ISA
Institut für
SoftwareArchitektur

THM
TECHNISCHE HOCHSCHULE MITTELHESSEN

Burkhardt Renz

ebC eBooks Collection Toolbox

2. Oktober 2014

Technischer Bericht Nr. 5
des Instituts für SoftwareArchitektur

ebC eBooks Collection Toolbox
– Manual –

Burkhardt Renz
Technische Hochschule Mittelhessen
Institut für SoftwareArchitektur
Fachbereich MNI
Wiesenstr. 14
D-35390 Gießen
Burkhardt.Renz@mni.thm.de

Version 4.1 – October 2, 2014

This manual describes the idea behind the eBooks Collection Toolbox (eBC), how to use it, and it provides some details about the implementation in the programming language Clojure.

The program is available from my homepage homepages.thm.de/~hg11260/ebc.html.

1 Idea and purpose

Using the internet as an information source resulted in an ever increasing number of files on my local computer: lecture notes, papers, programm documentation, specifications and so forth.

I used to manage these files by maintaining a HTML directory. After downloading a new document, it had to be inserted in the HTML page – a quite cumbersome method. Since it is appropriate to give the downloaded documents a describing file name anyway, it seems to make more sense to generate a HTML directory of the collection from the file names of the documents.

This is the idea behind the eBooks Collection: Store and organize the documents in the file system by naming folders and documents according to a certain naming convention, and use this convention to generate a HTML directory for

the collection.

There is an obvious limitation to this approach: the file system is hierarchically organized and according to this organization the eBooks Collection is hierarchically organized as well. But often a document may be searched by several search criteria. To overcome this limitation the eBC Toolbox allows to generate a full-text index for searching. It uses the Apache Lucene library for indexing and searching the collection of documents.

2 Naming conventions

The eBooks Collection is based on some simple naming conventions.

Names. It takes account of files and folders whose names begin with `e?_` (`?` is a wildcard for some character).

Folders. eBC uses three kinds of folders:

- *Categories*, marked by the prefix `ec_`. Categories are the main fields of interest in the structure of an eBooks Collection. Categories may contain subcategories and topics.
- *Subcategories*, marked by the prefix `es_`. Subcategories are subfolders of categories and can contain topics.
- *Topics*, marked by the prefix `et_`. Topics are subfolders of categories or subcategories.

As an example the folder structure for the collection in the base directory `eBooks`

```
eBooks
+ ec_Foundations
  + et_Logic
  + es_Mathematics
    + et_Group Theory
```

leads to the following structure of the collection of documents:

- Categorie "Foundations" includes the topic "Logic" and the subcategory "Mathematics".
- Subcategory "Mathematics" of "Foundations" includes the topic "Group Theory".

Files. The folders of an eBooks Collection contain files whose prefix `eb_` or `ex_` identifies them as ebooks. Files with the prefix `ex_` are marked as particularly interesting. They will be highlighted in the HTML directory. (No two file names in a folder may differ just by their prefix.)

The file names for the ebooks follow these conventions:

```
<filename>  : e[b|x]_<authors>_<title>.<extension>
<authors>   : <name>, <given name>+<authors>
            | <name>, <given name>+
            | <name>, <given name>
            | <name>
<name>      : string (as allowed by the file system)
<given name>: string (ditto)
<title>     : string (ditto)
<extension> : extension (according to the file type)
```

A + following the names of the authors indicates more authors not mentioned in the file name.

Example for a file name according to the eBC convention:
`eb_Bieri, Robert+Renz, Burkhardt_Valuations on free resolutions.pdf`

3 Using the eBooks Collection Toolbox

3.1 Collecting ebooks

For structuring the collection of documents, I recommend:

- The first step is the creation of the folders for the categories (`ec_*`) which are the main fields of interest. It's wise to consider the categories carefully. so that the structure is quite stable. One may as well take into account that in the generated directory the categories are ordered alphabetically.
- Then one can assign documents to the categories, just by naming them according to the convention for ebooks and copy them into the category folders. Most of my documents are pdf, so when downloading they get opened in my pdf reader, and I save them in the appropriate folder of the collection under a new file name consisting of the author's name(s) and the title.
- If the folder of the category gets overcrowded, one creates folders for topics (`et_*`) or subcategories (`es_*`) and moves the appropriate files into these subfolders.
- If one has documents that are particularly interesting, one can highlight them by using the prefix `ex_` instead of `eb_`. This marking can also be a reminder of the relevance of the document if one considers to delete the file.

3.2 The functions in the toolbox

The eBooks Collection Toolbox has a command line interface.

`java -jar ebc.jar ebc.main -h` shows the command line interface:

```
This is eBC (eBooks Collection) Rev 4.1

Usage: ebc              starts the GUI for searching
       ebc [-b basedir] -c checks filenames in the eBooks collection
       ebc [-b basedir] -d creates HTML directory for the eBooks Collection
       ebc [-b basedir] -i indexes the eBooks Collection
       ebc [-b basedir] -u updates the index
       ebc -h help
       ebc -v version
```

Furthermore there is a graphical user interface for the programm, which can be started just by clicking on the jar file of the distribution. It shows the panel shown in figure 1.

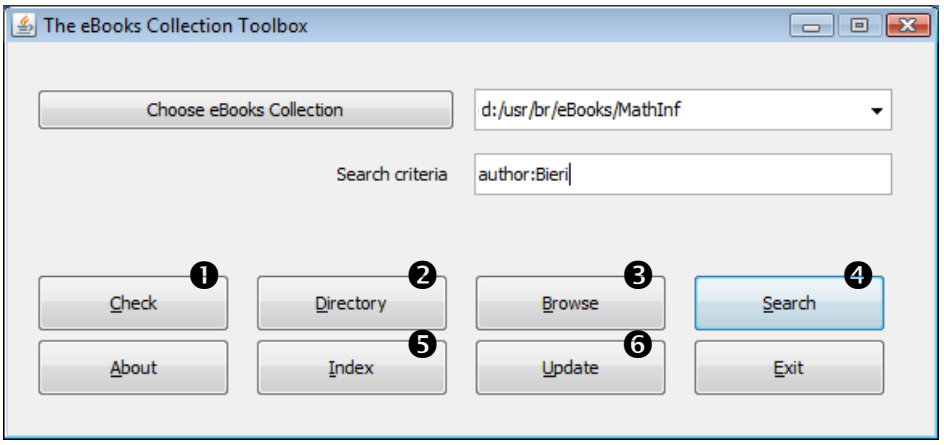

Figure 1: The eBooks Collection Toolbox – Functions

Location of collections. All functions in the eBooks Collection Toolbox work on the collection that is set in the combo box of the panel. The folders shown as preset in the combo box are read from the environment variable `ebcBasedirs`, which should contains the full paths to the collections. If you are using several different collections, the paths should be separated by semicolon `;`, e.g. `d:/usr/br/eBCOne;d:/usr/br/eBCTwo`.[1]

[1] By the way, one can use slash (/) as the path separator even on a Windows system.

Explanation of the functions:

❶ Checking the file names

This function traverses the folders of the eBooks Collection and checks whether file names beginning with a prefix `eb_` or of `ex_` are formed according to the naming conventions of eBC. It reports the results and shows the total number of ebooks in the collection.

When adding new documents to the collection the function can be used to assure, that they can be used properly by the other functions of the toolbox.

❷ Generating the HTML directory

This function generates a HTML directory for the collection of documents. The HTML pages are written in the base folder of the current collection. The home page is named `ebc-index.html`.

There is a file `ebc.css` in the base folder. This file contains the visual design of the HTML pages. It can be customized. If you want to do so, please take into account that the CSS file will be overwritten with each run of the generation function.

Generating the HTML directory needs some time. Generating the directory for a collection of about 1400 documents lasts about 90 seconds on my computer, so be patient.[2]

❸ Browsing the HTML directory

Having generated the HTML directory, this function starts your favorite browser with the index page (`ebc-index.html`) of the HTML directory of the current ebooks collection.

❹ Searching the full-text index

Given there is already a Lucene index for the collection (see the following functions) this function searches the full-text index for the search criteria given in the corresponding text field.

The syntax for the search is according to the Lucene query language, see e.g. `http://www.lucenetutorial.com/lucene-query-syntax.html`.

The index that is generated from each of the documents in the eBooks Collection contains the following fields:

- `path` – the path to the document, relative with respect to the base folder

[2] The test of eBC on a MAC Book Pro with an i5 processor and 256 GB flash memory was much faster, approximately by a factor of 2.

- `author` – the author or authors of the document, extracted from the documents name
- `title` – the title of the document, extracted from the documents name
- `content` – the text content of the document. This field is the default field for searching with Lucene.
- `ext` – the extension of the document's name, indicating the type of document
- `date` – the modification date of the file

❺ Generating the full-text index

The eBooks Collection Toolbox offers the functionality to generate a full-text index. It uses the Apache Lucene indexing and searching library for that. This function starts the generation of a brand new index from all the documents in the collection. If there is already an index, it will be replaced by the new one.

Depending on the type and the size of the documents, it is quite a task to extract the contents, analyze them and store them in the Lucene index. For a collection of about 1400 documents, mainly pdf, the generation of the index lasts on my pc (not a very fast one) about 30 minutes. The resulting Lucene index has a size of about 100 MB.

Fortunately it is not necessary to generate the index from scratch very often. After extensions or minor changes of the collection one can use the following function:

❻ Updating the full-text index

This function synchronizes the content of the Lucene index with changes in the collection of the documents.

4 Design and implementation

The eBooks Collection Toolbox is written in Clojure (http://clojure.org/). Clojure structures the code into namespaces. The organization of the namespaces used in eBC is roughly build according to the functionality of the application.

The public repository of the code is https://github.com/esb-dev/eBC.

4.1 Generating the HTML directory

Figure 2 shows the namespaces that are used to check the file names in the collection and to generate the HTML directory.

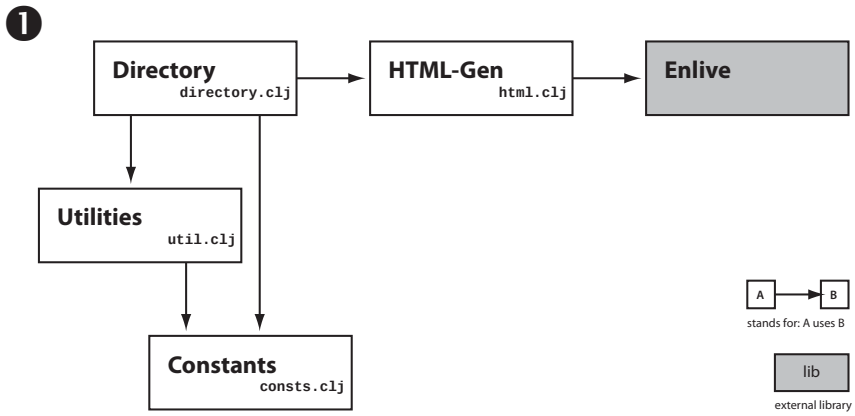

Figure 2: Components for eBC's directory

The namespace `consts.clj` contains most of the **constants** used in the application. Other namespaces that use some of the constants include the vars from `ebc.consts` with an alias of `c`, e.g.

```
(ns ebc.util
  (:require [ebc.consts :as c]
            ...))
```

This rule has the effect that the usage of constants throughout the code is easily recognizable, since they all begin with `c/`, e.g. `c/ebc-rev`.

Utilities are coded in the namespace `util.clj`. We have utility functions for path and file handling, localization for sorting, date as well as functions that are specific to the naming conventions of documents in eBC and the structuring of information about such documents in a Clojure map.

`consts.clj` and `util.clj` are used by nearly all of the other modules.

`directory.clj` contains the function that checks if the file names in a collection are constructed according to the naming conventions. In addition, the namespace has functions that collect the data needed for the generation of the HTML directory.

For the listing of books ordered by authors, by titles or by dates we use three vectors defining the groups for these listings. There are just 4 groups of documents for each type, yielding to 4 HTML pages respectively. In the current implementation this grouping is hard-wired in `directory.clj`.

`html.clj` uses the data collected by `directory.clj` to generate the HTML pages of the eBooks Collection.

The implementation relies on **Enlive** written by Christophe Grand (https://github.com/cgrand/enlive). Enlive has an sophisticated approach for the

generation of HTML:

One defines templates of the intended pages with some example data. For the visual design of the page, these templates can be used to define a CSS file that leads to an appealing appearance of the pages.

The data can be filled in using the template pages by locating the position in the tree of the HTML elements in the template via a selection mechanism similar to the selectors in cascading style sheets.

4.2 Generating, updating, and searching the full-text-index

Figure 3 depicts the dependencies of the namespaces for creating, searching and updating the full-text index for the eBooks Collection.

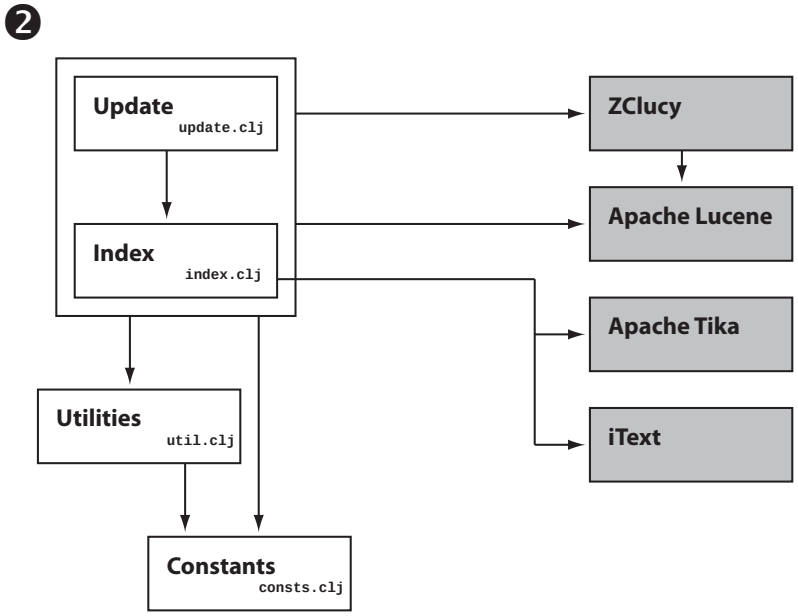

Figure 3: Components for the full-text index

eBC incorporates the **Apache Lucene** indexing and searching library (http://lucene.apache.org/). There is a Clojure wrapper for the essential functionality of Apache Lucene, **ZClucy**, written and maintained by David Raphael and others (https://github.com/ceterumnet/clucy). In eBC the main functionality of Lucene is provided by functions from ZClucy, just sometimes we need to use Lucene directly.

To index documents with Apache Lucene, one has to provide the text content of the document, i.e., one needs to extract text from the documents. **Apache Tika** (http://tika.apache.org/) is a library to detect and extract metadata

and content from all kinds of documents. eBC uses Tika to extract text from plain text files[3], text from HTML files and EPUB files.

Tika can extract content from PDF files too, but in eBC we use **iText** (http://itextpdf.com/) for that task[4].

The namespace `index.clj` contains the functions for generating the full-text index. In Clojure the files are provided by a lazy sequence of file objects. The processing of the files and the preparation for the indexing by Lucene can be done through map functions working on the items of that sequence. We use an Clojure agent to report the progress of the processing of the files in an asynchronical way.

The namespace `update.clj` is populated by the data and functions needed to synchronize the index with the current content of the collection of documents. In this task we benefit from the functions for sets in Clojure: The set of new documents e.g. is the set difference of the documents in the collection and the documents in the index. The constructs of Clojure enable concise code for such computations.

4.3 The user interfaces of eBC

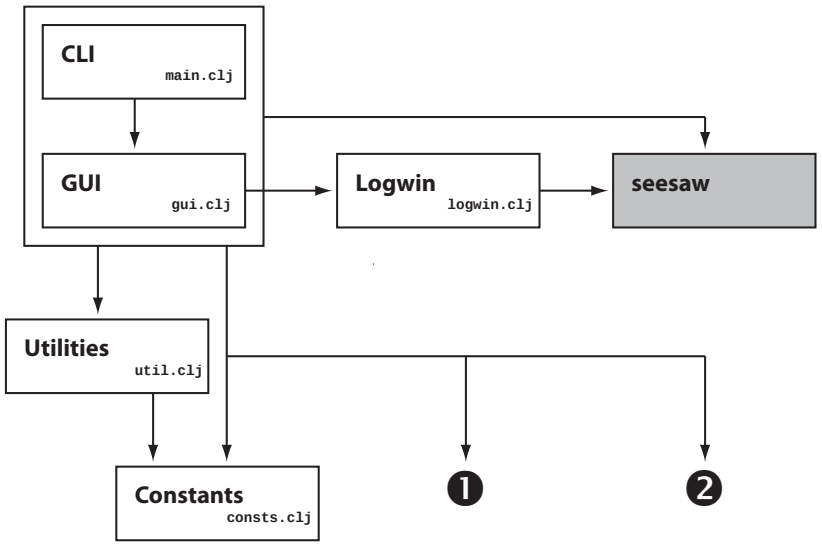

Figure 4: Components for the user interface

[3] Not an unnecessary undertaking, since Tika takes the encoding of the text files into account.

[4] Testing with the kind of PDF documents I usually have in my collection, I had the experience that iText was a bit faster than Apache PDFBox, the library that is used by Apache Tika.

In figure 4 the modules for the user interface are shown.

A command line interface **CLI** provides all the functions of eBC and starts the graphical user interface **GUI** as well.

The graphical user interface uses the Clojure library **seesaw** (https://github.com/daveray/seesaw) written by Dave Ray. On his seesaw page on github the first sentence is: "Seesaw turns the Horror of Swing into a friendly, well-documented, Clojure library" – that's true. The possibility to address GUI elements via selectors is one of the many properties of seesaw that makes coding with that library particularly convenient.

The **Logwin** is a sophisticated widget in the seesaw library which made the reporting of progress information in the graphical user interface of eBC a trouble-free undertaking. `logwin.clj` is just a thin wrapper aroung seesaw's `log_window`.

5 Remarks and hints

Exceptional documents. Sometimes one wants to include exceptional documents in the eBooks Collection that can not be used as a single file. An example is a collection of HTML pages that have links among themselves such that the file names can not be changed according to eBC's naming convention. In such cases there is a simple workaround: one creates a HTML page that has a link to the index HTML page of the files and puts this referring page into the eBooks collection.

Environment variables in Mac OSX. Mac OSX uses `launchd` to start daemons, applications and so forth, see http://launchd.info/. This mechanism can be used to set environment variables. On the mentioned web page under "Configuration" is noted: "You can set environment variables per launchd instance by issuing the launchctl command setenv. Logging out (in case of a user launchd) or rebooting (in case of the root launchd) will revert those changes. To make these changes persistent you'll have to placing this command in /etc/launchd-user.conf (in case of a user launchd) or /etc/launchd.conf (for all launchd processes)."

Example:

```
launchctl setenv ebcBasedirs /Users/br/Documents/eBooks
```

will set the environment variable ebcBasedirs for the next start of eBC. To persist the setting after reboot, one edits the file `/etc/launchd-br.conf` for user `br` or `/etc/launchd.conf` for all users by adding the line

```
setenv ebcBasedirs /Users/br/Documents/eBooks
```

Ideas for enhancing eBC

- Integration of an option whether the CSS file for the HTML directory should be written with the next generation of the directory.

- Making configurable the grouping of the listings of the books ordered by authors, title or dates.
- Streamlining the content extraction from various documents by using the content detection facility of Apache Tika.
- Improving the speed of eBC.