Extract from old-bailey texts

Paul Huygen <paul.huygen@huygen.nl>

7th November 2017
09:06 h.

Abstract

This nuweb project generates a script to extract texts from the XML documents that have been provided in Old Bailey Online.

Contents

1 Introduction

In the project "old Bailey Online" digital versions of reports of the proceedings in the "Old Bailey" in London have been made available in XML form. In order to use these proceeding in an educational context we generate a script to transfer the XML into NAF format and to load the files with metadata in Amcat.

At the moment we will not directly download the texts from `oldbailey.org`, but use tarball with the collection, obtained from the University of Giessen (BRD.

1.1 The collection with XML files

The corpus consists of a collection of XML files, one for each day that there was a court session. Each XML file is divided up in a `frontMatter` part listing e.g. the judges and a part for each casus of that day. Technically, each part is located in its own `div1` tag. The `div1` tags have the following attributes:

id Concatenation of the character `a`, `f` (frontMatter), `o`, `s` or `t` (trialAccount), a string encoding the session-date, a hyphen and a sequence number. Example: `t18341205-216`

type `trialAccount` or `frontMatter`.

n Sequence number (might be different from the sequence-number in the `id` tag).

We cannot extract all the information that is stored in the XML tags becaude they do not fit in the NAF.

The name of a file with the serssions of one day is a concatenation of `OBC2-` or `OBCPOS2-`, the date encoded as `yyyymmdd` and the extension `.xml`. In files of which the names begin with `OBCPOS2` the words are labeled with POS (Part Of Speech) tags. Currently we cannot use these, so we have to skip these files.

2 The program

2.1 General

We will build a Python script that does the following:

1. read the XML files one by one;
2. Generate separate NAF files from each `div1` section in each XML file.
3. Use the `id` attribute of the `div1` tag as filename for the NAF file.
4. Use the name of the XML file as pubId.
5. Extract the session-date from the `div1` tag and write it as *creation-date* in the NAF.

⟨ *do the work* 2a ⟩ ≡
```
⟨ get path to XML inputfiles (2b corpusdir ) 2g ⟩
⟨ get path for NAF outputfiles (2c nafdir ) 3a ⟩
for filename in os.listdir(corpusdir):
    ⟨ filter proper files and obtain sessiondate (2d filename,2e sessiondatestring ) 3b ⟩
    filepath = os.path.join(corpusdir, filename)
    ⟨ read the XML file and produce NAFs (2f filepath ) 4a ⟩
◇
```
Fragment referenced in 3d.

Environment variable `corpusdir` points to the directory with the XML files and environment variable `corpusdir` points to the directory for the NAF files to be generated.

⟨ *get path to XML inputfiles* 2g ⟩ ≡
```
@1 = os.environ['corpusdir']
◇
```
Fragment referenced in 2a.
Defines: `corpusdir` 2ab.

⟨ *get path for NAF outputfiles* 3a ⟩ ≡
```
@1 = os.environ['nafdir']
```
◇

Fragment referenced in 2a.
Defines: `nafdir` 2c, 7b.

Analyse the filename. Skip the file if it is of the type that contains POS tags. Otherwise, extract the session-date.

⟨ *filter proper files and obtain sessiondate* 3b ⟩ ≡
```
pat = re.compile('OBC2-(\d*).xml')
m = pat.match(@1)
if not m:
    continue
@2 = m.group(1)
```
◇

Fragment referenced in 2a.
Uses: `re` 3c.

⟨ *import modules* 3c ⟩ ≡
```
import re
```
◇

Fragment defined by 3ce, 6a, 7a.
Fragment referenced in 3d.
Defines: `re` 3b, 5a.

Let us generate the structure of the Python script that we are going to make.

`"../bailey_to_naf.py"` 3d≡
```
#!/usr/bin/env python
⟨ import modules 3c, ... ⟩
import os

⟨ methods in bailey_to_naf 4d, ... ⟩

if __name__ == '__main__':
    ⟨ do the work 2a ⟩


```
◇

2.2 Read and write

2.2.1 Parse the XML file

Use the BeautifulSoup module to parse the XML file.

⟨ *import modules* 3e ⟩ ≡
```
from bs4 import BeautifulSoup
```
◇

Fragment defined by 3ce, 6a, 7a.
Fragment referenced in 3d.
Defines: `BeautifulSoup` 4a, `bs4` Never used.

Find the `div1` sections in the XML file. Open for each section a NAF file with the ID of the section as filename and the session-date as timestamp.

⟨ *read the XML file and produce NAFs* 4a ⟩ ≡
```
with open(@1) as file:
    soup = BeautifulSoup(file, 'lxml')
    souptext = soup.find('text')
    for divi in souptext.find_all('div1'):
        ⟨ generate a NAF file (4b divi,4c sessiondatestring ) 7b ⟩
```
◇

Fragment referenced in 2a.
Uses: `sessiondatestring` 2a.

2.2.2 Extract the text from a div section

A `div1` section consists usually of a concatenation of text strings and XML tags that may also contain text-strings and tags. So, to collect all the text strings, find the elements in the tag, print elements that are text strings and recursively collect the text-strings in the tags.

The text to be obtained is enclosed in `p` tags. A brief investigation revealed that the `div1` sections may contain the following tags:

activity Contains quotation e.g "(says this witness)" .
hi Highlight the contained text.
interp Does not contain text, only references in the attributes.
join Does not contain text.
persname Name of a person. Sometimes it does not contain text, only referecnces.
placename Name of a place.
rs Section
u Quote. Should be replaced by quote marks.
xptr Reference without text

We will recursively extract the text from the tags, replace a persname tag without text-string by `Persname` and replace `<u>` and `</u>` tags by quote characters.

⟨ *methods in bailey_to_naf* 4d ⟩ ≡
```
def extract_text_from_tag(tag):
    extracted_text = ""
    for elem in tag.contents:
        if not elem.name:
            extracted_text = extracted_text +  " " + elem
        elif elem.name == 'persname':
            name = extract_text_from_tag(elem)
            if not_a_name(name):
                name = 'Anonymus'
            extracted_text = extracted_text + ' ' + name
        elif elem.name == 'u':
            extracted_text = extracted_text + ' "' + extract_text_from_tag(elem) + '"'
        else:
            extracted_text = extracted_text + ' ' + extract_text_from_tag(elem)
    return extracted_text
```
◇

Fragment defined by 4d, 5abd, 6bc.
Fragment referenced in 3d.

Find out whether the text extracted from a `persname` element contains characters. Otherwise, the tag does probably not contain a name.

⟨ *methods in bailey_to_naf* 5a ⟩ ≡
```
def not_a_name(s):
    pat = re.compile("[A-Za-z]")
    return not pat.search(s)
```
◇

Fragment defined by 4d, 5abd, 6bc.
Fragment referenced in 3d.
Uses: `re` 3c.

⟨ *methods in bailey_to_naf* 5b ⟩ ≡
```
def grab_text_from_xml_division(dsoup):
    grabbed_text = ''
    for par in dsoup.find_all('p'):
        grabbed_text = grabbed_text + '\n' + remove_excessive_linebreaksfrom(extract_text_from_tag(par
    return grabbed_text
```
◇

Fragment defined by 4d, 5abd, 6bc.
Fragment referenced in 3d.
Defines: `grab_text_from_xml` Never used.

⟨ *print the texts from the divi section* 5c ⟩ ≡
```
print("")
for par in @1.find_all('p'):
    print(remove_excessive_linebreaksfrom(extract_text_from_tag(par)))
```
◇

Fragment never referenced.
Uses: `print` 12b.

The extracted text seems to contain lots of linebreaks and double spaces. Let us remove them (admittedly in an awkward way).

⟨ *methods in bailey_to_naf* 5d ⟩ ≡
```
def remove_excessive_linebreaksfrom(s):
    s = s.replace('\n', ' ')
    s = s.replace('      ', ' ')
    s = s.replace('     ', ' ')
    s = s.replace('    ', ' ')
    s = s.replace('   ', ' ')
    s = s.replace('  ', ' ')
    return s
```
◇

Fragment defined by 4d, 5abd, 6bc.
Fragment referenced in 3d.

2.2.3 Generate NAF

⟨ *import modules* 6a ⟩ ≡

```
from KafNafParserPy import KafNafParser
```

◇

Fragment defined by 3ce, 6a, 7a.
Fragment referenced in 3d.
Defines: `KafNafParserPy` Never used.

To generate naf we steal code from Emiel Miltenburg's `text2naf` script.

⟨ *methods in bailey_to_naf* 6b ⟩ ≡

```
def _format_argument(label, value):
    "Format a an argument in an XML tag."
    if value == None:
        return ""
    else:
        return label + '="' + value + '"'
```

◇

Fragment defined by 4d, 5abd, 6bc.
Fragment referenced in 3d.

⟨ *methods in bailey_to_naf* 6c ⟩ ≡

```
def naffile(text, lang, date, uri, source, pubID):
     "Write text to a raw naf file."
     file_start       = '<NAF xml:lang="{}" version="v3">'.format(lang)
     nafheader_start  = '<nafHeader>'

     file_description = '<fileDesc {} {} type="plain text" />'.format( _format_argument("source", sour
                                                                      , _format_argument("creati
                                                                      )
     Id_tag = '<public {} {}/>'.format(_format_argument("publicId", pubID),  _format_argument("uri", u
     nafheader_end    = '</nafHeader>'
     contents_start   = '<raw><' + '![CDATA['
     contents_end     = ']]></raw>'
     rawtext_part     = contents_start + text + contents_end
     file_end         = '</NAF>'
     return '\n'.join( [file_start
                       , nafheader_start
                       , file_description
                       , Id_tag
                       , nafheader_end
                       , rawtext_part
                       , file_end
                       ]
                     )
```

◇

Fragment defined by 4d, 5abd, 6bc.
Fragment referenced in 3d.
Defines: `naffile` 7b.

⟨ *import modules* 7a ⟩ ≡

```
from dateutil.parser import parse
```

◇

Fragment defined by 3ce, 6a, 7a.
Fragment referenced in 3d.
Defines: `dateutil` Never used.

⟨ *generate a NAF file* 7b ⟩ ≡

```
naffilename = @1['id'] + '.naf'
nafpath = os.path.join(nafdir, naffilename)
sessiondate = parse(sessiondatestring)
uri = 'http://cltl.nl/old_bailey/sessionpaper/' + @1['id']
source = 'http://fedora.clarin-d.uni-saarland.de/oldbailey/downloads/OldBaileyCorpus2.zip'
rawtext = grab_text_from_xml_division(@1)
pubid = @1['id']
with open(nafpath, 'w') as naff:
    naff.write(naffile(rawtext, 'en', sessiondate.isoformat(), uri, source, pubid))

```

◇

Fragment referenced in 4a.
Uses: `nafdir` 3a, `naffile` 6c, `sessiondatestring` 2a.

A How to read and translate this document

This document is an example of *literate programming* [**?**]. 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_old_bailey.w`. Figure 1 shows pathways to translate it into print-

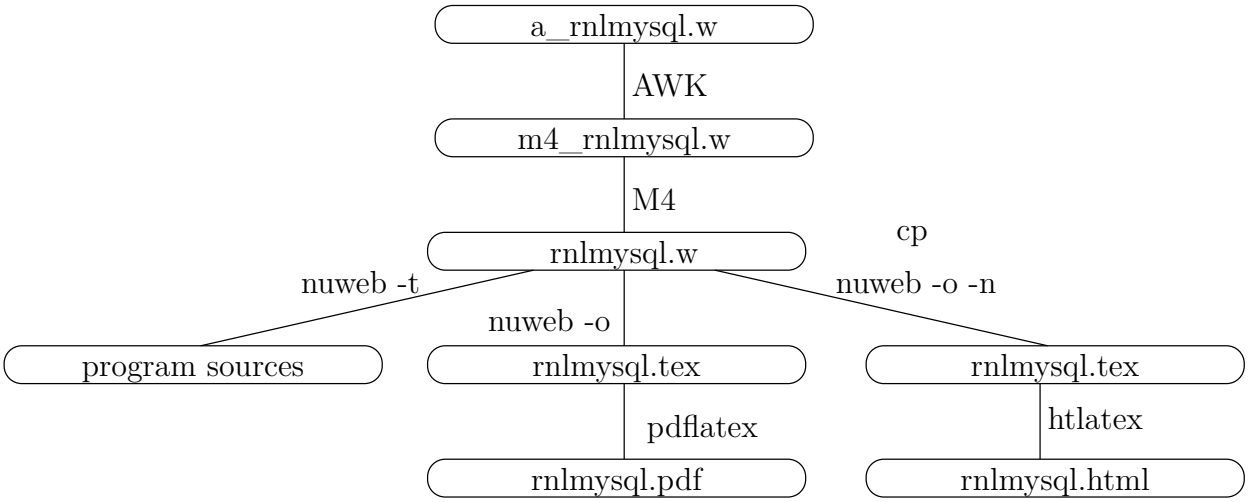

Figure 1: Translation of the raw code of this document into printable/viewable documents and into program sources. The figure shows the pathways and the main files involved.

able/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.

⟨ *parameters in Makefile* 8 ⟩ ≡

```
NUWEB=/usr/local/bin/nuweb
◇
```

Fragment defined by 8, 9d, 10c, 11a, 13a, 15c, 18d.
Fragment referenced in 9a.
Uses: `nuweb` 14c.

A.3 Translate and run

This chapter assembles the Makefile for this project.

```
"Makefile" 9a≡
    ⟨ default target 9b ⟩

    ⟨ parameters in Makefile 8, ... ⟩

    ⟨ impliciete make regels 11b, ... ⟩
    ⟨ expliciete make regels 10a, ... ⟩
    ⟨ make targets 12b, ... ⟩
    ◇
```

The default target of make is `all`.

```
⟨ default target 9b ⟩ ≡
    all : ⟨ all targets 9c ⟩
    .PHONY : all

    ◇
```

Fragment referenced in 9a.
Defines: `all` Never used, `PHONY` 12a.

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

```
⟨ all targets 9c ⟩ ≡
    old_bailey.pdf◇
```

Fragment referenced in 9b.
Uses: `pdf` 12b.

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 9d ⟩ ≡
    .SUFFIXES: .pdf .w .tex .html .aux .log .php

    ◇
```

Fragment defined by 8, 9d, 10c, 11a, 13a, 15c, 18d.
Fragment referenced in 9a.
Defines: `SUFFIXES` Never used.
Uses: `pdf` 12b.

A.4 Pre-processing

To make usable things from the raw input `a_old_bailey.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* 10a ⟩ ≡

```
m4_old_bailey.w : a_old_bailey.w
        gawk '{if(match($$0, "@%")) {printf("%s", substr($$0,1,RSTART-1))} else print}' a_old_bailey.w
          | gawk '{gsub(/[\\][\$$]/, "$$");print}'  > m4_old_bailey.w
```

◇

Fragment defined by 10ab, 12a, 13c, 16bcde.
Fragment referenced in 9a.
Uses: `print` 12b.

A.4.2 Run the M4 pre-processor

⟨ *expliciete make regels* 10b ⟩ ≡

```
old_bailey.w : m4_old_bailey.w
        m4 -P m4_old_bailey.w > old_bailey.w
```

◇

Fragment defined by 10ab, 12a, 13c, 16bcde.
Fragment referenced in 9a.

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* 10c ⟩ ≡

```
FIGFILES=fileschema
```

◇

Fragment defined by 8, 9d, 10c, 11a, 13a, 15c, 18d.
Fragment referenced in 9a.
Defines: `FIGFILES` 11a, 15c.

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* 11a ⟩ ≡

```
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 8, 9d, 10c, 11a, 13a, 15c, 18d.
Fragment referenced in 9a.
Defines: `FIGFILENAMES` Never used, `PDFT_NAMES` 12c, `PDF_FIG_NAMES` 12c, `PST_NAMES` Never used, `PS_FIG_NAMES` Never used.
Uses: `FIGFILES` 10c.

Create the graph files with program `fig2dev`:

⟨ *impliciete make regels* 11b ⟩ ≡

```
%.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 11b, 12c, 16a.
Fragment referenced in 9a.
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 `old_bailey.bib`. To create this file, copy the auxiliary file to another file `auxfil.aux`, but replace the argument of the command `\bibdata{old_bailey}` 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* 12a ⟩ ≡
```
bibfile : old_bailey.aux /home/paul/bin/mkportbib
        /home/paul/bin/mkportbib old_bailey litprog

.PHONY : bibfile
```
◇
Fragment defined by 10ab, 12a, 13c, 16bcde.
Fragment referenced in 9a.
Uses: PHONY 9b.

A.5.3 Create a printable/viewable document

Make a PDF document for printing and viewing.

⟨ *make targets* 12b ⟩ ≡
```
pdf : old_bailey.pdf

print : old_bailey.pdf
        lpr old_bailey.pdf

view : old_bailey.pdf
        evince old_bailey.pdf
```
◇
Fragment defined by 12b, 15b, 19ab.
Fragment referenced in 9a.
Defines: pdf 9cd, 12c, print 5c, 10a, 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* 12c ⟩ ≡
```
%.pdf : %.w $(W2PDF)  $(PDF_FIG_NAMES) $(PDFT_NAMES)
        chmod 775 $(W2PDF)
        $(W2PDF) $*
```
◇
Fragment defined by 11b, 12c, 16a.
Fragment referenced in 9a.
Uses: pdf 12b, PDFT_NAMES 11a, PDF_FIG_NAMES 11a.

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* 13a ⟩ ≡
```
W2PDF=../nuweb/bin/w2pdf
◇
```
Fragment defined by 8, 9d, 10c, 11a, 13a, 15c, 18d.
Fragment referenced in 9a.
Uses: `nuweb` 14c.

⟨ *directories to create* 13b ⟩ ≡
```
../nuweb/bin ◇
```
Fragment referenced in 19a.
Uses: `nuweb` 14c.

⟨ *expliciete make regels* 13c ⟩ ≡
```
$(W2PDF) : old_bailey.w
        $(NUWEB) old_bailey.w
◇
```
Fragment defined by 10ab, 12a, 13c, 16bcde.
Fragment referenced in 9a.

"../nuweb/bin/w2pdf" 13d≡
```
#!/bin/bash
# w2pdf -- compile a nuweb file
# usage: w2pdf [filename]
# 20171107 at 0906h: Generated by nuweb from a_old_bailey.w
NUWEB=/usr/local/bin/nuweb
LATEXCOMPILER=pdflatex
⟨ filenames in nuweb compile script 14a ⟩
⟨ compile nuweb 13e ⟩

◇
```
Uses: `filename` 2a, `nuweb` 14c.

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* 13e ⟩ ≡
```
NUWEB=m4_nuweb
⟨ run the processors until the aux file remains unchanged 15a ⟩
⟨ remove the copy of the aux file 14b ⟩
◇
```
Fragment referenced in 13d.

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* 14a ⟩ ≡

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

Fragment referenced in 13d.
Defines: `auxfil` 15a, 17c, 18a, `indexfil` 15a, 17c, `nufil` 14c, 17c, 18b, `oldaux` 14b, 15a, 17c, 18a, `oldindexfil` 15a, 17c, `texfil` 14c, 17c, 18b, `trunk` 14c, 17c, 18bc.

Remove the old copy if it is no longer needed.

⟨ *remove the copy of the aux file* 14b ⟩ ≡

```
rm $oldaux
◇
```

Fragment referenced in 13e, 17b.
Uses: `oldaux` 14a, 17c.

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* 14c ⟩ ≡

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

Fragment referenced in 15a.
Defines: `bibtex` 18bc, `makeindex` 18bc, `nuweb` 8, 13abd, 17a.
Uses: `nufil` 14a, 17c, `texfil` 14a, 17c, `trunk` 14a, 17c.

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* 15a ⟩ ≡

```
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 14c ⟩
  if [ $LOOPCOUNTER -ge 10 ]
  then
    cp $auxfil $oldaux
  fi;
done
◇
```

Fragment referenced in 13e.
Uses: `auxfil` 14a, 17c, `indexfil` 14a, `oldaux` 14a, 17c, `oldindexfil` 14a.

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* 15b ⟩ ≡

```
html : html/old_bailey.html

◇
```

Fragment defined by 12b, 15b, 19ab.
Fragment referenced in 9a.

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

Make lists of the graphics files and copy them.

⟨ *parameters in Makefile* 15c ⟩ ≡

```
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 8, 9d, 10c, 11a, 13a, 15c, 18d.
Fragment referenced in 9a.
Uses: `FIGFILES` 10c.

⟨ *impliciete make regels* 16a ⟩ ≡

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

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

◇

Fragment defined by 11b, 12c, 16a.
Fragment referenced in 9a.

Copy the nuweb file into the html directory.

⟨ *expliciete make regels* 16b ⟩ ≡

```
html/old_bailey.nw : old_bailey.w
        cp  old_bailey.w html/old_bailey.nw
```

◇

Fragment defined by 10ab, 12a, 13c, 16bcde.
Fragment referenced in 9a.

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* 16c ⟩ ≡

```
m4_4htfildest : m4_4htfilsource
        cp m4_4htfilsource m4_4htfildest
```

◇

Fragment defined by 10ab, 12a, 13c, 16bcde.
Fragment referenced in 9a.

Copy the bibliography.

⟨ *expliciete make regels* 16d ⟩ ≡

```
m4_htmlbibfil : m4_anuwebdir/old_bailey.bib
        cp m4_anuwebdir/old_bailey.bib m4_htmlbibfil
```

◇

Fragment defined by 10ab, 12a, 13c, 16bcde.
Fragment referenced in 9a.

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

⟨ *expliciete make regels* 16e ⟩ ≡

```
html/old_bailey.html : html/old_bailey.nw m4_4htfildest $(HTML_PS_FIG_NAMES) $(HTML_PST_NAMES) m4_html
        cp w2html /bin
        cd /bin && chmod 775 w2html
        cd m4_htmldocdir && /bin/w2html old_bailey.w
```

◇

Fragment defined by 10ab, 12a, 13c, 16bcde.
Fragment referenced in 9a.

Create a script that performs the translation.

```
"w2html" 17a≡
      #!/bin/bash
      # w2html -- make a html file from a nuweb file
      # usage: w2html [filename]
      #  [filename]: Name of the nuweb source file.
      # 20171107 at 0906h: Generated by nuweb from a_old_bailey.w
      echo "translate " $1 >w2html.log
      NUWEB=/usr/local/bin/nuweb
      ⟨ filenames in w2html 17c ⟩

      ⟨ perform the task of w2html 17b ⟩

      ◇
```

Uses: `filename` 2a, `nuweb` 14c.

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 17b ⟩ ≡
      ⟨ run the html processors until the aux file remains unchanged 18a ⟩
      ⟨ remove the copy of the aux file 14b ⟩
      ◇
```

Fragment referenced in 17a.

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 17c ⟩ ≡
      nufil=$1
      trunk=${1%%.*}
      texfil=${trunk}.tex
      auxfil=${trunk}.aux
      oldaux=old.${trunk}.aux
      indexfil=${trunk}.idx
      oldindexfil=old.${trunk}.idx
      ◇
```

Fragment referenced in 17a.
Defines: `auxfil` 14a, 15a, 18a, `nufil` 14ac, 18b, `oldaux` 14ab, 15a, 18a, `texfil` 14ac, 18b, `trunk` 14ac, 18bc.
Uses: `indexfil` 14a, `oldindexfil` 14a.

⟨ *run the html processors until the aux file remains unchanged* 18a ⟩ ≡

```
while
  ! cmp -s $auxfil $oldaux
do
  if [ -e $auxfil ]
  then
   cp $auxfil $oldaux
  fi
  ⟨ run the html processors 18b ⟩
done
⟨ run tex4ht 18c ⟩

◇
```

Fragment referenced in 17b.
Uses: `auxfil` 14a, 17c, `oldaux` 14a, 17c.

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

⟨ *run the html processors* 18b ⟩ ≡

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

Fragment referenced in 18a.
Uses: `bibtex` 14c, `makeindex` 14c, `nufil` 14a, 17c, `texfil` 14a, 17c, `trunk` 14a, 17c.

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* 18c ⟩ ≡

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

Fragment referenced in 18a.
Uses: `bibtex` 14c, `filename` 2a, `makeindex` 14c, `trunk` 14a, 17c.

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* 18d ⟩ ≡

```
MKDIR = mkdir -p

◇
```

Fragment defined by 8, 9d, 10c, 11a, 13a, 15c, 18d.
Fragment referenced in 9a.
Defines: `MKDIR` 19a.

⟨ *make targets* 19a ⟩ ≡

```
DIRS = ⟨ directories to create 13b ⟩

$(DIRS) :
        $(MKDIR) $@
```

◇

Fragment defined by 12b, 15b, 19ab.
Fragment referenced in 9a.
Defines: `DIRS` 19b.
Uses: `MKDIR` 18d.

⟨ *make targets* 19b ⟩ ≡

```
sources : old_bailey.w $(DIRS)
        $(NUWEB) old_bailey.w
```

◇

Fragment defined by 12b, 15b, 19ab.
Fragment referenced in 9a.
Uses: `DIRS` 19a.

B References

B.1 Literature

References

B.2 URL's

Nuweb: nuweb.sourceforge.net

C Indexes

C.1 Filenames

C.2 Macro's

⟨ impliciete make regels 11b, 12c, 16a ⟩ Referenced in 9a.
⟨ import modules 3ce, 6a, 7a ⟩ Referenced in 3d.
⟨ make targets 12b, 15b, 19ab ⟩ Referenced in 9a.
⟨ methods in bailey_to_naf 4d, 5abd, 6bc ⟩ Referenced in 3d.
⟨ parameters in Makefile 8, 9d, 10c, 11a, 13a, 15c, 18d ⟩ Referenced in 9a.
⟨ perform the task of w2html 17b ⟩ Referenced in 17a.
⟨ print the texts from the divi section 5c ⟩ Not referenced.
⟨ read the XML file and produce NAFs 4a ⟩ Referenced in 2a.
⟨ remove the copy of the aux file 14b ⟩ Referenced in 13e, 17b.
⟨ run tex4ht 18c ⟩ Referenced in 18a.
⟨ run the html processors 18b ⟩ Referenced in 18a.
⟨ run the html processors until the aux file remains unchanged 18a ⟩ Referenced in 17b.
⟨ run the processors until the aux file remains unchanged 15a ⟩ Referenced in 13e.
⟨ run the three processors 14c ⟩ Referenced in 15a.

C.3 Variables