	134	100%

Sample File 055	206	192	93%
Sample File 056	397	361	91%
Sample File 057	139	128	92%
Sample File 058	269	247	92%
Sample File 059	286	269	94%
Sample File 060	254	249	98%
Sample File 061	215	194	90%
Sample File 062	331	328	99%
Sample File 063	402	366	91%
Sample File 064	348	317	91%
Sample File 065	127	112	88%
Sample File 066	115	104	90%
Sample File 067	401	357	89%
Sample File 068	365	336	92%
Sample File 069	193	176	91%
Sample File 070	284	273	96%
Sample File 071	367	330	90%
Sample File 072	345	335	97%
Sample File 073	194	164	85%
Sample File 074	331	301	91%
Sample File 075	406	361	89%
Sample File 076	415	369	89%
Sample File 077	140	132	94%
Sample File 078	373	358	96%
Sample File 079	418	397	95%
Sample File 080	207	203	98%
Sample File 081	311	292	94%
Sample File 082	210	197	94%
Sample File 083	273	265	97%
Sample File 084	270	251	93%

Sample File 085	341	324	95%
Sample File 086	246	244	99%
Sample File 087	279	257	92%
Sample File 088	168	161	96%
Sample File 089	401	389	97%
Sample File 090	261	258	99%
Sample File 091	251	238	95%
Sample File 092	182	175	96%
Sample File 093	199	181	91%
Sample File 094	130	129	99%
Sample File 095	234	232	99%
Sample File 096	223	207	93%
Sample File 097	419	373	89%
Sample File 098	338	328	97%
Sample File 099	276	271	98%
Sample File 100	120	109	91%

Table 10: Words percentage vs Number of files

Extracted words percentage	Number of Files
85	1
86	0
87	0
88	3
89	6
90	10
91	11
92	7
93	9
94	6

95	8
96	9
97	9
98	7
99	9
100	5

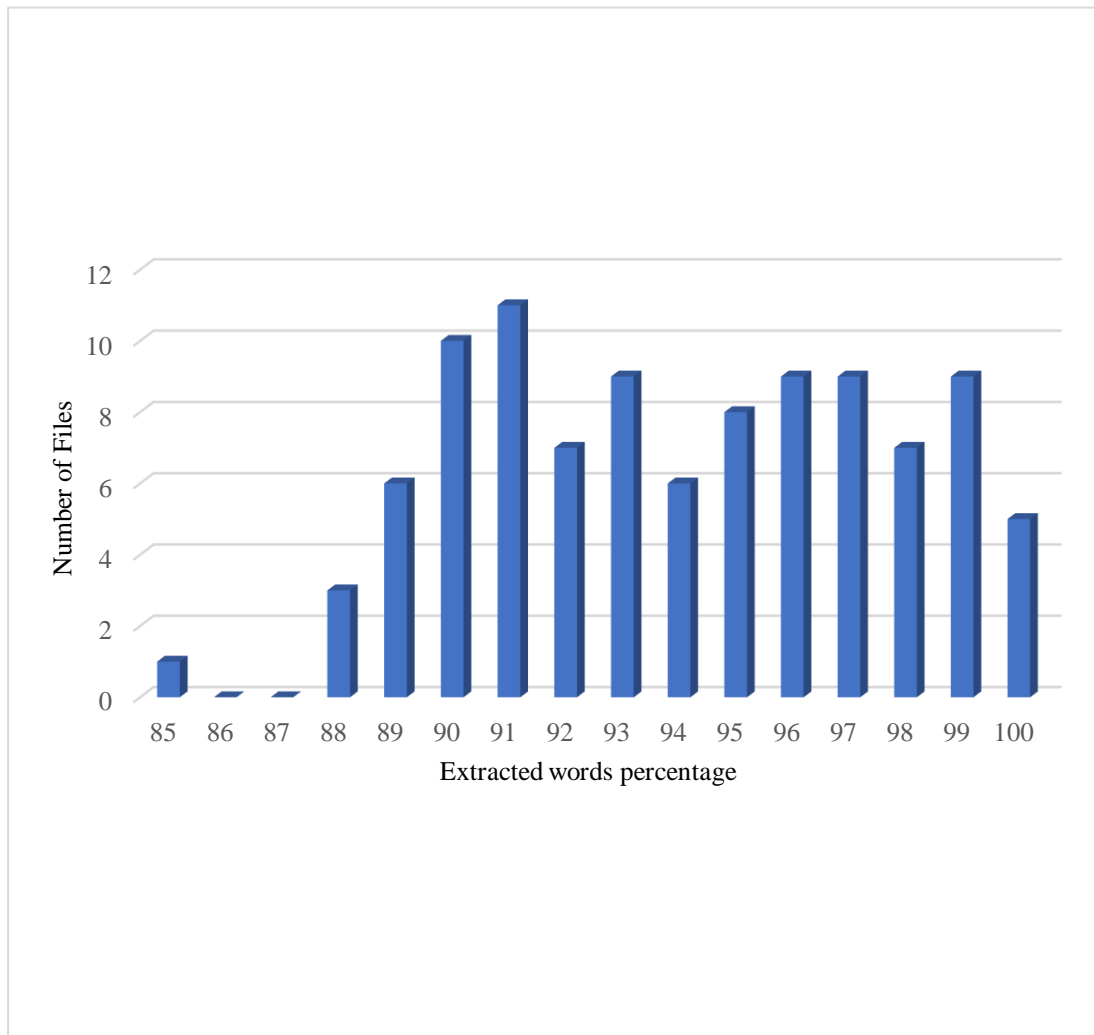


Figure 34: Extracted word percentage

6 CONCLUSION

6.1 Summary

A large number of communications between businesses currently using electronic documents. Portable document format (PDF) is the mostly using format for electronic documents. When dealing with PDF documents, indexing of documents is an important feature in this domain. Using document tagging we can group document sets in smaller subsets of similar correspondences. Using that we can boost document search engine efficiency by decreasing the search space. But if we can extract information from PDF files that will allows more structured search queries.

Detecting text from PDF file is difficult task. Because unlike other digital documents, PDF is a layout-based format that defines the fonts and locations of the individual characters instead of the text semantic units and their role in the file. In this research I have introduced a text content extraction method from PDF file. We can use it for allow more structured search queries on large document archives in output management systems typically work with world leading ERP systems.

When developing the extraction tool, I have mainly considered on four aspects. First one is ‘word identification’. This is a very important for an application like search. Because if word is not identified correctly that word is not found. Secondly ‘word order’ which is specify the right reading order of the words. Another one is ‘paragraph boundaries’ that determine the beginning and end of a paragraph. Last one is ‘semantic roles’ and it is used to separate the body text from the rest.

First, I have extracted the last xref offset in order to build the cross-reference table. Then xref objects were tokenized and the object ids and the relevant offsets were extracted. A Cross Reference Table contains all the object ids and their relevant offsets. Whenever the pdf is updated or modified a new xref with only the changed object ids and offsets is built.

To build the pages tree I extracted the root object from the trailer. A page object contains references to content objects where the text contents are stored. By checking all those object references I created a tree data structure for processing each page. In order to make sure the order of pages and contents, I used Depth First Traversal method to process the tree and built a list of content object ids in the correct order for further processing.

To further processing I had to tokenize the text stream of the PDF byte array into several tokens to match each object to a fixed pattern. For building these tokens I used many tokenizers such and xref tokenizer to tokenize the xref objects, Object tokenizer to tokenize object tokens, Text tokenizer to tokenize texts, Cmap tokenizer to tokenize cmap streams.

The binary code of a pdf is always object oriented. These objects needed to be constructed. Tokenizer was responsible for providing predefined token by token, as for the need of token handler. The token handler had to process the receiving tokens and use the object builder factory to construct corresponding objects.

I have implemented an object builder to create objects. “Factory method” is implemented for object builder. There is a builder factory for providing the specific object builder for creating pdf objects. Token Handler was used by the object builders. After the tokens are being handled, the object builder class will analyse the token list and build the particular object by using a builder of the Object Builder Factory.

To process content objects each content object is read using object reader by traversing the page tree in depth first traversal methodology. Then the streams which are encoded are tokenized and stored in stream tokens objects. Then they are decoded using the relevant decoder.

After extracting glyphs and relevant coordinates we had to develop a virtual coordinate plane to place them. For this I designed a custom data structure with linked lists where a page is a node and I used glyph objects to store the glyph details and stored them in a sorted dictionary where the y coordinate is the key and another sorted dictionary with x coordinate as the key and the glyph object as the value as the value of the main sorted dictionary.

Before extracting the actual text into a document, we had to read them to get the texts in the correct order. For that I read the texts from top to bottom left to right and separated them into lines as in the original pdf. Finally, I wrote the extracted texts content to an xml document using xml writer. Here I added tags to recognize the pages and rotation angle and no of images on each page.

After implementing this text content extraction tool, I have evaluated output files from the system by following guidelines that are suggested by Hannah Bast and Claudius Korzen in their research paper “A Benchmark and Evaluation for Text Extraction from PDF” [14]. It evaluates tool quality by considering four aspects which are paragraph boundaries, distinction of body text and non-body text, reading order and word boundaries.

- Research Contribution

The principal research contribution of this thesis is to support advanced search queries on business document archives by extracting business document contents on PDF/A files.

6.2 Future Works

This research primarily focused about allow more structured search queries on large PDF document archives by extracting text content from PDF files. Most of the time search queries only take a single point of view. it is only check words or phrases that appear in documents.

We can improve this tool by indexing a PDF file using different points of view.

a) General and syntactical viewpoint

We can capture general aspects, as well as details about representation and structure medium and type.

b) Domain viewpoint

In this viewpoint it considers about conceptual description of a domain.

c) Semantic viewpoint

In this viewpoint, semantic aspects of the segments are addressed, in which the description type and scope are preserved. Type of description represents the view from which a segment was written, as an example a structural description, refined in component description, component location description, etc.

d) Instructional viewpoint

Instructional aspects are considered in this viewpoint, with a focus on maintenance training. Learning goal, instructional strategy, instructional activity, knowledge type, instructor and learner actions and material use, each with its own collection of possible values will consist of an instructional description of a segment.

6.3 Limitations

This tool is not supporting for large files. Typically, large files cannot load into the memory as complete file. To extract data from large file we must load file as segments and need to process segment by segment. This kind of a process does not support with current implementation.

A PDF file can contain text with different styles and with different orientations. But data extraction method on this proposed solution having some issues when it is trying to extract text with different orientations. But researcher primarily focused on business archive domain and business documents such as receipts, invoices or quotations are rarely containing text with different orientation.

Some documents limit access to a PDF by setting passwords and by restricting certain features, such as printing and editing. Some of them are encrypting the document and the document metadata. Others are encrypting the contents of a document but still allows search engines access to the document metadata. The tool is not supporting for this kind of documents. By scanning header section of a PDF file, can determine whether the file contain encrypted data or not. Proposed tool will skip the data extraction if it found such a content.

Images are identifying on this extracting method but still it is not extracting the content of that image. Proposed tool can identify an image file and it will flag such a content. It will track images on the PDF file and include a tag to the output XML. Since this research targeting to support for search queries on business document archives, extraction of image content is not an important at the moment.

7 REFERENCES

- [1] H. Chao and J. Fan, "Layout and Content Extraction for PDF Documents," *document analysis systems*, pp. 213-224, 2004.
- [2] K. Hadjar, M. Rigamonti, D. Lalanne and R. Ingold, "Xed: a new tool for extracting hidden structures from electronic documents," in *First International Workshop on Document Image Analysis for Libraries, 2004. Proceedings.*, 2004.
- [3] E. Wustner, T. Hotzel and P. Buxmann, "Converting business documents: a classification of problems and solutions using XML/XSLT," in *Proceedings Fourth IEEE International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS 2002)*, 2002.
- [4] A. Dengel and F. Dubiel, "Clustering and classification of document structure-a machine learning approach," in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, 1995.
- [5] A. Anjewierden and S. Kabel, "Automatic indexing of PDF documents with ontologies," , 2001.
- [6] A. Anjewierden, "AIDAS: incremental logical structure discovery in PDF documents," in *Proceedings of Sixth International Conference on Document Analysis and Recognition*, 2001.
- [7] H. L. Chieu, H. T. Ng and Y. K. Lee, "Closing the Gap: Learning-Based Information Extraction Rivaling Knowledge-Engineering Methods," in *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, 2003.
- [8] D. Esser, D. Schuster, K. Muthmann, M. Berger and A. Schill, "Automatic indexing of scanned documents: a layout-based approach," in *Document Recognition and Retrieval XIX*, 2012.

- [9] C. Ramakrishnan, A. Patnia, E. H. Hovy and G. A. P. C. Burns, "Layout-aware text extraction from full-text PDF of scientific articles," *Source Code for Biology and Medicine*, vol. 7, no. 1, pp. 7-7, 2012.
- [10] R. Futrelle, M. Shao, C. Cieslik and A. Grimes, "Extraction, layout analysis and classification of diagrams in PDF documents," in *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, 2003.
- [11] R. Futrelle, "Ambiguity in visual language theory and its role in diagram parsing," in *Proceedings 1999 IEEE Symposium on Visual Languages*, 1999.
- [12] R. Futrelle and N. Nikolakis, "Efficient analysis of complex diagrams using constraint-based parsing," in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, 1995.
- [13] S.-H. Lin and J.-M. Ho, "Discovering informative content blocks from Web documents," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002.
- [14] H. Bast and C. Korzen, "A benchmark and evaluation for text extraction from PDF," in *Proceedings of the 17th ACM/IEEE Joint Conference on Digital Libraries*, 2017.
- [15] W. Kehong, "Optimized hierarchy clustering based extraction for logical document structures," *Journal of Tsinghua University*, 2005.
- [16] P. N. Smith and D. F. Brailsford, "Towards structured, block-based PDF," , 1995.
- [17] T. Padova, Adobe Acrobat 7 PDF Bible, 2001.
- [18] R. Cohn, Portable Document Format Reference Manual, 1993.
- [19] T. Joachims, Learning to Classify Text Using Support Vector Machines, 2002.

- [20] A. Jain and B. Yu, "Document representation and its application to page decomposition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 3, pp. 294-308, 1998.
- [21] T. Hu and R. Ingold, "A mixed approach toward an efficient logical structure recognition from document images," *Electronic Publishing*, vol. 6, pp. 457-468, 1993.
- [22] J. Fan, "Text extraction via an edge-bounded averaging and a parametric character model," in *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, 2003.
- [23] M. Lipinski, K. Yao, C. Breiting, J. Beel and B. Gipp, "Evaluation of header metadata extraction approaches and tools for scientific PDF documents," in *Proceedings of the 13th ACM/IEEE-CS joint conference on Digital libraries*, 2013.
- [24] Y. Wang, I. T. Phillips and R. M. Haralick, "A Study on the Document Zone Content Classification Problem," *document analysis systems*, pp. 212-223, 2002.
- [25] L. Zhang, Y. Pan and T. Zhang, "Focused named entity recognition using machine learning," in *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, 2004.
- [26] E. Saund, "Scientific challenges underlying production document processing," in *Proceedings of SPIE, the International Society for Optical Engineering*, 2011.
- [27] R. Futrelle, "Strategies for diagram understanding: generalized equivalence, spatial/object pyramids and animate vision," in *[1990] Proceedings. 10th International Conference on Pattern Recognition*, 1990.
- [28] E. A. El-Kwae and K. H. Atmakuri, "Document image representation using XML technologies," in *Document Recognition and Retrieval IX*, 2001.

- [29] H. Déjean and J.-L. Meunier, “A system for converting PDF documents into structured XML format,” *document analysis systems*, pp. 129-140, 2006.
- [30] D. Tkaczyk, A. Czezko, K. Rusek, L. Bolikowski and R. Bogacewicz, “GROTOAP: ground truth for open access publications,” in *Proceedings of the 12th ACM/IEEE-CS joint conference on Digital Libraries*, 2012.

APPENDIX

- Extracted data from sample files

```
<?xml version="1.0" encoding="utf-8"?>
<pages>
  <MetaData>
    No MetaData was found.</MetaData>
  <page pageNumber="1">
    <content contentId="9">
      <Section Rotation="0">
        ORDERBEKRÄFTELSE Sida1(1)
        Genius Output. Simplified. Ordernr Kund
        www.accure.eu 103519 1001826
        Orderdatum Datum
        Leveransadress Kund
        19-10-24
        European Spallation Source ERICDustin Sverige AB
        A. ESS Head Office Gate E/F03 BOX 1194
        Odarsliögsvägen 113 131 27 Nacka Strand
        22484 Lund Sverige
        Sverige
        Ert ordernr Er referens Referens Vår referens
        468557796 Susanne Jörnberg Susanne Jörnberg
        Betaln villkor Leveransvillkor Leveranssätt Orderdatum
        45 dagar netto FOB Ftgpaket 16:00 19-10-24
        RADER
        Rad Artikelnr Pris Kvt Enh Belopp
        10 30CYS15800 12.986,281,00 Styck 12.986,28
        CTO ThinkStation P330, Intel Core i7-9700K
        20 Frakter 30,001,00 Styck 30,00
        FRAKTER

        SUMMA
        Belopp
        TotalInterestAmount
        Summa
        Fakturatotal, exkl moms
        TotalItemAmount
        Rabatt
        Momsbelopp
        Avrundning
        Avgifter
        ATT BETALA SEK
        Pedab AB Pedab AB IBAN: SWIFT:
        Pipers väg 28 Pipers väg 28 SE896000000000037561HANDSESS
        170SOLNA 170SOLNA 898
        73 73 Konto VAT
        Orgnr. Tel.+46 33 700 15 00 SE556628927701
        55662892 Epoinfo@pedab.se Bg.
        st.
      </Section>
    <Section Rotation="90">
    </Section>
    <Image ImageNo="0">
    </Image>
    <Image ImageNo="1">
    </Image>
  </content>
</page>
</pages>
```

```

<?xml version="1.0" encoding="utf-8"?>
<pages>
  <MetaData>
    No MetaData was found.</MetaData>
  <page pageNumber="1">
    <content contentId="7">
      <Section Rotation="0">
        ORDER CONFIRMATION Page1(1

        )
        Genius Output. Simplified. Order no Customer
        www.accure.eu 0010000001Y20000
        Order dateDate
        02-28-10 12-12-19
        Customer address
        Delivery address
        Industry Customer - Mississippi
        Industry Customer - Mississippi
        Industry Customer- Adr line 1
        Industry Customer- Adr line 1
        Industry Customer- Adr line 2
        Industry Customer- Adr line 2
        Jackson MS ZIP
        Jackson MS ZIP
        United States
        United States
        Your order noYour referenceReference Sales
        representative
        Your referenceFood & BeverageTemplate
        deliveryaddressAdmin User Salesperson
        Payment termsDelivery termsDelivery methodOrder date
        30 days net Cost Insurance003 02-28-10
        and Freight
        LINES
        Ln Item no Price QtyUnit Amount
        1 Y22001 31.08 EA 559.44
        Fresh Turkey - Catchweight
        Fresh Turkey - Catchweight Item, Purchased for
        resale
        2 Y10033 37.00 CAS 370.00
        Product - Simplified Costing
        Product - Simplified Costing

        SUMMARY
        Total 929.44
        Total 929.44
        TOTAL USD 929.44

      </Section>
    </content>
  </page>
</pages>

```

```

<?xml version="1.0" encoding="utf-8"?>
<pages>
  <MetaData>
    No MetaData was found.</MetaData>
  <page pageNumber="1">
    <content contentId="10">
      <Section Rotation="0">
        Page 1(2)
        Invoice
        Order no Invoice no
        Finds your next business
        1000031729 551955555
        Customer Order date
        10000584 18-11-09
        10000584
        Gruene Wiese-Wittenbrink Pleus
        Gruene Wiese-Wittenbrink Pleus
        Sppiekerhof 29
        Sppiekerhof 29
        48143 BORÅS
        48143 BORÅS
        Reverse charge, intra-Community supply of goods.
        Invoice no Order no Customer Delivery no
        551955555 1000031729 10000584 1001025788
        Your order no Your reference Our reference Sales representative
        Simone Pleus 1744912 Mesut Anliacik
        Payment terms Delivery terms Delivery method Your VAT reg no
        30 Days net DAP 21

        Style no Name Qty PriceDiscountAmount
        131613 Daniel Logo Tee
        Tarc Co 6109100010 Made in IN 00001 16,00 0,00 192,00
        XXS XS S M L XL XXL
        Turquoise 1 2 2 1
        Turmeric 1 2 2 1

        SUMMARY
        Total 186,24
        Charges -5,76
        VAT
        VAT code Description VAT (%)Tax basisTax amount
        30 0,00 186,24

        TOTAL
        Item value 186,24
        TOTAL EUR 186,24

        Style no Name Qty PriceDiscountAmount
        131613 Daniel Logo Tee
        Tarc Co 6109100010 Made in IN 00001 16,00 0,00 192,00
        Pedab AB Pedab AB IBAN: SWIFTCODE:
        Pipers väg 28 Pipers väg 28 SE8960000000000000037561898 HANDSESS
        170 73SOLNA 170 73SOLNA Account no. VAT no
        Org no. Tel. +46 33 700 15 00 SE556628927701
        55662892 Email.info@pedab.se BG
      </Section>
    <Section Rotation="90">
    </Section>
    <Image ImageNo="0">
    </Image>
    <Image ImageNo="1">
    </Image>
  </content>
</page>
<page pageNumber="2">
  <content contentId="13">
    <Section Rotation="0">
      Order no Invoice no
      1000031729 551955555
      Customer Order date
      10000584 18-11-09
      Finds your next business Page 2(2)
      Invoice
    </Section>
  </content>
</page>

```

Style no	Name	Qty	Price	Discount	Amount
	XXS XS S M L XL XXL				
Turquoise	1 2 2 1				
Turmeric	1 2 2 1				

Pedab AB Pedab AB IBAN: SWIFTCODE:
Pipers väg 28 Pipers väg 28 SE8960000000000037561898 HANDSESS
170 73SOLNA 170 73SOLNA Account no. VAT no
Org no. Tel. +46 33 700 15 00 SE556628927701
55662892 Email.info@pedab.se BG

```
</Section>  
<Section Rotation="90">  
</Section>  
<Image ImageNo="0">  
</Image>  
</content>  
</page>  
</page>  
</pages>
```