**Natural Language Processing for PDF/TIFF/Image Documents   
Computer Vision for Image Data  
SPLITTER Module  
High Precision PDF Page Splitting/OCR/Text Extraction  
Technical Specification, Gap v0.91**

# 1 Document

## 1.1 Document Overview

The document classifier contains the following primary classes, and their relationships:

* Document – This is the base class for the representation of a stored document. The constructor for the class object takes as parameters the stored path to the document, optionally a directory path for storing extracted pages and text, and optionally an event completion handler when processing the document asynchronously, and optionally a config parameter for configuring the NLP preprocessing.

document = Document(“/somedir/mydocument.pdf”, “/mypages/mydocument”)

The constructors calls the \_exists() and \_collate() private methods for the specified document.

* Page – This is a base class for the representation of an extracted page from the document. The document class contains a list (index) of the extracted pages as Page objects.

**Page (Base Class)**

**Page (Base Class)**

**Document (Base Class)**

**Page (Base Class)**

**List**

**Fig. 1a High Level view of Document Class Object Relationships**

## 1.2 Initializer (Constructor)

***Synopsis***

Document( document=**None**, dir=’./’, ehandler=**None**, config=**None**)

***Parameters***

document: if not None, a string that is either:

1. local path to document
2. remote path to document ((i.e., http[s]://….)

The document must be one of the following types: PDF. JPG, PNG, BMP or TIF

dir: The directory where to store the machine learning ready data.

ehandler: If not None, the processing of the images into machine learning ready data will be asynchronous, and the value of the parameter is the function (or method) that is the event handler when processing is complete. The event handler takes the form:

**def** myHandler(images):   
 # where images is the Images object that was preprocessed.

config: If not None, a list of one or more configuration settings as strings:

bare  
pos  
roman  
segment  
stem=gap|porter|Lancaster|snowball|lemma

***Usage***

When specified with no parameters, an empty Document object is created. The Document object may then be used to subsequent load (retrieve) previously stored preprocessed machine learning ready data (see load()).  
  
Otherwise, the document parameter must be specified. The document specified by the document parameter will be preprocessed according to the optional parameters and configuration settings.

By default, the document will be preprocessed as follows:

1. The document will be split into individual pages.
2. A Page object will be created for each page.
3. If the document (or page) is an image (e.g., scanned PDF), it will be OCR’d.
4. The digital text will be extracted from each page and stored in the Page object.
5. The text will be optionally segmented into regions if the configuration setting segment is specified.
6. The text from each page object will be preprocessed into machine learning ready data (see syntax module specification), according to the optional parameters and configuration settings.

The machine learning ready data will be stored on a per page basis in the directory specified by the parameter dir. The following files are created and stored:

<document><pageno>.<suffix>  
 <document><pageno>.txt  
 <document>.<pageno>.json

The <document> is the root name of the document, and <pageno> is the corresponding page number starting at page 1. The file ending in the original file suffix (<suffix>) is the split page. The file ending in the file suffix .txt is the extracted text. The file ending in the file suffix .json is the NLP preprocessed machine learning data stored in a JSON format.

If the ehandler parameter is not None, then the above will occur asynchronously, and when completed, the corresponding event handler will be called with the Document object passed as a parameter.

If the path to the document file is remote (i.e., starts with http), an HTTP request will be made to fetch the contents of the file from the remote location.

***Exceptions***

A TypeError is raised if the type of the parameter is not the expected type.

A AttributeError is raised if an invalid configuration setting is specified.

A FileNotFoundError is raised if the document file does not exist.

A IOError is raised if an error occurs reading in the document file.

## 1.3 Document Properties

### 1.3.1 document

***Synopsis***

# Getter  
path = document.document

# Setter  
document.document = path

***Usage***

When used as a getter the property returns the path to the document file.

When used as a setter the property specifies the path of the document file to preprocess into machine learning ready data (see initializer).

***Exceptions***

A TypeError is raised if the type of the parameter is not the expected type.

A FileNotFoundError is raised if the document file does not exist.

A IOError is raised if an error occurs reading in the document file.

### 1.3.2 name

***Synopsis***

# Getter  
root = document.name

***Usage***

When used as a getter the property returns the root name of the document file(e.g., /mydir/mydocument.pdf -> mydocument).

### 1.3.3 type

***Synopsis***

# Getter  
suffix = document.type

***Usage***

When used as a getter the property returns the file suffix of the document file (e.g., pdf).

The Document class contains the following properties:

* text – The raw extracted text; returned as a list, one entry per page.
* pages – Index of the page objects, one per page.
* dir – Location to store extracted pages and text.
* size – The byte size of the document.
* scanned – If document is a scanned PDF, TIFF facsimile or image captured.
* time – The elapsed time in seconds for processing the document.
* label – The document classification.
* bagOfWords – Bag of Word with number of occurrences per word in the document as an unordered dictionary.
* freqDist – A sorted list of tuples of number of occurrences per word in the document, sorted in descending frequency.
* termFreq – A sorted list of tuples of the percentage that word occurs in the document.

The document class contains the following static variables:

* RESOLUTION – The image resolution when converting PDF to PNG for OCR (default 300).

## 1.3 Document Overridden Operators

The following operators have their implementations (inherited from the base Object) class overridden:

* len() – The \_\_len\_\_() method is overridden to return the number of pages in the document.
* str() – The \_\_str\_\_() method is overridden to return the document classification (label).
* [] – The \_\_getitem\_\_() method is overridden to return the Page object at the specified index (i.e., page number – 1). The \_\_setitem\_\_() method is overridden to replace the Page object at the specified index (i.e., page number – 1).
* += The \_\_iadd\_\_() method is overridden to add a Page object (i.e., append).

## 1.4 Document Public Methods

The Document class contains of following public methods:

* load() – This method reloads the document pages from storage. Using the document name and storage path, the method locates each of the stored pages and reconstructs the Page and corresponding Words objects.

document.load(“<document\_name>”, “storage\_path”)

## 1.5 Document Private Methods

The Document class contains the following private methods:

* \_exists() – This method checks if the document exists at the specified stored path. If not, a FileNotFound exception is thrown.
* \_collate() – This method performs the collation task, which includes:
  + Determines the number of pages in the document.
  + Splits the document into individual pages, where each page is individually stored in the same format as the document. The pages are named as follows:

<name><pageno>.<suffix>

Each page is stored in the subdirectory specified by the property dir. If dir is None, then the page is stored in the same directory where program is ran; otherwise, if the subdirectory does not exist, it is created.

* + If the page is a scanned PDF page, the scanned image is extracted and saved as a PNG image. The PNG image is then OCR’d to convert to text.

<name><pageno>.png

* + If the page is a TIFF facsimile, the TIFF image is then OCR’d to convert to text.

<name><pageno>.tif

* + If the page is an image capture (e.g. camera capture), the captured image (e.g., JPG) is then OCR’d to convert to text.

<name><pageno>.jpg

* + Extracts the raw text from the page , where each page is individually stored in a raw text format. The pages are named as follows:

<name><pageno>.txt

Each page is stored in the subdirectory specified by the property dir. If dir is None, then the page is stored in the same directory where program is ran.

* + Create a Page object for each page and adds them to the pages index property.
  + If the document format is raw text, then:
    - Treats as a single page.
    - Stores only a single page text file.
  + If the document format is PDF, then page splitting and extraction of the raw text per page is done with the open source version of Ghostscript. If the document is a scanned PDF, the image is extracted and converted to PNG using Ghostscript and then OCR’d using open source Tesseract.
  + If the document format is TIFF, then page splitting is done with the open source Magick and then OCR’d using open source Tesseract.
* \_async() – This method performs asynchronous processing of the \_collate() function, when the optional ehandler parameter to the constructor is not None. When processing is completed, the ehandler parameter value is called as a function to signal completion of the processing, and the document object is passed as a parameter.

# 2 Page

## 2.1 Page Overview

The page classifier contains the following primary classes, and their relationships:

* Page – This is a base class for the representation of an extracted page from a document. The constructor for the class object takes optionally as parameters the stored path to the page, and the extracted raw text.

page = Page( ‘/mypages/page1.pdf’, ‘some text’)

* <Category>Page – This is a derived class for representation of an extracted page from a category specific document, such as a medical document.
* Words – This is a base class for representation of the text as NLP preprocessed list of words.

**Words (Base Class)**

**Page (Base Class)**

**<Category>Page (Derived Class)**

**Fig. 2a High Level view of Page Class Object Relationships**

## 2.2 Page Properties

The Page class contains the following properties:

* path – The path to the stored document.
* text – The raw extracted text.
* label – The page classification
* size – The byte size of the raw extracted text.
* pageno – The page number in the parent document
* words – The page text converted to a sequenced tokenized set of words which has been NLP preprocessed.
  + Acronym, Number, Dates, Date of Birth, Telephone, SSN, and Proper Names Detection
  + Punctuation Removal
  + Lowercased
  + Stopword Removal
  + Word Stemming
* bagOfWords – Bag of Word with number of occurrences per word in the page as an unordered dictionary.
* freqDist – A sorted list of tuples of number of occurrences per word in the page, sorted in descending frequency.
* termFreq - A sorted list of tuples of the percentage that word occurs in the document.

## 2.3 Page Overwritten Operators

The following operators have their implementations (inherited from the base Object) class overridden:

* len() – The \_\_len\_\_() method is overridden to return the number of tokenized words in the page.
* str() – The \_\_str\_\_() method is overridden to return the page classification (label).
* += The \_\_iadd\_\_() method is overridden to add text to the page (append).

## 2.4 Page Private Methods

The Page class contains the following private methods:

* There are no private methods.

## 2.5 Page Public Methods

The Page class contains the following public methods:

* store() – Store the NLP tokenized list to a file. The tokenized list is written in JSON format.
* load() – Load the NLP tokenized list and text from corresponding file into the words list.

# APPENDIX I: Updates

*Pre-Gap (Epipog) v1.1*

1. Added time property.
2. Added scanned property.
3. Added support for TIFF and JPG/PNG.

*Pre-Gap (Epipog) v1.3*

1. Add direct read of PDF resource element to determine if scanned page.
2. Fix not detecting scanned PDF if text extraction produced noise.

*Pre-Gap (Epipog) v1.4*

1. Added pageno property to Page class.
2. Added methods store() and load() to Page class to store/load NLP tokenized words to file.
3. Added method load() to Document class to reload NLP tokenized words from storage.
4. Added config keyword arguent to Document initializer to configure NLP preprocessing.

*Pre-Gap (Epipog) v1.5*

1. Added bagOfWords, freqDist and termFreq properties to Document and Page class..

# APPENDIX II: Anticipated Engineering

The following has been identified as enhancement/issues to be addressed in subsequent update:

1. What does it mean to add text to a document.
2. Break raw text into pages for > 50 lines
3. Refactor page counting for faster performance
4. Add page split endpoint for streaming interface and URL
5. Add more pdf test files
6. Fix bug of not handling Cryllic characters in page load() method.

**Proprietary Information  
Copyright ©2018, Epipog, All Rights Reserved**