# Project One

Wikipedia Indexer – Version 1.0

## Table of Contents

## 1. Project Description

This project aims to build a Wikipedia indexer with the following goals:

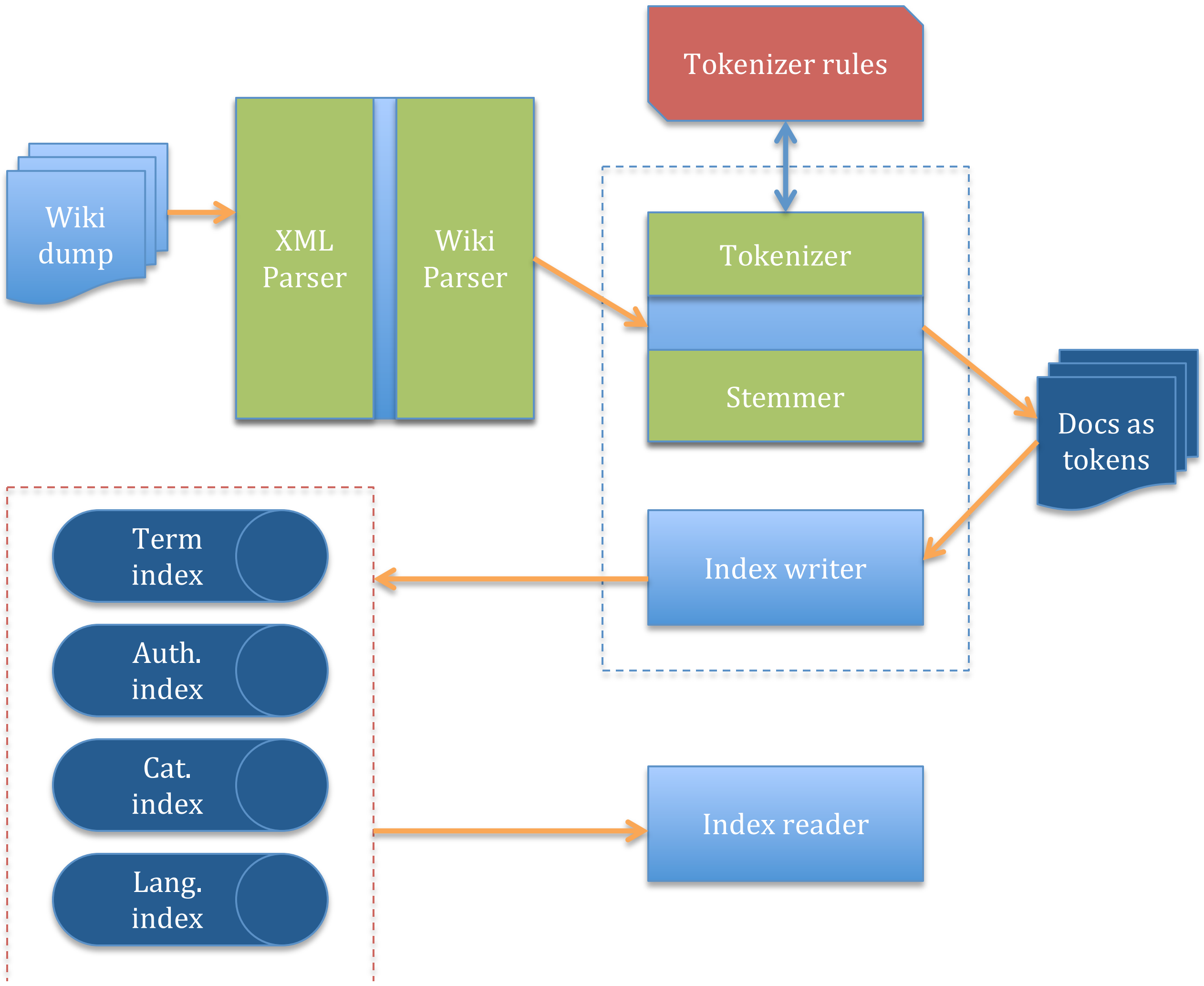
- Parse fairly involved Wikipedia markup.
- Index a decent sized subset of the Wikipedia corpus.
- Create multiple indexes on the page data as well as metadata.
- Provide an index introspection mechanism that can later be built upon to support queries.

You will be implementing this project purely in Java and starter code will be provided with detailed description of each component, method etc.

All deliverables are due by **29th September 2013, 23:59 EST/EDT**

## 2. System components

The system consists of two main components: a parser and an indexer. An overall system diagram is shown below.



The following subsections describe the functions of each component in more detail.

## 2.1. Parser

This component is responsible for converting the Wikipedia dump xml into a collection of documents. We have two document views as tabulated below:

| S. no. | Functional view | Technical counterpart | Description |
|---|---|---|---|
| **a.** | Document with pre-defined fields | WikipediaDocument | The page XML is parsed as-is and represented as a collection of pre-defined fields. The only transformation is removal of Wikipedia markup |
| **b.** | Document as a collection of terms | IndexableDocument | The underlying text is tokenized, normalized, etc. and represented as a collection of token streams and thereby, tokens. |

More technical details can be obtained by clicking on the class links listed above. For now, the distinction to understand is the demarcation between the two views. The first is an intermediate step that is closest to how the page appears in a browser. The second is where the structural details of the document are lost and the document is essentially nothing but a collection of tokens. As is evident, the first stage would focus on removal of Wikipedia markup and the second does the heavy lifting: tokenization, stemming and other linguistic normalizations. More details on this follow.

## 2.2. Indexer

Once the documents have been converted into a bag of tokens, the indexer then writes them onto multiple index files. Each index would also have its corresponding dictionary where the term to id mappings would be stored. You are expected to implement the following indexes. Representative snapshots of the indexes and dictionaries follow.

- **Term index**: An index that maps different terms to documents. This is the standard index on which you would perform retrieval.
- **Author index**: An author to document index, stores the different documents written by a given author.
- **Category index**: A category to document index, stores the different documents classified by a given category.
- **Language index**: This is a set of two indexes, one that maps the different documents in English to the corresponding language and the other a forward index that maps the different language specific pages for a given page.

- Link index: A forward index that maps the different Wikipedia pages referenced by a given page.

a. **Document dictionary**: This contains a document, i.e., Wikipedia page to document id mapping. This should contain mappings for all base English pages as well as referenced language pages. See the table below:

| **Document or page** | **Doc id** |
|---|---|
| Page_1 | 1 |
| Page_2 | 2 |
| ….. | |
| ar:Page_k | 5001 |
| …. | |
| zh:Page_m | 9001 |
| …… | |

This dictionary will be referenced by **all** the indexes.

b. **Term dictionary**: Like above it contains a term to term id mapping like illustrated below

| **Term** | **Term id** |
|---|---|
| … | … |
| apple | 12 |
| applicat | 13 |
| … | … |

c. **Term index**: It is a simple inverted index storing the document postings list for each term as follows

| | |
|---|---|
| 1 | 10,22,65,…. |
| 2 | 17,31,88,… |
| 3 | 12,16,82,112,…. |
| …… | |

d. **Author dictionary & author index**: Analogous to term dictionary and index; create a dictionary and index for authors.
e. **Category dictionary & category index**: Similar to term and author indexes, create a category dictionary and index.
f. **Language dictionary**: Create a language dictionary mapping each unique language found to a language id.
g. **Language indexes**: The inverted language index is a little tricky. Consider a page in English with a document id of say 100. It is referenced in say Arabic and German and the respective document ids for those pages are 7128 and 9432 respectively. There is another document in English with an id of 500 referenced in Arabic and Russian by ids 7832 and 8523 respectively. Then, the two indexes

would look as follows if the language ids for Arabic, German and Russian are 7, 12 and 15 respectively:

Inverted index

| **Language id** | **Postings list** |
|---|---|
| 7 | ...,100, 500,... |
| 12 | ...,100,... |
| 15 | ...,500,... |

Forward index

| **Document id** | **Postings list** |
|---|---|
| 100 | 7128,9432,... |
| 500 | 7832,8532,... |

h. **Link index**: This is a forward index that simply indexes all the linked pages by their document ids from a given page like:

| **Document id** | **Postings list** |
|---|---|
| ... | ... |
| 12 | 71,99,112,... |
| 13 | 16,51,88,... |
| ... | ...... |

We leave it to the students to decide whether they want to create one large merged index or multiple smaller indexes. However, for each index there should not be more than 27 buckets: one for each character and one more for symbols, special characters and numbers.

## 3. Technical details

This section provides details about the starter code, and corresponding methods that need to be implemented.

We provide an entry point class called the Runner to trigger the indexing process. The main method needs two arguments:

- The fully qualified properties filename. A reference properties file has been provided. You are expected to modify the entries as required to make your code run.
- Mode: One amongst i (index only), t (run tests only) and b (do both).

### 3.1. Parsing

This stage consists of mainly three classes:

### 3.1.1. Parser
This class is responsible for parsing the given dump file into a collection of WikipediaDocument instances. It should internally invoke methods from WIkipediaParser class to parse Wikipedia markup. This class should primarily parse XML.

Methods to be implemented:

| | | |
|---|---|---|
| void | parse | This method is the entry point into this class. This method should do all the parsing and populate the collection with the parsed documents. |
| 1. | String filename | This is the name of the XML file to be parsed. It is a fully qualified filename. |
| 2. | Collection <WikipediaDocument> docs | This is a reference to the collection to which the parsed documents must be added. See below. |

Utility methods:

| | | |
|---|---|---|
| void | add | This method is provided to add the parsed WikipediaDocument objects into the specified collection. The method does not do much but merely provides a synchronized way to do the adding. |
| 1. | WikipediaDocument doc | The parsed document that needs to be added to the collection. |
| 2. | Collection <WikipediaDocument> documents | The collection to which the document needs to be added |

### 3.1.2. WikipediaParser
This class is responsible for removing Wikipedia mark-up. It contains a bunch of static methods, each with a specific purpose. Refer to the Javadoc for more details on each method.

### 3.1.3. WIkipediaDocument
This class represents a structured view of the Wikipedia page. The different fields are listed below:

- publishDate: The ***timestamp*** tag within the revision tag from the XML file.
- author: The ***username*** or ***ip*** tag within the contributor tag from the XML
- id: The ***id*** tag within the top level page tag.
- Links: A set of all pages referenced by this page. We are only bothered about other Wikipedia pages here and external URLs, file links, etc. can be ignored.

- categories: A list of all categories attached to this page.
- langLinks: A map containing entries for language to the corresponding page.
- Sections: A given page is broken into various sections. For pages with no sections or orphaned content, a section title of “Default” should be used.

### 3.2. Indexing

TBD

### 3.3. Index reader

TBD

## 4. Code testing and evaluation

The project would be evaluated by running unit tests. A majority of the tests will be shipped with the starter code. We would run some additional tests as well as run the code against a larger corpus. Refer grading guidelines for more details.

### 4.1. Running local tests

For every method that you are expected to implement, corresponding tests will be provided. For a Class C in package P, its corresponding test class would be named CTest and can be found in the package P.test. Each method M would have its test method named as MTest.

We encourage you to write your own tests for code you add as well as any additional tests you think pertinent. We would be evaluating your code with our own test classes, so don’t be afraid to experiment.

### 4.2. Remote evaluation

Once your code has been submitted, we would only use your code files. We do not expect any external libraries to be required for this project. We will ignore any extraneous code submitted and you stand to lose points or worse, not get any if your code does not compile or run.

### 4.3. Grading guidelines

TBD

## 5. Submission guidelines

TBD