Scraper example

Paul Huygen <paul.huygen@huygen.nl>

13th September 2016
09:07 h.

Abstract

In this document a web-scraper is constructed that scrapes the forum http://web.archive.org/web/20160323072944/http://ragingbull.com, using Python and Beautifulsoup.

Contents

1 Introduction

- Scrape a forum on a website.
- In this case http://web.archive.org/web/20160323072944/http://ragingbull.com.
- Use Python and Beautifulsoup.

1.1 Structure of the forum

The forum consists of a set of *boards* with different subjects. Each board has an identifying number and a name, e.g. board 14242 is about *Current Events*, abbreviated as CEVT. The main page of that board has as URL: http://web.archive.org/web/20160323072944/http://ragingbull.com/board/14242. It contains a table with a list of topics and, when there are too many topics for a single page, references to other URL's that contain lists of older topics. These URL's look like http://web.archive.org/web/20160323072944/http://ragingbull.com/board/14242/page/2.

A topic has as url e.g. http://web.archive.org/web/20160323072944/http://ragingbull.com/topic/1061702 and a title. The page of the topic contains a list of posts.

1.2 What are we going to do?

1. Read the pages of the board and collect the url's of the topics
2. Read the pages of the topics and extract the posts.
3. Wrap each post (text and metadata) in a NAF file.

1.3 Metadata

We need to collect for each post the following metadata:

1. board name and ID.
2. Topic name and ID.
3. Sequence number of the post in the topic.
4. Author ID.
5. Date of the post.

To test whether we have gathered a post with the correct metadata, we can print it as follows:

⟨ *methods of the main program* 2 ⟩ ≡
```
def print_post(board_id, board_name, topic, seq, author, post_date, text):
    print( "Board:   {} ({})".format(board_id, board_name))
    print( "Topic:   {}".format(topic))
    print( "Post nr: {}".format(seq))
    print( "Date:    {}".format(post_date))
    print( "Text: {}".format(text))

◇
```
Fragment defined by 2, 4ac, 5c, 7a, 8a.
Fragment referenced in 8b.
Defines: `print_post` 8c.
Uses: `print` 14a.

2 The program

2.1 Read the command-line

In this demo-phase we can either parse url http://web.archive.org/web/20160719235030/http://ragingbull.com/forum/topic/1051970, or parse a file of which the name is mentioned in the first argument.

⟨ *get program options* 3a ⟩ ≡
```
boardnum = 14242
topicURL = "http://web.archive.org/web/20160719235030/http://ragingbull.com/forum/topic/1051970"

infile = 'none'
if len(sys.argv) > 1:
    infile = sys.argv[1]

◇
```
Fragment referenced in 8b.
Defines: `boardnum` 8c, `topicURL` Never used.

2.2 BeautifulSoup

We will use Python's BeautifulSoup module to extract the posts from the forum.

⟨ *import modules in main program* 3b ⟩ ≡
```
from bs4 import BeautifulSoup
import requests
◇
```
Fragment defined by 3b, 4b, 6, 7b.
Fragment referenced in 8b.
Defines: `BeautifulSoup` 8a, `bs4` Never used, `requests` 8a.

2.3 Extract the posts from a topic page

A topic page contains a number of posts, wrapped in `<article>`/`</article>` tags. Between these two tags we can find:

Post-id: as argument "id" in the `article` tag.
Author name: In a tag "header", in a `div` "author-and-time", in an anchor of class "author-name".

⟨ *methods of the main program* 4a ⟩ ≡

```
def next_article(soup, postnum=0):
     for article in soup.find_all("article"):
         postnum += 1
         header = article.header
         for sp in header.find_all("span"):
             if sp['class'][0] == "postId":
                 postid = sp.string
             elif sp['class'][0] == "time":
                 posttime = sp.string
         for div in header.find_all("div"):
             if div['class'][0] =="author-and-time":
                 for anchor in div.find_all("a"):
                     if anchor['class'][0] == "author-name":
                         author=anchor.string
                         author_url = anchor.href
         text = article.textarea.string
         yield [ postid, posttime, postnum, author, author_url, text ]
```

◇

Fragment defined by 2, 4ac, 5c, 7a, 8a.
Fragment referenced in 8b.
Defines: `nextarticle` Never used.

2.4 Generate the NAF file

Generate the NAF file with the KafNafParserPy package.

⟨ *import modules in main program* 4b ⟩ ≡

```
import KafNafParserPy
```

◇

Fragment defined by 3b, 4b, 6, 7b.
Fragment referenced in 8b.

If you construct a NAF from scratch, it doesn't have a header section. To work around this, we read in a template of a NAF file that contains an empty header. Fill in the header, add a `raw` tag with the textof the post and write out to a file that is named after the ID of the post:

⟨ *methods of the main program* 4c ⟩ ≡

```
def printnaf(post_id, topic, author, post_date, text):
    naf = KafNafParserPy.KafNafParser(filename = 'template.naf')
    naf.set_language("en")
    outtext = Contents_block(text)
    naf.set_raw(outtext.without_bbcode())
    ⟨ create the naf header 5b ⟩
    naf.dump(filename = str(post_id) + ".naf")
```

◇

Fragment defined by 2, 4ac, 5c, 7a, 8a.
Fragment referenced in 8b.
Defines: `printnaf` 8b.

"../template.naf" 5a≡
```
<?xml version="1.0" encoding="UTF-8"?>
<NAF>
  <nafHeader></nafHeader>
</NAF>

◇
```

The following metadata goes in the NAF header:

- Topic
- Author
- Date of the post.

⟨ *create the naf header* 5b ⟩ ≡
```
header = naf.get_header()
fileDesc = KafNafParserPy.CfileDesc()
header.set_fileDesc(fileDesc)
fileDesc.set_title(topic)
fileDesc.set_author(author)
fileDesc.set_creationtime(convert_timestring(post_date))
◇
```
Fragment referenced in 4c.
Uses: `convert_timestring` 5c.

Find the time of the post. The time is expressed as a string like `Mar 22 22:48`. This must be converted to an ISO 8601 format. Therefore, create a datetime object from the time-string. The year does not seem to be included in the timestring. we have to solve this later.

⟨ *methods of the main program* 5c ⟩ ≡
```
def convert_timestring(post_string):
    [ monthname, daynum, time_of_day ] = post_string.split()
    [ hour, minute ] = time_of_day.split(':')
    year = 2016
    pubdate = datetime.datetime(year, monthnums[monthname], int(daynum), int(hour), int(minute))
    return pubdate.isoformat()



◇
```
Fragment defined by 2, 4ac, 5c, 7a, 8a.
Fragment referenced in 8b.
Defines: `convert_timestring` 5b.
Uses: `datetime` 6, `monthnums` 5d.

To convert month-names (e.g. “Jan”) to month-numbers (e.g. 1), use the following dictionary.

⟨ *variables of the main program* 5d ⟩ ≡
```
monthnums = {v: k for k,v in enumerate(calendar.month_abbr)}
◇
```
Fragment referenced in 8b.
Defines: `monthnums` 5c.
Uses: `calendar` 6.

⟨ *import modules in main program* 6 ⟩ ≡
```
    import datetime
    import calendar
    ◇
```
Fragment defined by 3b, 4b, 6, 7b.
Fragment referenced in 8b.
Defines: `calendar` 5d, `datetime` 5c, 8c.

2.5 Remove mark-up from the text

The HTML pages of Ragingbull contain the text od the posts as HTML code or as "bb-code". A concise guide for bb-code can be found here.

tag	**description**	**action**
`[b], [/b]:`	boldface	remove mark-up
`[i], [/i]:`	italic	remove mark-up
`[u], [/u]:`	underline	remove mark-up
`[s], [/s]:`	strike-through	remove tag
`[color], [/color]:`	back-ground color	remove mark-up
`[center], [/center]:`	centered text	remove mark-up
`[quote], [/quote]:`	quotation	Add quotation marks
`[quote={name}], [/quote]:`	quotation	`name said: `` ... ''`
`[url], [/url]:`	Link	remove mark-up
`[url={url}], [/url]:`	Link	Leave the text.
`[img ...], [/img]:`	image	replace by "image"
`[ul], [/ul]:`	Unordened list	remove mark-up
`[ol], [/ol]:`	ordened list	remove mark-up
`[list], [/list]:`	list	remove mark-up
`[li], [/li]:`	list item	
`[code], [/code]:`	Verbatim	
`[table], [/table]:`	table	
`[tr], [/tr]:`	teble row	
`[th], [/th]:`	table heading	
`[td], [/td]:`	table cell	
`[youtube], [/youtube]:`	URL to Youtube	remove mark-up
`[gvideo], [/gvideo]:`	URL to video	remove mark-up

⟨ *methods of the main program* 7a ⟩ ≡

```
class Contents_block:
        def __init__(self,intext):
             self.intext = intext

        def _strip_bbtag(self, intext, tagname):
             s1 = intext.replace('[' + tagname + ']', '')
             return s1.replace('[/' + tagname + ']', '')

        def _strip_bbtagged_substring(self, intext, tagname):
             pattern = re.compile('\[' + tagname + '\].*\[/' + tagname + '\]')
             return re.sub(pattern, '', intext)

        def _replace_bbtagged_substring(self, intext, tagname, repl):
             pattern = re.compile('\[' + tagname + '\].*\[/' + tagname + '\]')
             return re.sub(pattern, repl, intext)

        def _unquote(self, intext):
             out = self._strip_bbtag(intext, 'quote')
             pattern = re.compile('\[quote=(.*)\](.*)\[/quote\]')
             out = re.sub(pattern, '\1 said: "\2"', out)
             return out

        def _un_url(self, intext):
             out = self._strip_bbtag(intext, 'url')
             pattern = re.compile('\[url=(.*)\](.*)\[/url\]')
             out = re.sub(pattern, '\2' + ' (' + '\1' + ')', out)
             return out


        def without_bbcode(self):
             out = self._strip_bbtag(self.intext, 'b')
             out = self._strip_bbtag(out, 'i')
             out = self._strip_bbtag(out, 'u')
             out = self._strip_bbtag(out, 'color')
             out = self._strip_bbtag(out, 'youtube')
             out = self._strip_bbtag(out, 'gvideo')
             out = self._strip_bbtagged_substring(out, 's')
             out = self._strip_bbtagged_substring(out, 'img')
             out = self._unquote(out)
             return out

◇
```

Fragment defined by 2, 4ac, 5c, 7a, 8a.
Fragment referenced in 8b.
Uses: `re` 7b.

⟨ *import modules in main program* 7b ⟩ ≡

```
import re
◇
```

Fragment defined by 3b, 4b, 6, 7b.
Fragment referenced in 8b.
Defines: `re` 7a.

2.6 Test with a single "topic" page

⟨ *methods of the main program* 8a ⟩ ≡

```
def get_testsoup():
    if infile == 'none':
        r = requests.get('http://web.archive.org/web/20160719235030/http://ragingbull.com/forum/topic/
        if r.status_code != 200:
             print("Http request result: {}".format(r.status_code))
             print("Error exit")
             sys.exit()
        soup = BeautifulSoup(r.content, 'lxml')
    else:
        with open(infile, 'r') as content_file:
            content = content_file.read()
        soup = BeautifulSoup(content, 'lxml')
    return soup
◇
```

Fragment defined by 2, 4ac, 5c, 7a, 8a.
Fragment referenced in 8b.
Uses: `BeautifulSoup` 3b, `print` 14a, `requests` 3b.

2.7 The program file

`"../scrape.py"` 8b≡

```
⟨ import modules in main program 3b, ... ⟩
import sys
⟨ variables of the main program 5d ⟩
⟨ methods of the main program 2, ... ⟩

if __name__ == "__main__" :
    ⟨ get program options 3a ⟩
    soup = get_testsoup()
    seq = 0
    for [postid, posttime, postnum, author, author_url, text] in  next_article(soup):
        seq += 1
        printnaf(postid, "topic", author, posttime, text)
◇
```

Uses: `printnaf` 4c.

For now, the program just prints a mock-up of a post:

⟨ *print the testpost* 8c ⟩ ≡

```
print_post(boardnum, "CEVT", "Gallup: life got better", 1, "juddism", datetime.datetime.now(), "Come
◇
```

Fragment never referenced.
Uses: `boardnum` 3a, `datetime` 6, `print_post` 2.

A How to read and translate this document

This document is an example of *literate programming* [1]. It contains the code of all sorts of scripts and programs, combined with explaining texts. In this document the literate programming tool `nuweb` is used, that is currently available from Sourceforge (URL:`nuweb.sourceforge.net`). The advantages of Nuweb are, that it can be used for every programming language and scripting language, that it can contain multiple program sources and that it is very simple.

A.1 Read this document

The document contains *code scraps* that are collected into output files. An output file (e.g. `output.fil`) shows up in the text as follows:

"output.fil" 4a ≡

```
    # output.fil
    < a macro 4b >
    < another macro 4c >
    ◇
```

The above construction contains text for the file. It is labelled with a code (in this case 4a) The constructions between the < and > brackets are macro's, placeholders for texts that can be found in other places of the document. The test for a macro is found in constructions that look like:

< a macro 4b > ≡

```
    This is a scrap of code inside the macro.
    It is concatenated with other scraps inside the
    macro. The concatenated scraps replace
    the invocation of the macro.
```

```
Macro defined by 4b, 87e
Macro referenced in 4a
```

Macro's can be defined on different places. They can contain other macroÂ´s.

< a scrap 87e > ≡

```
    This is another scrap in the macro. It is
    concatenated to the text of scrap 4b.
    This scrap contains another macro:
    < another macro 45b >
```

```
Macro defined by 4b, 87e
Macro referenced in 4a
```

A.2 Process the document

The raw document is named `a_myscrapexamp.w`. Figure 1 shows pathways to translate it into printable/viewable documents and to extract the program sources. Table 1 lists the tools that are

Tool	**Source**	**Description**
gawk	`www.gnu.org/software/gawk/`	text-processing scripting language
M4	`www.gnu.org/software/m4/`	Gnu macro processor
nuweb	`nuweb.sourceforge.net`	Literate programming tool
tex	`www.ctan.org`	Typesetting system
tex4ht	`www.ctan.org`	Convert TEX documents into `xml`/`html`

Table 1: Tools to translate this document into readable code and to extract the program sources

needed for a translation. Most of the tools (except Nuweb) are available on a well-equipped Linux system.

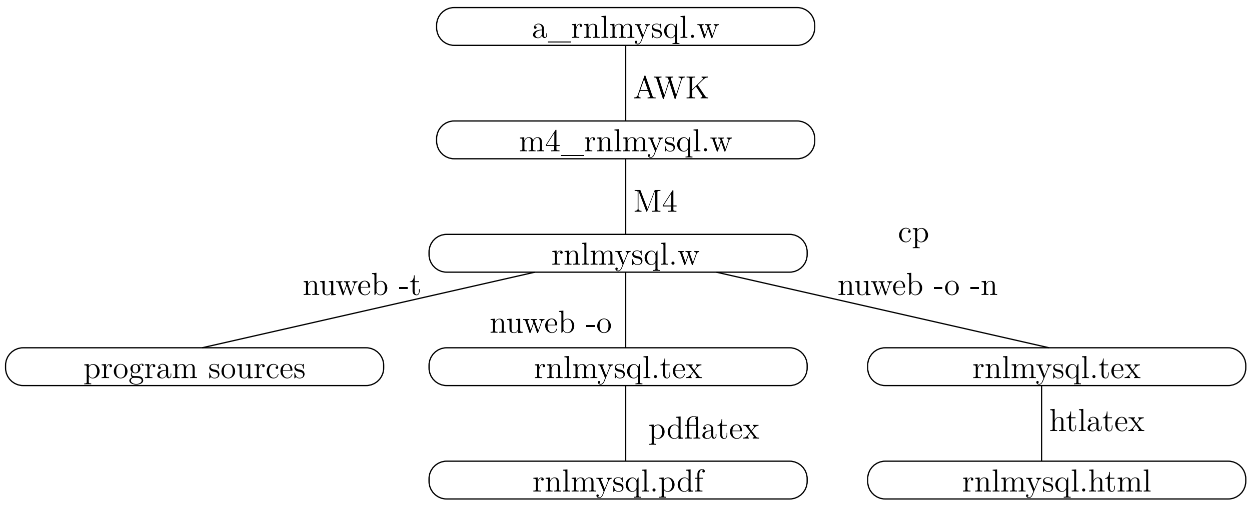

Figure 1: Translation of the raw code of this document into printable/viewable documents and into program sources. The figure shows the pathways and the main files involved.

⟨ *parameters in Makefile* 10a ⟩ ≡

```
        NUWEB=/usr/local/bin/nuweb
```

◇

Fragment defined by 10a, 11b, 12ab, 14c, 17b, 20a.
Fragment referenced in 10b.
Uses: `nuweb` 16a.

A.3 Translate and run

This chapter assembles the Makefile for this project.

`"Makefile"` 10b≡

⟨ *default target* 10c ⟩

⟨ *parameters in Makefile* 10a, ... ⟩

⟨ *impliciete make regels* 13a, ... ⟩
⟨ *expliciete make regels* 11c, ... ⟩
⟨ *make targets* 14a, ... ⟩
◇

The default target of make is `all`.

⟨ *default target* 10c ⟩ ≡

```
        all : ⟨ all targets 11a ⟩
        .PHONY : all
```

◇

Fragment referenced in 10b.
Defines: `all` Never used, `PHONY` 13b.

One of the targets is certainly the PDF version of this document.

⟨ *all targets* 11a ⟩ ≡

```
    myscrapexamp.pdf◇
```

Fragment referenced in 10c.
Uses: `pdf` 14a.

We use many suffixes that were not known by the C-programmers who constructed the `make` utility. Add these suffixes to the list.

⟨ *parameters in Makefile* 11b ⟩ ≡

```
    .SUFFIXES: .pdf .w .tex .html .aux .log .php

    ◇
```

Fragment defined by 10a, 11b, 12ab, 14c, 17b, 20a.
Fragment referenced in 10b.
Defines: `SUFFIXES` Never used.
Uses: `pdf` 14a.

A.4 Pre-processing

To make usable things from the raw input `a_myscrapexamp.w`, do the following:

1. Process `$` characters.
2. Run the m4 pre-processor.
3. Run nuweb.

This results in a LaTeX file, that can be converted into a PDF or a HTML document, and in the program sources and scripts.

A.4.1 Process ‘dollar’ characters

Many “intelligent” TeX editors (e.g. the auctex utility of Emacs) handle `$` characters as special, to switch into mathematics mode. This is irritating in program texts, that often contain `$` characters as well. Therefore, we make a stub, that translates the two-character sequence `\$` into the single `$` character.

⟨ *expliciete make regels* 11c ⟩ ≡

```
    m4_myscrapexamp.w : a_myscrapexamp.w
            gawk '{if(match($$0, "@%")) {printf("%s", substr($$0,1,RSTART-1))} else print}' a_myscrapexamp
              | gawk '{gsub(/[\\][\$$]/, "$$");print}'  > m4_myscrapexamp.w

    ◇
```

Fragment defined by 11cd, 13b, 15a, 17de, 18ab.
Fragment referenced in 10b.
Uses: `print` 14a.

A.4.2 Run the M4 pre-processor

⟨ *expliciete make regels* 11d ⟩ ≡

```
    myscrapexamp.w : m4_myscrapexamp.w
            m4 -P m4_myscrapexamp.w > myscrapexamp.w

    ◇
```

Fragment defined by 11cd, 13b, 15a, 17de, 18ab.
Fragment referenced in 10b.

A.5 Typeset this document

Enable the following:

1. Create a PDF document.
2. Print the typeset document.
3. View the typeset document with a viewer.
4. Create a HTMLdocument.

In the three items, a typeset PDF document is required or it is the requirement itself.

A.5.1 Figures

This document contains figures that have been made by `xfig`. Post-process the figures to enable inclusion in this document.

The list of figures to be included:

⟨ *parameters in Makefile* 12a ⟩ ≡

```
FIGFILES=fileschema

◇
```

Fragment defined by 10a, 11b, 12ab, 14c, 17b, 20a.
Fragment referenced in 10b.
Defines: `FIGFILES` 12b, 17b.

We use the package `figlatex` to include the pictures. This package expects two files with extensions `.pdftex` and `.pdftex_t` for `pdflatex` and two files with extensions `.pstex` and `.pstex_t` for the `latex/dvips` combination. Probably tex4ht uses the latter two formats too.

Make lists of the graphical files that have to be present for latex/pdflatex:

⟨ *parameters in Makefile* 12b ⟩ ≡

```
FIGFILENAMES=$(foreach fil,$(FIGFILES), $(fil).fig)
PDFT_NAMES=$(foreach fil,$(FIGFILES), $(fil).pdftex_t)
PDF_FIG_NAMES=$(foreach fil,$(FIGFILES), $(fil).pdftex)
PST_NAMES=$(foreach fil,$(FIGFILES), $(fil).pstex_t)
PS_FIG_NAMES=$(foreach fil,$(FIGFILES), $(fil).pstex)

◇
```

Fragment defined by 10a, 11b, 12ab, 14c, 17b, 20a.
Fragment referenced in 10b.
Defines: `FIGFILENAMES` Never used, `PDFT_NAMES` 14b, `PDF_FIG_NAMES` 14b, `PST_NAMES` Never used, `PS_FIG_NAMES` Never used.
Uses: `FIGFILES` 12a.

Create the graph files with program `fig2dev`:

⟨ *impliciete make regels* 13a ⟩ ≡
```
%.eps: %.fig
        fig2dev -L eps $< > $@

%.pstex: %.fig
        fig2dev -L pstex $< > $@

.PRECIOUS : %.pstex
%.pstex_t: %.fig %.pstex
        fig2dev -L pstex_t -p $*.pstex $< > $@

%.pdftex: %.fig
        fig2dev -L pdftex $< > $@

.PRECIOUS : %.pdftex
%.pdftex_t: %.fig %.pstex
        fig2dev -L pdftex_t -p $*.pdftex $< > $@

◇
```
Fragment defined by 13a, 14b, 17c.
Fragment referenced in 10b.
Defines: `fig2dev` Never used.

A.5.2 Bibliography

To keep this document portable, create a portable bibliography file. It works as follows: This document refers in the |bibliography| statement to the local `bib`-file `myscrapexamp.bib`. To create this file, copy the auxiliary file to another file `auxfil.aux`, but replace the argument of the command `\bibdata{myscrapexamp}` to the names of the bibliography files that contain the actual references (they should exist on the computer on which you try this). This procedure should only be performed on the computer of the author. Therefore, it is dependent of a binary file on his computer.

⟨ *expliciete make regels* 13b ⟩ ≡
```
bibfile : myscrapexamp.aux /home/paul/bin/mkportbib
        /home/paul/bin/mkportbib myscrapexamp litprog

.PHONY : bibfile
◇
```
Fragment defined by 11cd, 13b, 15a, 17de, 18ab.
Fragment referenced in 10b.
Uses: `PHONY` 10c.

A.5.3 Create a printable/viewable document

Make a PDF document for printing and viewing.

⟨ *make targets* 14a ⟩ ≡

```
pdf : myscrapexamp.pdf

print : myscrapexamp.pdf
        lpr myscrapexamp.pdf

view : myscrapexamp.pdf
        evince myscrapexamp.pdf

◇
```

Fragment defined by 14a, 17a, 20bc.
Fragment referenced in 10b.
Defines: `pdf` 11ab, 14b, `print` 2, 8a, 11c, `view` Never used.

Create the PDF document. This may involve multiple runs of nuweb, the LaTeX processor and the bibTeX processor, and depends on the state of the `aux` file that the LaTeX processor creates as a by-product. Therefore, this is performed in a separate script, `w2pdf`.

The w2pdf script The three processors nuweb, LaTeX and bibTeX are intertwined. LaTeX and bibTeX create parameters or change the value of parameters, and write them in an auxiliary file. The other processors may need those values to produce the correct output. The LaTeX processor may even need the parameters in a second run. Therefore, consider the creation of the (PDF) document finished when none of the processors causes the auxiliary file to change. This is performed by a shell script `w2pdf`.

Note, that in the following `make` construct, the implicit rule `.w.pdf` is not used. It turned out, that make did not calculate the dependencies correctly when I did use this rule.

⟨ *impliciete make regels* 14b ⟩ ≡

```
%.pdf : %.w $(W2PDF)  $(PDF_FIG_NAMES) $(PDFT_NAMES)
        chmod 775 $(W2PDF)
        $(W2PDF) $*

◇
```

Fragment defined by 13a, 14b, 17c.
Fragment referenced in 10b.
Uses: `pdf` 14a, `PDFT_NAMES` 12b, `PDF_FIG_NAMES` 12b.

The following is an ugly fix of an unsolved problem. Currently I develop this thing, while it resides on a remote computer that is connected via the `sshfs` filesystem. On my home computer I cannot run executables on this system, but on my work-computer I can. Therefore, place the following script on a local directory.

⟨ *parameters in Makefile* 14c ⟩ ≡

```
W2PDF=../nuweb/bin/w2pdf
◇
```

Fragment defined by 10a, 11b, 12ab, 14c, 17b, 20a.
Fragment referenced in 10b.
Uses: `nuweb` 16a.

⟨ *directories to create* 14d ⟩ ≡

```
../nuweb/bin ◇
```

Fragment referenced in 20b.
Uses: `nuweb` 16a.

⟨ *expliciete make regels* 15a ⟩ ≡
```
$(W2PDF) : myscrapexamp.w
        $(NUWEB) myscrapexamp.w
◇
```
Fragment defined by 11cd, 13b, 15a, 17de, 18ab.
Fragment referenced in 10b.

"../nuweb/bin/w2pdf" 15b≡
```
#!/bin/bash
# w2pdf -- compile a nuweb file
# usage: w2pdf [filename]
# 20160913 at 0907h: Generated by nuweb from a_myscrapexamp.w
NUWEB=/usr/local/bin/nuweb
LATEXCOMPILER=pdflatex
⟨ filenames in nuweb compile script 15d ⟩
⟨ compile nuweb 15c ⟩

◇
```
Uses: `nuweb` 16a.

The script retains a copy of the latest version of the auxiliary file. Then it runs the four processors nuweb, LaTeX, MakeIndex and bibTeX, until they do not change the auxiliary file or the index.

⟨ *compile nuweb* 15c ⟩ ≡
```
NUWEB=m4_nuweb
⟨ run the processors until the aux file remains unchanged 16b ⟩
⟨ remove the copy of the aux file 15e ⟩
◇
```
Fragment referenced in 15b.

The user provides the name of the nuweb file as argument. Strip the extension (e.g. `.w`) from the filename and create the names of the LaTeX file (ends with `.tex`), the auxiliary file (ends with `.aux`) and the copy of the auxiliary file (add `old.` as a prefix to the auxiliary filename).

⟨ *filenames in nuweb compile script* 15d ⟩ ≡
```
nufil=$1
trunk=${1%%.*}
texfil=${trunk}.tex
auxfil=${trunk}.aux
oldaux=old.${trunk}.aux
indexfil=${trunk}.idx
oldindexfil=old.${trunk}.idx
◇
```
Fragment referenced in 15b.
Defines: `auxfil` 16b, 19ab, `indexfil` 16b, 19a, `nufil` 16a, 19ac, `oldaux` 15e, 16b, 19ab, `oldindexfil` 16b, 19a, `texfil` 16a, 19ac, `trunk` 16a, 19acd.

Remove the old copy if it is no longer needed.

⟨ *remove the copy of the aux file* 15e ⟩ ≡
```
rm $oldaux
◇
```
Fragment referenced in 15c, 18d.
Uses: `oldaux` 15d, 19a.

Run the three processors. Do not use the option `-o` (to suppres generation of program sources) for nuweb, because `w2pdf` must be kept up to date as well.

⟨ *run the three processors* 16a ⟩ ≡

```
$NUWEB $nufil
$LATEXCOMPILER $texfil
makeindex $trunk
bibtex $trunk
◇
```

Fragment referenced in 16b.
Defines: `bibtex` 19cd, `makeindex` 19cd, `nuweb` 10a, 14cd, 15b, 18c.
Uses: `nufil` 15d, 19a, `texfil` 15d, 19a, `trunk` 15d, 19a.

Repeat to copy the auxiliary file and the index file and run the processors until the auxiliary file and the index file are equal to their copies. However, since I have not yet been able to test the `aux` file and the `idx` in the same test statement, currently only the `aux` file is tested.

It turns out, that sometimes a strange loop occurs in which the `aux` file will keep to change. Therefore, with a counter we prevent the loop to occur more than 10 times.

⟨ *run the processors until the aux file remains unchanged* 16b ⟩ ≡

```
LOOPCOUNTER=0
while
  ! cmp -s $auxfil $oldaux
do
  if [ -e $auxfil ]
  then
   cp $auxfil $oldaux
  fi
  if [ -e $indexfil ]
  then
   cp $indexfil $oldindexfil
  fi
  ⟨ run the three processors 16a ⟩
  if [ $LOOPCOUNTER -ge 10 ]
  then
    cp $auxfil $oldaux
  fi;
done
◇
```

Fragment referenced in 15c.
Uses: `auxfil` 15d, 19a, `indexfil` 15d, `oldaux` 15d, 19a, `oldindexfil` 15d.

A.5.4 Create HTML files

HTML is easier to read on-line than a PDF document that was made for printing. We use `tex4ht` to generate HTML code. An advantage of this system is, that we can include figures in the same way as we do for `pdflatex`.

Nuweb creates a LaTeX file that is suitable for `latex2html` if the source file has `.hw` as suffix instead of `.w`. However, this feature is not compatible with tex4ht.

Make html file:

⟨ *make targets* 17a ⟩ ≡

```
html : m4_htmltarget

◇
```

Fragment defined by 14a, 17a, 20bc.
Fragment referenced in 10b.

The HTML file depends on its source file and the graphics files.

Make lists of the graphics files and copy them.

⟨ *parameters in Makefile* 17b ⟩ ≡

```
HTML_PS_FIG_NAMES=$(foreach fil,$(FIGFILES), m4_htmldocdir/$(fil).pstex)
HTML_PST_NAMES=$(foreach fil,$(FIGFILES), m4_htmldocdir/$(fil).pstex_t)
◇
```

Fragment defined by 10a, 11b, 12ab, 14c, 17b, 20a.
Fragment referenced in 10b.
Uses: `FIGFILES` 12a.

⟨ *impliciete make regels* 17c ⟩ ≡

```
m4_htmldocdir/%.pstex : %.pstex
        cp  $< $@

m4_htmldocdir/%.pstex_t : %.pstex_t
        cp  $< $@

◇
```

Fragment defined by 13a, 14b, 17c.
Fragment referenced in 10b.

Copy the nuweb file into the html directory.

⟨ *expliciete make regels* 17d ⟩ ≡

```
m4_htmlsource : myscrapexamp.w
        cp  myscrapexamp.w m4_htmlsource

◇
```

Fragment defined by 11cd, 13b, 15a, 17de, 18ab.
Fragment referenced in 10b.

We also need a file with the same name as the documentstyle and suffix `.4ht`. Just copy the file `report.4ht` from the tex4ht distribution. Currently this seems to work.

⟨ *expliciete make regels* 17e ⟩ ≡

```
m4_4htfildest : m4_4htfilsource
        cp m4_4htfilsource m4_4htfildest

◇
```

Fragment defined by 11cd, 13b, 15a, 17de, 18ab.
Fragment referenced in 10b.

Copy the bibliography.

⟨ *expliciete make regels* 18a ⟩ ≡
```
m4_htmlbibfil : m4_anuwebdir/myscrapexamp.bib
        cp m4_anuwebdir/myscrapexamp.bib m4_htmlbibfil

◇
```
Fragment defined by 11cd, 13b, 15a, 17de, 18ab.
Fragment referenced in 10b.

Make a dvi file with `w2html` and then run `htlatex`.

⟨ *expliciete make regels* 18b ⟩ ≡
```

m4_htmltarget : m4_htmlsource m4_4htfildest $(HTML_PS_FIG_NAMES) $(HTML_PST_NAMES) m4_htmlbibfil
        cp w2html /home/paul/projecten/cltl/emoeco/myscrapexamp/bin
        cd /home/paul/projecten/cltl/emoeco/myscrapexamp/bin && chmod 775 w2html
        cd m4_htmldocdir && /home/paul/projecten/cltl/emoeco/myscrapexamp/bin/w2html myscrapexamp.w

◇
```
Fragment defined by 11cd, 13b, 15a, 17de, 18ab.
Fragment referenced in 10b.

Create a script that performs the translation.

"w2html" 18c≡
```
#!/bin/bash
# w2html -- make a html file from a nuweb file
# usage: w2html [filename]
#  [filename]: Name of the nuweb source file.
`#' m4_header
echo "translate " $1 >w2html.log
NUWEB=/usr/local/bin/nuweb
⟨ filenames in w2html 19a ⟩

⟨ perform the task of w2html 18d ⟩

◇
```
Uses: `nuweb` 16a.

The script is very much like the `w2pdf` script, but at this moment I have still difficulties to compile the source smoothly into HTML and that is why I make a separate file and do not recycle parts from the other file. However, the file works similar.

⟨ *perform the task of w2html* 18d ⟩ ≡
```
⟨ run the html processors until the aux file remains unchanged 19b ⟩
⟨ remove the copy of the aux file 15e ⟩
◇
```
Fragment referenced in 18c.

The user provides the name of the nuweb file as argument. Strip the extension (e.g. `.w`) from the filename and create the names of the LaTeX file (ends with `.tex`), the auxiliary file (ends with `.aux`) and the copy of the auxiliary file (add `old.` as a prefix to the auxiliary filename).

⟨ *filenames in w2html* 19a ⟩ ≡

```
nufil=$1
trunk=${1%%.*}
texfil=${trunk}.tex
auxfil=${trunk}.aux
oldaux=old.${trunk}.aux
indexfil=${trunk}.idx
oldindexfil=old.${trunk}.idx
◇
```

Fragment referenced in 18c.
Defines: `auxfil` 15d, 16b, 19b, `nufil` 15d, 16a, 19c, `oldaux` 15de, 16b, 19b, `texfil` 15d, 16a, 19c, `trunk` 15d, 16a, 19cd.
Uses: `indexfil` 15d, `oldindexfil` 15d.

⟨ *run the html processors until the aux file remains unchanged* 19b ⟩ ≡

```
while
  ! cmp -s $auxfil $oldaux
do
  if [ -e $auxfil ]
  then
   cp $auxfil $oldaux
  fi
  ⟨ run the html processors 19c ⟩
done
⟨ run tex4ht 19d ⟩

◇
```

Fragment referenced in 18d.
Uses: `auxfil` 15d, 19a, `oldaux` 15d, 19a.

To work for HTML, nuweb *must* be run with the `-n` option, because there are no page numbers.

⟨ *run the html processors* 19c ⟩ ≡

```
$NUWEB -o -n $nufil
latex $texfil
makeindex $trunk
bibtex $trunk
htlatex $trunk
◇
```

Fragment referenced in 19b.
Uses: `bibtex` 16a, `makeindex` 16a, `nufil` 15d, 19a, `texfil` 15d, 19a, `trunk` 15d, 19a.

When the compilation has been satisfied, run makeindex in a special way, run bibtex again (I don't know why this is necessary) and then run htlatex another time.

⟨ *run tex4ht* 19d ⟩ ≡

```
tex '\def\filename{{myscrapexamp}{idx}{4dx}{ind}} \input idxmake.4ht'
makeindex -o $trunk.ind $trunk.4dx
bibtex $trunk
htlatex $trunk
◇
```

Fragment referenced in 19b.
Uses: `bibtex` 16a, `makeindex` 16a, `trunk` 15d, 19a.

create the program sources Run nuweb, but suppress the creation of the LaTeX documentation. Nuweb creates only sources that do not yet exist or that have been modified. Therefore make does not have to check this. However, "make" has to create the directories for the sources if they do not yet exist. So, let's create the directories first.

⟨ *parameters in Makefile* 20a ⟩ ≡
```
MKDIR = mkdir -p

◇
```
Fragment defined by 10a, 11b, 12ab, 14c, 17b, 20a.
Fragment referenced in 10b.
Defines: `MKDIR` 20b.

⟨ *make targets* 20b ⟩ ≡
```
DIRS = ⟨ directories to create 14d ⟩

$(DIRS) :
        $(MKDIR) $@

◇
```
Fragment defined by 14a, 17a, 20bc.
Fragment referenced in 10b.
Defines: `DIRS` 20c.
Uses: `MKDIR` 20a.

⟨ *make targets* 20c ⟩ ≡
```
sources : myscrapexamp.w $(DIRS)
        $(NUWEB) myscrapexamp.w

test : sources
        cd .. && python scrape.py 1051970

◇
```
Fragment defined by 14a, 17a, 20bc.
Fragment referenced in 10b.
Uses: `DIRS` 20b.

B References

B.1 Literature

References

[1] Donald E. Knuth. Literate programming. Technical report STAN-CS-83-981, Stanford University, Department of Computer Science, 1983.

B.2 URL's

Nuweb: `nuweb.sourceforge.net`
Apache Velocity: `m4_velocityURL`
Velocitytools: `m4_velocitytoolsURL`
Parameterparser tool: `m4_parameterparserdocURL`
Cookietool: `m4_cookietooldocURL`

VelocityView: `m4_velocityviewURL`
VelocityLayoutServlet: `m4_velocitylayoutservletURL`
Jetty: `m4_jettycodehausURL`
UserBase javadoc: `m4_userbasejavadocURL`
VU corpus Management development site: `http://code.google.com/p/vucom`

C Indexes

C.1 Filenames

C.2 Macro's

C.3 Variables