The HLT Web Manager

(version 1.4.2)

User Manual

Christian Girardi (cgirardi@fbk.eu)
FBK-irst, Povo, Trento, Italy
January 2011

Introduction

This document is the usage manual of the *HLT Web Manager*, a toolkit for crawling web pages. It consists of three modules: *webdown*, *webparser* and *webaligner*. Their main functionalities are:

- crawling HTML pages from the Web
- specification of URLs to be crawled via regular expressions
- possibility of recursive crawling by following links
- storing of crawled web pages in a compressed archive
- extracting content information (text, images, videos, etc.) of stored web pages and their display in XML format
- alignment of web pages including the same content but written in different languages

Many other functions are featured by the three modules. Details are provided below.

Getting started

The scripts described in this manual are included in the distribution file package WebManager-1.4.2.zip. In order to use the tool, copy the package under any directory of your PC and unzip it.

Environment Settings

The environment variable `WEBDOWN`, by default set to ".", must be set to the path of the HLT Web Manager package:

- under Linux or Mac OS: set it in the scripts *webdown* and *webparser*
- under Windows OS: set it in the *webdown.bat* and *webparser.bat* files

Requirements

Java 1.5.0 or later is required. Note that the following libraries are needed (all jars are already included into the distribution):

- jericho-html-3.2.jar (HTML parser: http://jericho.htmlparser.net/)
- lc4j-0.3.jar (Lc4j, a language categorization Java library: http://olivo.net/software/lc4j/)
- colt.jar (mathematical library: http://acs.lbl.gov/software/colt/)
- fastutil-5.1.5.jar (extension of the Java™ Collections Framework: http://fastutil.dsi.unimi.it/)
- jdom.jar (code for manipulating XML data: http://www.jdom.org/)
- json_simple-1.1.jar (a simple Java toolkit for JSON: http://code.google.com/p/json-simple/)

Examples

The data sets used in the following examples can be found in the `test/` directory of the package.
Given a starting URL, for example `http://www.fbk.eu/`, its web page can be downloaded running the `webdown` script:

```
$> webdown -O test/ "http://www.fbk.eu/"
```

If the download is successful, the parameters used, the number of followed links (by default the starting url only), the number of downloaded pages and the number of pages stored in the output archive, a file with .cgt extension, will be displayed in the shell, as follows:

```
# Parameters used:   -O test/ http://www.fbk.eu/
# Followed links:    1
# Downloaded pages:  1
# Stored documents:  1 (/Users/cgirardi/test/www.fbk.eu.cgt)
```

The output archive, containing the downloaded data, is created and stored in the directory `test/`.

The `webparser` command allows the information included in the HLT Web Manager distribution. It reads an input .cgt file and creates an XML file with some default header information (such as encoding, date of downloading, url, ...) and the relevant text for each stored web page. The command to be run is the following:

```
$> webparser test/www.fbk.eu.cgt
```

The output file `test/www.fbk.eu.xml` is created.

More on webdown

The script `webdown` generates a web page archive by crawling an url, a file or the files of a directory (in order to store all subfolders, the option -r must be used). The usage is the following:

```
$> webdown [OPTIONS] <URL | FILE | DIR>
```

A number of options are available so that the user can customize the crawling:

```
-h            show this help
-v            verbose mode
-H            go to the URLs that have a different domain (foreing hosts) than the starting URL.
              This checking is applied when recursive mode is activated
-np           do not ascend to the parent directory
-img          get all images
-all          get all resource types except images (e.g. text, html, pdf, mov, ...)
-r            specify recursive mode
-l DEPTH      in recursive mode, maximum recursion DEPTH (0 for infinite, the default)
-Q NUMBER     set retrieval quota to NUMBER (0 for infinite, the default)
-O DIR        write the archive file in the directory DIR
-a NAME       append downloaded pages to file NAME.cgt (the URL host is the default)
-n            create a new archive. By default, the downloaded pages are appended to the archive
              (if already exists)
-i FILE       read URLs from FILE (a URL per line)
-e ENCODING   by default, webdown automatically detects the encoding of the crawled
              page. Alternatively, a user can force to read the page with encoding
              ENCODING (i.e. -e "UTF8")
-s REGEXP     store links that match the regular expression REGEXP (i.e. -s '.*\.html$').
              A list is accepted using "|" as delimiter (i.e. -s '.*.html$|.*news*.')
-ns REGEXP    do not store links that match the regular expression REGEXP (i.e. -ns '.*\.asp\?.*').
              A list is accepted using "|" as delimiter (i.e. -ns '.*.asp\?.*|.*index.html?')
-I REGEXP     follow links that match the regular expression REGEXP (i.e. -I '.*index.*').
              A list is accepted using "|" as delimiter (i.e. -I '.*index.*|.*news.*').
              Option -np is not considered
-X REGEXP     do not follow links that match the regular expression REGEXP (i.e. -X '.*index.*').
              A list is accepted using "|" as delimiter (i.e. -X '.*index.*|.*news.*').
              Option -np is not considered
-pt           append the download time at the end of the URL (useful in case of multiple
              downloading of the same URL, i.e. dynamic pages)
-w WAIT       wait from 0...WAIT secs between retrievals. This option should be used to avoid
              overloading the web server
-t            set the number of retries to NUMBER (the default is 1). This option can be used when the
              web server is busy in order to try the downloading NUMBER times
-proxyhost HOST     set HOST as proxy host
-proxyport PORT     set PORT as proxy port.
```

Usage examples:

1. If you need to crawl an entire web site following all the links (see -r) and download all textual pages, run the following command:

   ```
   $> webdown -O test -v -w 5 -r http://www.fbk.eu/
   ```

   The option -w 5 tells the crawler to wait 5 seconds before the download of each page. Note that by default `webdown` does not crawl hosts which are external with respect to the specified URL; in the above example, it would follow only urls starting with www.fbk.eu. This behavior can be disabled with the option -H (be careful). By setting the -l option, the user can define the recursion depth, i.e. the maximum number of levels that the retrieval must descend.

2. `webdown` is particularly powerful when the purpose is to crawl only specific pages of a web site. For instance, if a user is interested in all pages with the prefix `http://www.fbk.eu/info`, the options -I, -X, -s and -ns allows a very refined customization; the command:

   ```
   $> webdown -O test -v -r -s "http://www.fbk.eu/info.*htm" -I "http://www.fbk.eu/info.*" http://www.fbk.eu/
   ```

   stores all pages matching the pattern "`http://www.fbk.eu/info/.*htm`" by following all links complying with the pattern "`http://www.fbk.eu/info/.*`". The exclusion of some pages can be performed as well:

   ```
   $> webdown -O test -v -r -ns ".*asp" -X "http://www.fbk.eu/info/.*" http://www.fbk.eu/
   ```

With this command, the user can recursively download pages from `http://www.fbk.eu/`. All pages are followed, apart those with prefix http://www.fbk.eu/info/, according to option -X; besides, only pages not ending with ".*asp" will be stored into the archive.

3. Given a list of URLs written in a text file (e.g. list_of_urls.url, one URL per line), we can download all the pages with the command:

   ```
   $> webdown -O test -a archive_of_urls.cgt -i test/list_of_urls.url
   ```

   The output will be appended to the archive `archive_of_urls.cgt` in the `test/` directory.

4. For appending only the HTML files of the directory file_directory/ with a specific enconding (e.g. "ISO-8859-1") to the test/filesystem_archive.cgt:

   ```
   $> webdown -O test -a filesystem_archive.cgt -e "ISO-8859-1" file_directory/
   ```

5. For downloading images from the Flickr web site, which uses JSON data representation:

   ```
   $> webdown -a image_archive.cgt -Q 10 -img "http://api.flickr.com/services/feeds/photos_public.gne?format=json"
   ```

   The option -Q 10 defines the maximum number of images to be downloaded.

Note: When `webdown` accesses a server to crawl web pages, it leaves the *user agent string* specified in the `conf/webdownload.properties` file. If the string is missing, no sign is left, but in this case some servers deny the crawling.

More on webparser

Given an archive, `webparser` allows the handle of its contents. It mainly performs two tasks: provides information about the archive and manipulates the downloaded resources. The relevant content of each page is extracted by CleanPro library[1].
The command to parse an archive is:

```
$> webparser [OPTION] <FILE.cgt>
```

In the default setting, webparser converts the pages of the archive `FILE.cgt` into a structured XML document, with some parts of the HTML page turned into XML element (for instance the metadata of the header and the body).

The options are:

```
-h          show this help
-v          verbose
-l          show only the list of URLs in the archive and print it on STDOUT
-la         show only the list of URLs with complete info and print it on STDOUT
-p URL      print the content of the URL to STDOUT
-ld LANG    language detection mode to extract only the content written in the given
            language
-m          mirror all stored pages on filesystem (force creation of directories)
-mu URL     mirror only the URL page on filesystem (force creation of directories)
-mf FILE    mirror stored pages from the URLs in the FILE (one URL per line) on filesystem
            (force creation of directories)
-a NAME     set the output filename to NAME (the default is the input filename, with file extension .xml)
-O DIR      write output file in the directory DIR
-X REGEXP   do not parse URLs that match the regular expression REGEXP
-c NUM      extract only pages including at least NUM chars of relevant content
-text       html2txt function is applied to store HTML files, instead of CleanPro
-cl         use a specific Java class to parse the stored pages. This class must implement
            the methods of the inferface class fbk.hlt.web.WebPageInterface.
            The method signatures are:
            -> public Map getHeader (String url, String html)
            -> public String getBody (String url, String html)
```

The command:

```
$> webparser test/www.fbk.eu.cgt
```

creates the XML file `test/www.fbk.eu.xml` by parsing the archive `test/www.fbk.eu.cgt`.

By default, the class `fbk.hlt.web.Html2RelevantText` included in the jar file `cleanpro-1.1.2.jar` is used. This class is able to extract the relevant text from a generic web page the relevant text written in any language. As an alternative, users can create their own class to parse web pages and pass it to `webparser` via the option `-cl`. The customized class must include two methods of the interface class `fbk.hlt.web.WebPageInterface` implemented as follows:

```
import fbk.hlt.web.WebPageInterface;

public class MyParseClass implements WebPageInterface {

    public Map getHeader(String url, String html) {
        Map tab = new HashMap();
        ...
        return tab;
    }

    public String getBody(String url, String html) {
         String body = "";
            ...
         return body;
    }
}
```

Some usage examples for customized data extraction:

1. To extract from the archive only the pages with at least 100 characters of Italian, run the command:

   ```
   $> webparser -ld italian -c 100 test/www.fbk.eu.cgt
   ```

   It is possible to extract data in a specific language (see the available languages in Appendix 1).
2. To extract some specific metadata and tables from a given webpage, for example `http://hlt.fbk.eu/en/people/cgirardi`, first the page must be downloaded:

```
$> webdown -n -O test -a test.cgt "http://hlt.fbk.eu/en/people/cgirardi"
```

The command locally saves the webpage into the archive `test/test.cgt`.
Now a Java class for data extraction must be created. The class must implement both methods `getHeader` and `getBody`; Appendix 2 shows a possible implementation. In particular, `getHeader` extracts the content of meta tags and title tags. The selection of these tags can be carried out through the `findAllElements` method on Source objects (see Jericho manual for more details). In our example, we focus on meta tags with attribute "name". In a similar way, `getBody` finds the table that meets some constraints, like a certain value of the width attribute or a specific word in the content. The class is compiled with the following command:

```
$> javac -cp "classes/:lib/jericho-html-3.2.jar" -d classes/ test/TestParser.java
```

The file .class is created in the `classes/` folder and can be used to parse the pages included in the archive `test/test.cgt`:

```
$> webparser -cl TestParser test/test.cgt
```

The output file `test/test.xml` contains the data required: the element "head" contains all data extracted by `getHeader` method, while the information extracted by `getBody` is saved as `content`.

In general, if the CleanPro parse class does not meet your needs, a specific parse class should be implemented for each archive.

More on webaligner

Given an archive, the script `webaligner` aligns its web pages that have the same textual contents written in different languages.
The implementation relies on the following assumptions: if two - and only two - pages of the archive written in different languages contain the same images, or they are linked to each other, likely they are alignable.
The command to align two or more XML files is:

```
$> webaligner [OPTION] [<FILE.xml>]+
```

The options are:

```
-v         verbose
-e DIR     extract output file in the directory DIR
```

To extract aligned content at page level from a web site, first an archive must be created with the `webdown` script; for example, the archive from `http://www.fbk.eu/` is built by running:

```
$> webdown -O /tmp -a fbk.cgt -l 2 -r -v "http://www.fbk.eu/"
```

This way, all pages reachable from `http://www.fbk.eu` with at most 2 recursive steps are stored in the `/tmp/fbk.cgt` archive. Note that during the download, `webdown` does not append to the archive pages already stored.
Then, `webparser` is used to separate, for example, English and Italian pages through the `-ld` option:

```
$> webparser -O /tmp/ -a fbk_english.xml -ld english -v /tmp/fbk.cgt
$> webparser -O /tmp/ -a fbk_italian.xml -ld italian -v /tmp/fbk.cgt
```

Finally, `webaligner` is called to create a new XML file for each language containing only aligned pages:

```
$> webaligner -e /tmp/aligned_fbk_IT-EN/ /tmp/fbk_italian.xml /tmp/fbk_english.xml
```

References

[1] Christian Girardi, *Htmcleaner: Extracting Relevant Text from Web*, 3rd Web as Corpus workshop (WAC3), Presses Universitaires de Louvain, 2007, pp. 141- 143 (3rd Web as Corpus workshop (WAC3), Lovain la Neuve, Belgium, 15/09/2007 - 16/09/2007).

If you use this tool, please cite:
Christian Girardi, *The HLT Web Manager*. FBK-Technical Report no. 23969, 2011.

FAQ *(in progress)*

- Is it possible to follow links contained in page frames?

Yes. By default `webdown` handles frames, anchors and JSON links.

- Which encoding is used by the HLT Web Manager toolkit?

During the download, `webdown` detects the encoding of the page and stores it in the original format. However, the tool internally maps the given encoding to UTF8. Therefore, in the output XML all content is saved in UTF8.

- Does `webdown` respect `robots.txt` files?

No. Currently webdown ignores the information found in `robots.txt`. However, for the sake of fairness, FBK leaves own user agent sign and download pages according to the policy described in `http://hlt.fbk.eu/hltbot.html`.

Many thanks to Sara Tonelli and Mauro Cettolo for having tested the tool and their helpful comments.

Appendix 1

List of available languages:
afrikaans, albanian, alemannich, amharic, arabic, armenian, asturian, basque, belarus, belarusian, bosnian, breton, bulgarian, catalan, chinese, chuvash, croatian, czech, danish, dutch, english, esperanto, estonian, faroese, finnish, french, frisian, galician, georgian, german, greek, hawaian, hebrew, hindi, hungarian, icelandic, ido, indonesian, interlingua, irish, italian, japanese, javanese, korean, kurdish, latin, latvian, limburgish, lithuanian, lituanian, low_saxon, luxembourgish, macedonian, malay, manx, marathi, mf, middle_frisian, min_nan, mingo, nepali, norwegian, norwegian_bokmal, norwegian_nynorsk, occitan, ossetic, persian, polish, portuguese, quechua, romanian, rumantsch, russian, sanskrit, scots, scots_gaelic, serbian, serbo_croatian, slovak, slovenian, spanish, swahili, swedish, tagalog, tahi, tamil, tatar, thai, turkish, ukrainian, ukranian, vietnamese, walloon, welsh, western_frisian, yiddish

Appendix 2

Example of customized page parsing:

```
import net.htmlparser.jericho.Element;
import net.htmlparser.jericho.HTMLElementName;
import net.htmlparser.jericho.Source;

import java.util.Map;
import java.util.HashMap;
import java.util.List;

public class TestParser {
    //this method extracts all metadata of the page
    public Map getHeader(String url, String html) {
        HashMap mapValues = new HashMap();
                mapValues.put("url", url);
        Source source = new Source(html);
        Element el;
        String attr;
        //get META tags (like keyword, description, ...)
        List els =source.getAllElements(HTMLElementName.META);
        if (els != null)
            for (int l=0; l< els.size(); l++) {
                el = (Element) els.get(l);
                attr = el.getAttributeValue("name");
                if (attr != null) {
                    mapValues.put("meta_" + attr.toLowerCase(), el.getAttributeValue("content")); // Attribute values are automatically decode
                }
            }

        //get TITLE tag
        els =source.getAllElements(HTMLElementName.TITLE);
        if (els != null && els.size() > 0)  {
            el = (Element) els.get(0);
            mapValues.put("head_title", el.getTextExtractor().toString());
        }
        return mapValues;
    }

    //this method extracts the content of the page. In this example all div elements with the attribute class="content" are considered
    public String getBody(String url, String html) {
        Source source = new Source(html);
        StringBuffer content = new StringBuffer();
        List els =source.getAllElements(HTMLElementName.DIV);
        Element el;
        for (int l=0; l< els.size(); l++) {
            el = (Element) els.get(l);
            String attr = el.getAttributeValue("class");
            if (attr != null && attr.equalsIgnoreCase("content")) {
                content.append(el.getTextExtractor().toString()).append("\n");
            }
        }

        return content.toString();
    }
}
```