The NEWS Parser

(version 1.0)

User Manual

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

Introduction

This manual illustrates the functionalities of the News Parser toolkit.

It should put you quickly in the condition of:

- moving from zip archive in a custom optimized archive; News Parser stores the document in a compressed archive with some meta information;
- extracting the information from the archive and displaying them in a single xml format;
- it allows a user to apply a user-defined function (by Java class) on the all archive content generating a new annotation, again in the custom archive format;

Getting started

The scripts described in this manual are included in the distribution file package NewsParser-1.0.zip. In order to use the tool, you just have to copy the package under any directory of your PC and unzip it.

Requirements

Java 1.7.0 or later is required. Note that the following libraries are needed (all jars are already included into the distribution):

- jdom.jar (code for manipulating XML data: http://www.jdom.org/)

Examples

The data sets used in the following examples can be found in the `test/` directory.
Given a starting zip archive, for example test/input/lexisnexis-example.zip, its content can be managed running the `zip2cgt.sh` script:

```
$> zip2cgt.sh test/input/lexisnexis-example.zip test/output/lexisnexis-example.cgt
```

If the elaboration is successful, the amount of the saved document will be displayed in the shell, as follows:

```
The archive test/output/lexisnexis-example.cgt contains 53 documents.
```

The output archive with .cgt extension, containing the parsed data, has been created in the directory test/output/. The class ZipManager.java is able to parser each XML file of the zip input file (provided by LexisNexis) and to extract the relevant text and the metadata you are interested in (for more details about the parsing procedure see the Java source file src/main/java/org/fbk/hlt/newsreader/ZipManager.java).
By default, for each LexisNexis XML, the value of attribute /nitf/head/docdata/date.issue/norm is mapped as the content of the element /xml/file/head/date/, the content of the elements /nitf/body/body.head/headline/hl1/ is put in the element /xml/file/head/title/, while the concatenation of the content of the elements /nitf/body/body.head/headline/hl1/ and /nitf/body/body.content/block/p/ is the content of the element /xml/file/content/ in the .cgt archive.

More on cgtparser.sh

Given an .cgt archive, you can handle its content with the command `cgtparser.sh`. It mainly performs two tasks: it provides information about the archive and manipulates the documents. The script to parse an archive is:

```
$> cgtparser.sh [OPTION] <FILE.cgt>
```

A number of options are available:

```
 -h           show this help
 -v           verbose
 -l           show the list of URLs only
 -la          show list of URLs with complete info
 -p URL       print the content of the URL
 -m           mirror all stored pages on filesystem (it forces the creation of the directories)
 -mu URL      mirror the URL page on filesystem (it forces the creation of the directories)
 -mf FILE     mirror stored pages from URLs of the FILE (a URL per line) on filesystem (it forces the creation of the directories)
 -O DIR       write output file in the directory DIR
 -a NAME      set the output filename to NAME (the defult is the input filename, with file extension .xml, is the default)
 -e ENCODING  force to read the page with encoding ENCODING (i.e. -e "ISO-8859-1")
 -I REGEXP    the parsing is applied only to the urls that match the regular expression REGEXP.
              A list is accepted using "|" as delimiter (i.e. -s '.*.html$|.*news*.');
 -X REGEXP    do not parse urls that match the regular expression REGEXP.
              A list is accepted using "|" as delimiter (i.e. -s '.*.html$|.*news*.');
 -cl CLASS    use a specify Java CLASS to parse the stored page. This class must have the methods of the
              inferface class fbk.hlt.parsing.CGTDocumentInterface. The method signatures are:
                - public LinkedHashMap getHeader (String url, String text, LinkedHashMap header)
                - public String getBody (String url, String text, LinkedHashMap header)
```

The script cgtparser.sh converts the documents of the archive FILE.cgt into a structured XML document with two main parts: the metadata (called the header) and the main relevant content (called the body).
In order to obtain an XML file, you can run the command:

```
$> cgtparser.sh -v test/output/lexisnexis-example.cgt
```

that will create the XML test/output/lexisnexis-example.xml, given the archive test/output/lexisnexis-example.cgt.

By default, the class src/fbk/hlt/parsing/BaseParser.java is used on each stored document. Otherwise, the user can create his own class and with the option -cl, the user can specify the class which is instatiated in the parsing document phase.
In order to create a customized class, two methods of the interface class fbk.hlt.parsing.CGTDocumentInterface must be implemented as follows:

```
import fbk.hlt.parsing.CGTDocumentInterface;

public class MyParseClass implements CGTDocumentInterface {

    LinkedHashMap getHeader (String url, String text, LinkedHashMap header) {
        LinkedHashMap<String, String> myheader = new LinkedHashMap<String, String>();
        ...
        return myheader;
    }

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

Some usage examples for customized data extraction:

1. If you want to list of the urls (or better the path of the XML file in the source .zip file) from the archive, you can run the command:

   ```
   $> cgtparser.sh -l test/output/lexisnexis-example.cgt
   ```

2. If you want to extract just a document identified by the path "2003/10/1/49NM-91M0-009F-S1DG.xml":

   ```
   $> cgtparser.sh -mu "2003/10/1/49NM-91M0-009F-S1DG.xml" test/output/lexisnexis-example.cgt
   ```

   The document is locally saved in the 2003/ folder.

3. If you want to extract all the documents on the file system, you can run:

   ```
   $> cgtparser.sh -m test/output/lexisnexis-example.cgt
   ```

4. Let's assume that we want to apply your annotation tool on the all document of the archive test/output/lexisnexis-example.cgt.
   We first have to create a Java class for the annotation. This class implements both methods getHeader and getBody (see the code reported in Appendix 2). In particular, getHeader will handle the LinkedHashMap with existing metadata (they are coded as key/value pairs) and it returns a new richer LinkedHashMap. In a similar way, getBody finds the annotation you want that starts with the initial test and metadata of the Map header. The Appendix 1 reports an example of a simple annotation. The class has to be compiled with the following command:

   ```
   $> javac -cp "classes/" TestParser.java
   ```

   and finally you can call:

   ```
   $> ./cgtparser.sh -v -O ./ -cl TestParser test/output/lexisnexis-example.cgt
   ```

   If you want to annotate only some documents you can filter them using the -I option as following:

   ```
   $> ./cgtparser.sh -v -O ./ -cl TestParser -I ".*49NM-91J0.*|.*49NM-91R0.*" test/output/lexisnexis-example.cgt
   ```

   Viceversa or alternatively you can use the option -X for excluding some url using the regular expressions.

Appendix 1

Example of customized page parsing:

```
import java.util.*;
import fbk.hlt.parsing.CGTDocumentInterface;

public class TestParser implements CGTDocumentInterface {

    public LinkedHashMap getHeader(String url, String text, LinkedHashMap header) {
        return null;
    }

    public String getBody(String url, String text, LinkedHashMap header) {
        String annotation = "";
        if (header != null)
            annotation += "HEADER METADATA: " + header.size() + "\n";
        else
            annotation += "NO HEADER\n";

        if (text != null)
            annotation += "TEXT LENGTH: " + String.valueOf(text.length()) + " chars\n";
        else
            annotation += "NO TEXT\n";

        return annotation;
    }
}
```

Appendix 2

Example of TextPro's annotation during the page parsing. This is useful for anther external system call (specifying some parameters using an external properties file)

```
import java.io.*;
import java.util.*;
import fbk.hlt.parsing.CGTDocumentInterface;

/**
 * Author: Christian Girardi (cgirardi@fbk.eu)
 * Date: 04-apr-2013
 */

public class TextproParser implements CGTDocumentInterface {
    static String TEXTPRO_PATH = null;

    public TextproParser() {
        try {
            //load the TextPro home dir path from the file ./conf/textpro.properties
            Properties properties = new Properties();
            properties.load(new FileInputStream("./conf/textpro.properties"));
            TEXTPRO_PATH = properties.getProperty("textpropath");
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }


    public LinkedHashMap getHeader(String url, String text, LinkedHashMap header) {
        return null;
    }

    public String getBody(String url, String text, LinkedHashMap header) {
        if (text != null && text.length() > 0) {
            File tmpFile = new File("/tmp/aa");
            OutputStreamWriter out = null;
            try {
                // write the input
                out = new OutputStreamWriter(new FileOutputStream(tmpFile), "UTF8");
                out.write(text);
                out.close();

                // run TextPro
                String[] CONFIG = {"TEXTPRO=" + TEXTPRO_PATH, "PATH=" + "/usr/bin/" + ":."};
                String[] cmd = {"/bin/tcsh", "-c", "perl " + TEXTPRO_PATH + "/textpro.pl -l eng -y "+tmpFile};
                Process process = run(cmd, CONFIG);
                process.waitFor();

                //read the TextPro's output
                BufferedReader txpFile = new BufferedReader (new InputStreamReader (new FileInputStream (tmpFile.getCanonicalPath() + ".txp"))
                StringBuilder result = new StringBuilder();
                String line;
                while ((line = txpFile.readLine()) != null) {
                    result.append(line).append("\n");
                }
                txpFile.close();

                return result.toString();
            } catch (UnsupportedEncodingException e) {
                e.printStackTrace();
            } catch (FileNotFoundException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
```

```
        return "";
    }

    /** Runs executable command
     * @param command
     * @param config the configuration setting
     * @exception IOException
     */
    private Process run(String[] command, String[] config) throws IOException {
        try {
            Runtime rt = Runtime.getRuntime();
            return rt.exec(command, config);

        } catch(Exception e) {
            throw new IOException("Run process error: " + e.getMessage());
        }

    }
}
```