

1.0 Overview

This document provides a guide to using *PDF Java Toolkit* to extract text from PDF files.

Text extraction refers to a set of APIs that enable users to find and extract text from within PDF documents. The basic unit of text is a word and the text extraction feature needs to provide for the logical delination of text into words. The list of words and related information need to be made available to the user.

Information about words includes location, font, bounding box, and character widths.

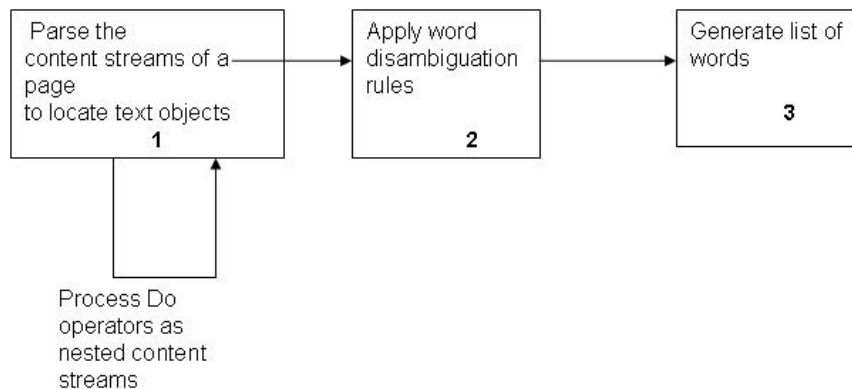
The purpose of the text extraction feature is to provide users with the following functionality:

- The ability to find text on a page known to be in a certain location.
- The ability to search and index PDF content.
- The ability to repurpose PDF text content.
- Provide a way for search engines to deal with PDF with content that may be more complex than simple text.

2.0 Model

PDF Java Toolkit uses a text extraction model that is loosely based on Acrobat's Wordy engine. Text extraction consists of three phases as shown below.

FIGURE 1. Text Extraction Phases



Form XObjects are treated as nested content streams. To detect them, we look for the **Do** operator in a page's content stream.

Each of these areas has its own list of features and issues which are discussed in the following sections.

1. [Determining Glyph Encoding](#)
2. [Sorting and Packaging the List of Words](#)

3.0 Text Encodings

The text extraction APIs generate a list of words in a PDF document that use Roman or Unicode encoding. Text extraction works for both non-structured and tagged PDF.

Since *PDF Java Toolkit* is Java-based, and Java provides native support for Unicode, the APIs provide the extracted text in Unicode format.

Beyond this, text extraction is designed to handle text in any standard encodings. Users may extract text from the entire document or on any element of the structure tree.

3.1 Determining Glyph Encoding

Determining the glyph encoding for non-tagged PDF is a somewhat complicated process.. The basic outline for determining glyph encoding is given in Section 5.9.1 (“Mapping Character Codes to Unicode Values”) of the *PDF Reference*. Basically, conversion to Unicode can be done if the font’s characters are identified using a character set that is known to the library. This character identification can occur if either the font uses a standard named encoding, or the characters in the font are identified by standard character names or CIDs in a well-known collection.

For Tagged PDF, the extraction of character informaton is somewhat less ambiguous because Section 10.7.2 of PDF Reference dictates producers of tagged PDF should provide enough information to perform the Unicode mapping using according to one of the methods in Section 5.9.1. Additionally, Tagged PDF has information for dealing with artifacts (for example, hyphens) which aids word disambiguation.

The basic process is as specified in Section 5.9.1 of *PDF Reference*.

3.2 Treatment of Glyphs and Fonts

In general, glyphs whose names that are not recognizable and that don't have **/ToUnicode** entries cannot be converted to Unicode. See Appendix D, *PDF Rerence* for details of the character sets and encodings.

- All Fonts except Type3 fonts have a built-in encoding. The four different encodings possible are `StandardEncoding`, `MacRomanEncoding`, `WinAnsiEncoding` and `PDFDocEncoding`. This encoding may be overridden by populating the **Encoding** entry of a `PDFFont` Dictionary with an `EncodingDictionary`. (Table 5.11 - Pg 397)
- For text extraction, `StandardEncoding` is applied to simple fonts that lack an encoding.

To map a character code to it's Unicode value for Simple fonts or character code to character identifier for Composite Fonts, *PDF Java Toolkit* uses the `CMap` file present in the **/ToUnicode** entry of the font dictionary. Section 5.9.2 Pg 442 explains the **/ToUnicode** `Cmaps` in detail.

Note: For this library, glyph names in `PDFDocEncoding` or Unicode are looked up only in range from U+0000 to U+00FF. Additionally, this library does not handle ligatures or perform character code guessing.

4.0 Word Extraction

4.1 Assumptions for word breaking

If two or more consecutive structure or marked-content sequences has an **ActualText** entry, they should be treated as if no word break is present between them.



In a non-structured document, pages are in reading order and reading order within a page is the drawing order of text operators on that page. In a structured document, the structure elements are in reading order and the reading order within an element is the order of occurrence in the element's **/Kids** array. Reading order within marked content sequences or form XObject referenced from a structure element is the drawing order of text operators in the sequence or form.

In structured PDF, words do not span structure elements. The first character of a structure element is the first character of the first word in that element. Similarly, the last character of a structure element is the last character of the last word in that element.

4.2 Word breaking characters

- A period or comma is a word breaking character except if it is preceded or followed by a numeral in which case it is included as part of a “numeral word.”
- Multiple non-whitespace word-breaking characters are themselves considered as a word. For example, the string “!@#% ^&*” contains two words.
- Multiple whitespace characters have no additional word breaking meaning.
- The hyphen character is *not* a word-breaking character.

4.3 Line endings

A new line of text starts a new word unless the previous line ended with a hyphen or endash. More rigorously stated, a character that is not a hyphen or endash and which is followed by another character whose baseline is not “aligned” with it, denotes the end of a word.

4.4 Baseline alignment

Two characters are considered to have aligned baselines if the second character's origin is located within 10% of the height of the larger of the two characters in the direction perpendicular to the baseline of the first character.

In general, large fluctuations in character spacing, fonts, color, size, etc. are ignored for the purposes of identifying words. Additionally, no word breaking information can be gleaned from the usage pattern of the various text operators.

4.5 Sorting and Packaging the List of Words

You can get a list of words on a page by using a `Word` object and following these guidelines. See [Example: Text Extraction](#) for a code listing.

- Iterate over a `Word` list.
- Get a Unicode string representing the word.
- Get the page number where the word is located.
- Get a list of bounding quads for the word.
- Iterate over the list of quads.
- Convert quads to strings.

5.0 Text Extraction from PDF Files

Adobe PDF Java Toolkit supports text extraction from PDF files. Text extraction makes it possible to save the PDF source as plain text.

Text extraction draws from two distinct areas of the PDF document.

- From form XObjects in a page's content stream.
- From form fields and Annotations.

PDF Java Toolkit presents text as iterable Java objects. To get the text, user applications are required to take the following steps.

1. Iterate over the objects.
2. Retrieve the text.
3. Save it in the desired format.

The Text Extraction from XObjects example shows how to implement these steps.

5.1 Text Extraction from form XObjects in a page's content stream

This section provides a discussion of text objects present in Form XObjects. A Form XObject is a PDF content stream that is a self-contained description of any sequence of graphics objects (including path objects, text objects, and sampled images).

5.1.1 Example: Content stream containing an XObject.

```
q
0.8987885 -0.4383698 0.4383698 0.8987885 209.2356262
460.8054199 cm

/GS0 gs

/Fm0 Do

Q

q
```

5.1.2 Example: XObject Fm0 in the resource dictionary

```
<< /Type /XObject

/Subtype /Form

/FormType 1

/BBox [0 0 1000 1000]

/Matrix [1 0 0 1 0 0]

/Resources << /ProcSet [/PDF] >>

/Length 58

>>

stream

0.25 0.333 1 rg

0 i

BT
```



```

/TT0 1 Tf
0 Tc 0 Tw 0 Ts 100 Tz 0 Tr 24 0 0 24 0 -24 Tm
(This is a Background.)Tj
ET
Endstream

```

5.1.3 Example: Text Extraction

```

try
{
    //Create a new text extractor for the pdfDocument.

    TextExtractor text_extractor = TextExtractor.newInstance(pdfDoc);

    /** Get the list of words in the document. Alternatively, you can call getPage-
    WordList(pagenumber) to get the words for a particular page.

        A null word list is returned if no words are found.

    **/

    List docWordList = text_extractor.getDocumentWordList();

    if (docWordList != null)
    {
        //process the document word list

        Iterator wordIter = docWordList.iterator();

        while (wordIter.hasNext())
        {
            /*

            * Using a Word object, you can perform the following tasks:

            1) Get a Unicode string representing the word.

            2) Get the page number where the word is located.

            3) Get a list of bounding quads for the word.

```



```
*/
```

```
/**
```

The following code extracts text from a PDF document.

```
*/
```

```
Word word = (Word)wordIter.next();
```

```
String str = word.toString();
```

```
List quads = word.getBoundingQuads();
```

```
if (quads != null)
```

```
{
```

```
    Iterator quadIter = quads.iterator();
```

```
    while (quadIter.hasNext())
```

```
    {
```

```
        String quadStr = ( (ASQuad) quadIter.next()).toString();
```

```
        System.out.println(quadStr);
```

```
    }
```

```
}
```

```
}
```

```
}
```

```
}
```

```
catch (Exception e)
```

```
{
```

```
    e.printStackTrace();
```

```
}
```

5.2 Text extraction from Form Fields and Annotations

PDF Java Toolkit does not provide "text extraction services" for annotations and form fields. Text can be obtained from the appropriate dictionary fields. Quads are not computed and the word content is not run through the disambiguation algorithm. The position information available is limited to the "location" dictionary entry of the field/annot on the page.



5.2.1 Text Extraction from Annotations

An annotation associates an object such as a note, sound, or movie with a location on a page of a PDF document,

The optional **Annots** entry in a page object holds an array of annotation dictionaries, each representing an annotation associated with the given page. A given annotation dictionary may be referenced from the **Annots** array of only one page.

The entries that are relevant in the context of Text Extraction are listed in the following table.

TABLE 1. Annots entries relevant to Text Extraction

Key	Type	Value
Type	Name	(Optional)The type of PDF object that this dictionary describes; if present, must be Annot for an annotation dictionary.
Subtype	Name	(Required)The type of annotation that this dictionary describes.
Rect	Rectangle	(Required)Annotation rectangle defining the location of the annotation on the page in default user space units.
Contents	Text string	(Optional) Text to be displayed for the annotation or, if this type of annotation does not display text, an alternate description of the annotation's contents in human-readable form. In either case, this text is useful when extracting the document's contents in support of accessibility to users with disabilities or for other purposes.
M	Date or string	(Optional; PDF 1.1) The date and time when the annotation was most recently modified. Viewer applications should be prepared to accept and display a string in any format.
T	Text string	(Optional; PDF 1.1) The text label to be displayed in the title bar of the annotation's pop-up window when open and active. By convention, this entry identifies the user who added the annotation.
/V	Various	(Optional; inheritable) The field's value, whose format varies depending on the field type. See the descriptions of individual field types for further information.

5.2.2 Annotation Types

There are various types of Annotations. The Annotation's type is identified in the Annotation dictionary's **Subtype** entry. See [Section 8.4, "Annotations"](#) in *PDF Reference*.

Many annotation types are defined as markup annotations because they are used primarily to mark up PDF documents. These annotations have text that appears as part of the annotation.

Annotations can be broadly classified into Markup and Non-MarkUp annotations.

Markup Annotations: Markup annotations can be divided into the following groups:

- Free text annotations display text directly on the page. The annotation's **Contents** entry specifies the displayed text.
- Most other markup annotations have an associated pop-up window that may contain text. The annotation's **Contents** entry specifies the text to be displayed when the pop-up window is opened. These include text, line, square, circle, polygon, polyline, highlight, underline, squiggly-underline, strikeout, rubber stamp, caret, ink, and file attachment annotations.
- Sound annotations do not have a pop-up window but may also have associated text specified by the **Contents** entry.
- A subset of markup annotations is called text markup annotations.

Non-Markup Annotations: The pop-up annotation type typically does not appear by itself; it is associated with a markup annotation that uses it to display text.

Note: The **Contents** entry for a pop-up annotation is relevant only if it has no parent. In that case, it represents the text of the annotation.

For all other annotation types (Link, Movie, Widget, PrinterMark, and TrapNet), the **Contents** entry provides an alternate representation of the annotation's contents in human-readable form, which is useful when extracting the document's contents in support of accessibility to users with disabilities or for other purposes

5.2.3 Text Extraction from Form Fields

For form fields, the contents saved by Acrobat indicated by the */V* entry.

TABLE 2. Form Fields entries relevant to Text Extraction

Key	Type	Value
<i>/V</i>	Various	(Optional; inheritable) The field's value, whose format varies depending on the field type. See the descriptions of individual field types for further information.

To get the */V* entry using the *PDF Java Toolkit* methods, use the `PDFField.getValueList()` method as in the following example.

5.2.4 Example: Field Value Extraction

```
PDFInteractiveForm iforms = pdfDocument.getInteractiveForm();
```

```
    // Get Field iterator
```

```
    Iterator fieldIterator = iforms.iterator();
```

```
    // Iterate over form fields
```

```
    while(fieldIterator.hasNext())
```

```
    {
```

```
        // Get next field
```

```
        PDFField pdfField = (PDFField)(fieldIterator.next());
```

```
        // The value of a field can be a list of strings.
```

```
        // In most cases this is just a name or a single string.
```

```
        //In any case, the list of values is always represented as a list of strings.
```

```
        List valueList = (pdfField).getValueList(); // get the list of values
```




```
if (valueList != null)
{
    Iterator valueIterator = valueList.iterator(); // get the iterator of the list

    while (valueIterator.hasNext())
    {
        String value = (String) valueIterator.next();
    }
}
}
```

Note: Refer to the *PDF Java Toolkit* javadoc for the `PDFField` and `PDFAnnotation` classes for more details.

6.0 Behavior of Acrobat when Saving Documents with Annotations.

We tested the Acrobat's *Save As plain/accessible text* operation on files belonging to each annotation type.

The observed behavior is shown in the following table.

TABLE 3. Behavior Comparison of Acrobat and PDF Java Toolkit

ID	Annotation Type	Format and Fields used for Acrobat "Save As" Accessible Text	Format and Fields used for Acrobat "Save" as Plain Text
1	Text	(Note comment <i>/T /M /Contents</i>)	<i>/Contents</i>
2	Free Text	(Text Box comment <i>/Contents</i>)	Ignored
3	Square	(Rectangle comment <i>/T /M /Contents</i>)	Ignored
4	Circle	(Oval comment <i>/T /M /Contents</i>)	Ignored
5	Polygon	(Polygon comment <i>/T /M /Contents</i>)	Ignored
6	Polyline	(Polygonal Line comment <i>/T /M /Contents</i>)	Ignored
7	Stamp	(Stamp comment <i>/T /M /Contents</i>)	Ignored
8	Ink	(Pencil comment <i>/T /M /Contents</i>)	Ignored
9	File Attachment	(File Attachment comment <i>/Contents</i>)	Ignored
10	Sound	(Sound Attachment <i>/Contents</i>)	Ignored
11	Widget	(text) <i>/V</i>	<i>/V</i>

6.1 Annotation Types ignored by Acrobat during Save operations

- Line
- Highlight
- Underline
- Squiggly
- StrikeOut

6.2 Annotation Types not supported by *Adobe PDF Java Toolkit*

- Caret
- Screen
- PrinterMark
- TrapNet
- WaterMark
- 3D

6.3 Relevant API's

From [TABLE 3.Behavior Comparison of Acrobat and PDF Java Toolkit](#) it is clear that the values that are used when saving a PDF document containing annotations are extracted from the following fields of the Annotation:

/Contents, /T and /M

In this example, you iterate over all the pages in the PDF document. For each page, you can enumerate the annots and extract the required information as shown below.

6.3.1 Example: Extract information from a list of annotations

```
// Get the list of annotations on the page.

PDFAnnotationList annotations =

    pdfPage.getAnnotationList();

// Get annotation iterator.

annotIterator = annotations.iterator();

while (annotIterator.hasNext())
{
    // Get the next annotation.

    PDFAnnotation pdfAnnotation =

        (PDFAnnotation) (annotIterator.next());

    //Get the /Contents entry as a String.

    String annotation_content =

        pdfAnnotation.getContents();
```



```
//Get the /M entry or modification date as an ASDate.

ASDate modification_date =

    pdfAnnotation.getModificationDate();


//Get the location of the annotation as a PDFRectangle.

PDFRectangle annot_location =

    pdfAnnotation.getLocation();


/** You can get the Quad Points for all text markup and
link annotations. */

double[] annot_quadpts =

    pdfAnnotation.getQuadPoints()


//All Markup Annotations have the /T entry.

if(pdfAnnotation instanceof PDFAnnotationMarkup)

{

    //Get the /T entry as a string.

    String title = pdfAnnotation.getTitle();

}

}
```

7.0 Limitations

- Text extraction from group and reference XObjects are not supported.
- No support for caching of Words extracted from Form XObjects.
 - Existing limitations in the algorithm, prior to adding XObject support
 - No support for extracting text from /Actual Text entries.
 - No support for vertical writing mode.
 - No support for clipping paths.
 - No checking of the OCG associated with the content.
 - No support for any CMAPs other than identity (we don't have access to external CMAPs)
 - No checking for the effects of transparency on text

- No support for CID Keyed fonts
- Rotation not applied to bounding boxes
- Duplicate text runs with the same bounding box not removed.

8.0 References

See the following topics in the *PDF Reference 1.6* .

[Section 4.9, "Form XObjects";](#)

[Chapter 5, "Text";](#)

[Section 8.6 "Interactive Forms".](#)

