# Client-script for VUnlp system

**Paul Huygen <paul.huygen@huygen.nl>**

**26th June 2014**
**13:57 h.**

**Abstract**

This document constructs a client for the VU-NLP system to perform NLP processing on supercomputer Lisa. Features are: 1) Single source document that is easy to distribute; 2) automatic processing of the files in a directory.

## Contents



## 1 Introduction

Natural Language Processing (NLP) is an important tool to extract the meaning from text documents. It enables a methodologic jump in many scientific disciplines, because it enables to analyse

all the documents that are available for a given subject instead of only a fraction of them that a scientist is capable to read in a limited time.

To analyse a document, a computer has to perform a sequence of parsing steps that can be resource-intensive e.g. because machine-learning is involved. Complete parsing of a single document (e.g, news-article) may take up several minutes of processing-time. To investigate a scientific or scholar problem it can be necessary to analyse hundred-thousands of documents. Therefore, supercomputing facilities are needed.

The VU-University has made a facility to enable processing large quantities of documents on supercomputer Lisa, of which the VU-University is co-owner. There is a web-service that enables scientist to upload documents and download the parsed results.

This document describes and implements client software that makes it easy to utilize the VU-nlp service.

# 2 Technique

## 2.1 The webservice

The webservice is a so-called "restful web-server". Table 1 lists the requests that it supports.

| request | type | function | default |
|---|---|---|---|
| `hello` | get | Laten zien dat er iets werkt. | |
| `parsers` | get | Welke parsers zijn er? | Alpino, Stanford |
| `batch` | post | Register a new batch | |
| `batch/status` | get | Test the status of a batch | |
| `batch/start` | put | Start processing a batch | |
| `batch/text` | post | Upload a text | |
| `batch/text/status` | get | Get info about processing status | |
| `batch/text/text` | get | Get text back | |
| `batch/text/log` | head | Get info about presence logfile | |
| `batch/text/log` | get | Get logfile | |
| `batch/text/parse` | get | Get parse | |

Table 1: Web-service requests

⟨ *default settings* 2 ⟩ ≡

```
# Templates for API calls, should be instantiated with .format(url=".."[,filename=".."])
REQUEST_ID =          "{url}/batch"
REQUEST_UPLOAD =      "{url}/batch/{batchid}/text"
REQUEST_STARTBATCH = "{url}/batch/{batchid}/start"
REQUEST_STATUS=       "{url}/batch/{batchid}/text/{textid}/status"
REQUEST_LOGCHECK=       "{url}/batch/{batchid}/text/{textid}/log"
REQUEST_RETRIEVE =    "{url}/getparse/{batchid}/{filename}"
REQUEST_LOGRETRIEVE =    "{url}/batch/{batchid}/text/{textid}/log"
#vunlp.REQUEST_IDCHECK =     "{url}/batch/{batchid}/status"
```

◇

Fragment referenced in 7b.
Defines: `REQUEST_ID` 5, 6, `REQUEST_LOGCHECK` Never used, `REQUEST_LOGRETRIEVE` 6, `REQUEST_RETRIEVE` Never used, `REQUEST_STARTBATCH` 6, `REQUEST_STATUS` Never used, `REQUEST_UPLOAD` 6.
Uses: `url` 4c.

Currently, the client runs on a test-webserver on the local computer of the developer.

⟨ *user-controllable settings* 3a ⟩ ≡
```
DEFAULT_URL = localhost:8090
```
◇
Fragment referenced in 7b.
Defines: DEFAULT_URL 4c.

## 2.2 The client

The client has the following properties:

1. It is a Python script. Python is present on any decent computer.
2. It does not need special VUNLP libraries. However, it may need Python libraries that are generally available but have to be installed on your computer.
3. One of its functions is, that it can be included in a directory with documents to be processed and then takes care to process the documents.
4. Another function is, that it can be used as a library or as a utility for other applications.

The client contains a "client" Python class, that can by used by other python modules. There are two versions of the client. One version performs automatic processing. The other version functions as a utility that can be called from the command line or by other scripts.

The description in the script:

⟨ *description of the script* 3b ⟩ ≡
```
The VU NLP eLab offers a web service to facilitate natural language preprocessing
by running the preprocessing jobs, e.g. on a computer cluster like SARA's lisa.

This module contains a class Client to facilitate talking to the web service
for parsing files.

Command line usage:
 - python client.py init recipe
 - python client.py start batchid
 - python client.py COMMAND batchid textid [< text]

 COMMAND :== "upload" | "check" | "getlog" | "download"

  - init:     Start a new batch with the given recipe and
            return a batch-id to use as a label.
  - upload:   upload the text to be parsed, providing textid
            as a unique label within the batch.
  - start:    Start the batch.
  - check:    check the status of the text with the given batchid/textid.
  - getlog:   retrieve the log of the parser wrt. the text with
            the given batchid/textid.
  - download: retrieve the parser output of the text with the given
            batchid/textid and remove text, parse and log from the database.

@file:   client.py

@author: Wouter van Atteveldt <wouter@vanatteveldt.com> and Paul Huygen <paul.huygen@huygen.nl>

@copyright:   GNU Affero General Public License
```
◇
Fragment referenced in 7a.
Uses: Client 4b, id 4c.

### 2.3 The Client class

Develop a Class "Client" that takes care of the communication with the webservice. All operations are performed via methods of this class.

⟨ *description of class Client* 4a ⟩ ≡
```
Class that communicates with the vu nlp web service to upload, check,
and retrieve parses.
Since each Connection has a unique id, use the same connection object
for all actions on a file.
```
◇

Fragment referenced in 4b.
Uses: `all` 11b, `id` 4c.

⟨ *class Client* 4b ⟩ ≡
```
class Client():
    """
    ⟨ description of class Client 4a ⟩
    """

    ⟨ methods of class Client 4c, ... ⟩


```
◇

Fragment referenced in 7a.
Defines: `Client` 3b, 8a.

On instantion, a Client object needs to obtain the URL of the webservice. Furthermore, it is possible that the object is instantiated to handle an ongoing batch process. In that case it has to obtain the ID of the batch. Finally, we may enable or disable the capacity to download a logfile for each document.

⟨ *methods of class Client* 4c ⟩ ≡
```
def __init__(self, url=DEFAULT_URL, batchid = None, logfiles = True):
  """

  @param url: the url of the web service
  @param batchid: An existing batch id or None
  @param logfiles: If True, download logfiles as well
  """

  log.debug("Execute init method")
  self.url = url
  self._id = batchid
  self.downloadlogfiles = logfiles
```
◇

Fragment defined by 4c, 5, 6.
Fragment referenced in 4b.
Defines: `downloadlogfiles` Never used, `id` 3b, 4a, 6, `logfiles` Never used, `url` 2, 5, 6, `_init__` Never used.
Uses: `DEFAULT_URL` 3a.

The class supports the following methods to process a batch of documents on the webservice:

**batchstatus:** Get status of a running batch.
**existing_batchid:** Find out whether a given ID is known as batchid.
**initbatch:** Initialize a new batch and provide a recipe.

**upload:** Upload a document for a given batch.
**start_batch:**
**check:** Check the parse status of a given document.
**logfile_available:** Check whether a logfile is available and not empty.
**download:** Retrieve a processed document.

Furthermore, the Client class supports convenience methods to issue a request to the server:

**getrequest:** Perform a "get" request.
**postrequest** Perform a "post" request.

The general way to process a bunch of text-documents is as follows:

1. Set up a new batch (initbatch method) and provide a *recipy* (description what the server must do with each document.
2. Upload the documents and receive an ID (*handle*) for each document (upload method).
3. Start the batch (start_batch method). The supercomputer can be used more efficiently when all documents are available for it when processing starts.
4. Check on a regular bases whether documents have been processed and retrieve processed documents and logfiles.

## 2.4 Inplementation of the methods of the Client class

### 2.4.1 Set up a batch

To identify a bunch of files that have to be processed in the same way and to avoid confusion with the documents of other users, The files and operations will be labelled with an ID, the *batchid*. When the batch is set-up the *recipy*, i.e. the parsing operation that has to be performed on the documents, is attached to the batchid.

So the first thing we have to do, is to request a batchid and specify the recipy:

⟨ *methods of class Client* 5 ⟩ ≡

```
def initbatch(self, recipe):
  """
  Initialize a new batch and provide the recipe.
  Initializes variable _id

  @param recipe: the parse-command to apply to the uploaded texts
  @return: the handle to connect to the batch.
  """
  payload = []
  payload = vunlp.pack_recipe(recipe)
  mess = self.postrequest(vunlp.REQUEST_ID.format(url=self.url), payload)
  self._id = vunlp.unpack_batchid(mess)
  return  self._id
```

◇

Fragment defined by 4c, 5, 6.
Fragment referenced in 4b.
Uses: `REQUEST_ID` 2, `url` 4c.

⟨ *methods of class Client* 6 ⟩ ≡

```
#
# Convenience functions
#

    def _path2id(self, path):
        """ convert a path in an id that can be included in an url for a http request.
        @param path:
        @return: string with id
        """
        return path.replace('/', 'X')

#
# Functions to perform requests
#

    def getrequest(self, request):
        """Perform a GET request

        @return: The un-jsonned response or None.
        """
        headers = vunlp.JSONHEADER
        r = requests.get(request, headers = headers)
        r.raise_for_status()
        if r.text.__len__() > 0:
            return r.json()
        else:
            return None

    def postrequest(self, request, payload):
        """Perform a POST request
        @param payload: The body to be uploaded
        @return: The un-jsonned response or None.
        """
        headers = vunlp.JSONHEADER
        r = requests.post(request, headers = headers, data=json.dumps(payload))
        r.raise_for_status()
        if r.text.__len__() > 0:
            return r.json()
        else:
            return None

    def putrequest(self, request, payload):
        """Perform a POST request
        @param payload: The body to be jsonned and uploaded or None
        @return: The un-jsonned response or None.
        """
        headers = vunlp.JSONHEADER
        if payload == None:
            r = requests.put(request, headers = headers)
        else:
            r = requests.put(request, headers = headers,  data=json.dumps(payload))
        r.raise_for_status()
        if r.text.__len__() > 0:
            return r.json()
        else:
            return None

    def _set_check_batchid(self, batchid):
      """Internal function to adopt a given batch-id in stand-alone mode and check existence of the ba
      if batchid != None:
        self._id = batchid
      if self._id == None:
        raise Exception('No batch-id known.')

#  def  get filename(self):
```

## 3 The module

"../client.py" 7a≡

```
#!/usr/bin/env python
⟨ VU python blurb 9 ⟩
"""
⟨ description of the script 3b ⟩
"""
from __future__ import unicode_literals, print_function, absolute_import

⟨ program parameters 7b ⟩
⟨ imports 8c ⟩

⟨ class Client 4b ⟩

if __name__ == '__main__':
  ⟨ script code 8a ⟩



◇
```

⟨ program parameters 7b ⟩ ≡

```
# The following parameters are modifiable by a user to create a real
# stand-alone application.
⟨ user-controllable settings 3a ⟩

#
# Users do probably not need to meddle with the following settings
⟨ default settings 2 ⟩
◇
```

Fragment referenced in 7a.

⟨ *script code* 8a ⟩ ≡

```
logging.basicConfig(level=logging.DEBUG, format='[%(asctime)-15s %(name)s:%(lineno)d %(levelname)s] %(
ok = True
import sys
if len(sys.argv) == 3:
  command, arg1 = sys.argv[1:]
  if command == "init":
    batchid = Client().initbatch(arg1)
    print(batchid)
  elif command == "start":
    Client().start_batch(arg1)
    print("batch " + str(arg1) + " started")
  else:
    ok = False
elif len(sys.argv) == 4:
  command, thisbatchid, filename = sys.argv[1:]
  if command == "upload":
    text = sys.stdin.read()
    Client().upload(text, filename, batchid = thisbatchid)
  elif command == "check":
    print(Client().check(filename, batchid = thisbatchid))
  elif command == "download":
    print(Client().download(filename, batchid = thisbatchid))
  else:
   ok = False
else:
   ok = False
if not ok:
  print(__doc__, file=sys.stderr)
  sys.exit(64)
◇
```

Fragment referenced in 7a.
Uses: `Client` 4b, `print` 14b.

## 3.1 Logging

Set up logging:

⟨ *set up logging* 8b ⟩ ≡

```
log = logging.getLogger(__name__)
◇
```

Fragment never referenced.

⟨ *imports* 8c ⟩ ≡

```
import requests, logging
import urllib
#import vunlp
import json
#import clientinterface
◇
```

Fragment referenced in 7a.

### 3.2 Ramaining things

⟨ *VU python blurb* 9 ⟩ ≡

```
##############################################################################
#          (C) Vrije Universiteit, Amsterdam (the Netherlands)               #
#                                                                            #
# This file is part of vunlp, the VU University NLP e-lab                    #
#                                                                            #
# vunlp is free software: you can redistribute it and/or modify it under     #
# the terms of the GNU Affero General Public License as published by the     #
# Free Software Foundation, either version 3 of the License, or (at your     #
# option) any later version.                                                 #
#                                                                            #
# vunlp is distributed in the hope that it will be useful, but WITHOUT       #
# ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or      #
# FITNESS FOR A PARTICULAR PURPOSE. See the GNU Affero General Public        #
# License for more details.                                                  #
#                                                                            #
# You should have received a copy of the GNU Affero General Public           #
# License along with vunlp.  If not, see <http://www.gnu.org/licenses/>.     #
##############################################################################
◇
```

Fragment referenced in 7a.

## 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_vunlpclient.w`. Figure 1 shows pathways to translate it into

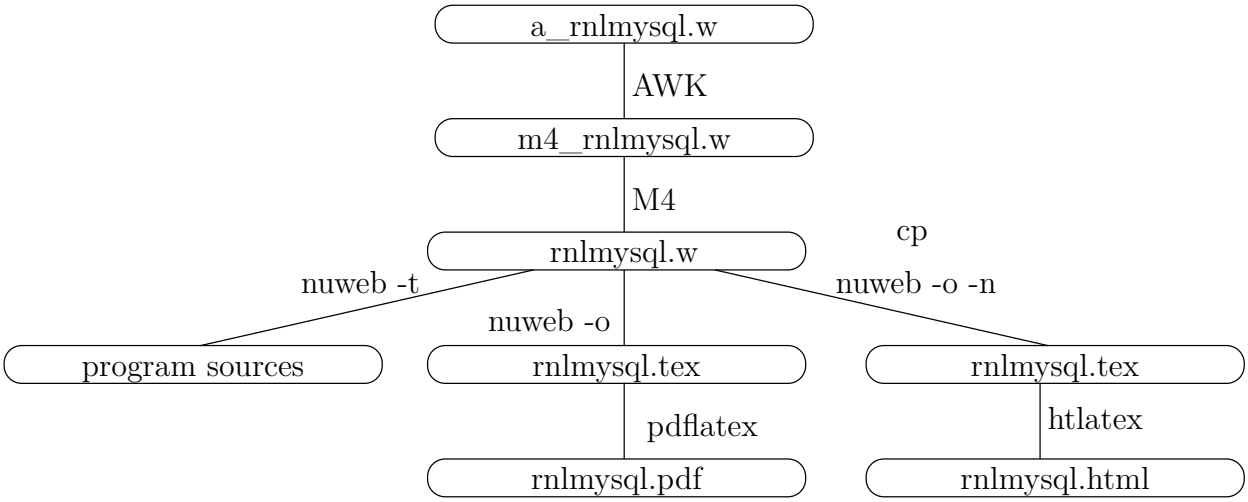


Figure 1: Translation of the raw code of this document into printable/viewable documents and into program sources. The figure shows the pathways and the main files involved.

printable/viewable documents and to extract the program sources. Table 2 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 2: 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* 10 ⟩ ≡

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

◇

Fragment defined by 10, 11d, 12c, 13a, 15a, 17c, 20d.
Fragment referenced in 11a.
Uses: `nuweb` 16c.

## A.3 Translate and run

This chapter assembles the Makefile for this project.

"Makefile" 11a≡
⟨ default target 11b ⟩

⟨ parameters in Makefile 10, ... ⟩

⟨ impliciete make regels 13b, ... ⟩
⟨ expliciete make regels 12a, ... ⟩
⟨ make targets 14b, ... ⟩
◇

The default target of make is `all`.

⟨ default target 11b ⟩ ≡
```
all : ⟨ all targets 11c ⟩
.PHONY : all

```
◇
Fragment referenced in 11a.
Defines: `all` 4a, `PHONY` 14a.

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

⟨ all targets 11c ⟩ ≡
```
vunlpclient.pdf
```
◇
Fragment referenced in 11b.
Uses: `pdf` 14b.

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 11d ⟩ ≡
```
.SUFFIXES: .pdf .w .tex .html .aux .log .php

```
◇
Fragment defined by 10, 11d, 12c, 13a, 15a, 17c, 20d.
Fragment referenced in 11a.
Defines: `SUFFIXES` Never used.
Uses: `pdf` 14b.

## A.4 Pre-processing

To make usable things from the raw input `a_vunlpclient.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* 12a ⟩ ≡

```
m4_vunlpclient.w : a_vunlpclient.w
        gawk '{if(match($$0, "@%")) {printf("%s", substr($$0,1,RSTART-1))} else print}' a_vunlpclient
          | gawk '{gsub(/[\\][\$$]/, "$$");print}'  > m4_vunlpclient.w
```

◇

Fragment defined by 12ab, 14a, 15b, 18bcde.
Fragment referenced in 11a.
Uses: `print` 14b.

### A.4.2 Run the M4 pre-processor

⟨ *expliciete make regels* 12b ⟩ ≡

```
vunlpclient.w : m4_vunlpclient.w
        m4 -P m4_vunlpclient.w > vunlpclient.w
```

◇

Fragment defined by 12ab, 14a, 15b, 18bcde.
Fragment referenced in 11a.

## 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* 12c ⟩ ≡

```
FIGFILES=fileschema
```

◇

Fragment defined by 10, 11d, 12c, 13a, 15a, 17c, 20d.
Fragment referenced in 11a.
Defines: `FIGFILES` 13a, 17c.

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* 13a ⟩ ≡

```
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 10, 11d, 12c, 13a, 15a, 17c, 20d.
Fragment referenced in 11a.
Defines: `FIGFILENAMES` Never used, `PDFT_NAMES` 14c, `PDF_FIG_NAMES` 14c, `PST_NAMES` Never used, `PS_FIG_NAMES` Never used.
Uses: `FIGFILES` 12c.

Create the graph files with program `fig2dev`:

⟨ *impliciete make regels* 13b ⟩ ≡

```
%.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 13b, 14c, 18a.
Fragment referenced in 11a.
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 `vunlpclient.bib`. To create this file, copy the auxiliary file to another file `auxfil.aux`, but replace the argument of the command `\bibdata{vunlpclient}` 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* 14a ⟩ ≡
```
bibfile : vunlpclient.aux /home/paul/bin/mkportbib
        /home/paul/bin/mkportbib vunlpclient litprog

.PHONY : bibfile
```
◇
Fragment defined by 12ab, 14a, 15b, 18bcde.
Fragment referenced in 11a.
Uses: PHONY 11b.

### A.5.3 Create a printable/viewable document

Make a PDF document for printing and viewing.

⟨ *make targets* 14b ⟩ ≡
```
pdf : vunlpclient.pdf

print : vunlpclient.pdf
        lpr vunlpclient.pdf

view : vunlpclient.pdf
        evince vunlpclient.pdf
```
◇
Fragment defined by 14b, 17b, 21ab.
Fragment referenced in 11a.
Defines: pdf 11cd, 14c, print 8a, 12a, 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* 14c ⟩ ≡
```
%.pdf : %.w $(W2PDF)  $(PDF_FIG_NAMES) $(PDFT_NAMES)
        chmod 775 $(W2PDF)
        $(W2PDF) $*
```
◇
Fragment defined by 13b, 14c, 18a.
Fragment referenced in 11a.
Uses: pdf 14b, PDFT_NAMES 13a, PDF_FIG_NAMES 13a.

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* 15a ⟩ ≡
```
W2PDF=../nuweb/bin/w2pdf
```
◇
Fragment defined by 10, 11d, 12c, 13a, 15a, 17c, 20d.
Fragment referenced in 11a.
Uses: `nuweb` 16c.

⟨ *expliciete make regels* 15b ⟩ ≡
```
$(W2PDF) : vunlpclient.w
        $(NUWEB) vunlpclient.w
```
◇
Fragment defined by 12ab, 14a, 15b, 18bcde.
Fragment referenced in 11a.

"../nuweb/bin/w2pdf" 15c≡
```
#!/bin/bash
# w2pdf -- compile a nuweb file
# usage: w2pdf [filename]
# 20140626 at 1357h: Generated by nuweb from a_vunlpclient.w
NUWEB=/usr/local/bin/nuweb
LATEXCOMPILER=pdflatex
```
⟨ *filenames in nuweb compile script* 16a ⟩
⟨ *compile nuweb* 15d ⟩

◇
Uses: `nuweb` 16c.

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* 15d ⟩ ≡
```
NUWEB=nuweb
```
⟨ *run the processors until the aux file remains unchanged* 17a ⟩
⟨ *remove the copy of the aux file* 16b ⟩
◇
Fragment referenced in 15c.
Uses: `nuweb` 16c.

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* 16a ⟩ ≡

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

Fragment referenced in 15c.
Defines: `auxfil` 17a, 19c, 20a, `indexfil` 17a, 19c, `nufil` 16c, 19c, 20b, `oldaux` 16b, 17a, 19c, 20a, `oldindexfil` 17a, 19c, `texfil` 16c, 19c, 20b, `trunk` 16c, 19c, 20bc.

Remove the old copy if it is no longer needed.

⟨ *remove the copy of the aux file* 16b ⟩ ≡

```
rm $oldaux
◇
```

Fragment referenced in 15d, 19b.
Uses: `oldaux` 16a, 19c.

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* 16c ⟩ ≡

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

Fragment referenced in 17a.
Defines: `bibtex` 20bc, `makeindex` 20bc, `nuweb` 10, 15acd, 19a.
Uses: `nufil` 16a, 19c, `texfil` 16a, 19c, `trunk` 16a, 19c.

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* 17a ⟩ ≡

```
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 16c ⟩
  if [ $LOOPCOUNTER -ge 10 ]
  then
    cp $auxfil $oldaux
  fi;
done
◇
```

Fragment referenced in 15d.
Uses: `auxfil` 16a, 19c, `indexfil` 16a, `oldaux` 16a, 19c, `oldindexfil` 16a.

### 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* 17b ⟩ ≡

```
html : m4_htmltarget


◇
```

Fragment defined by 14b, 17b, 21ab.
Fragment referenced in 11a.

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

Make lists of the graphics files and copy them.

⟨ *parameters in Makefile* 17c ⟩ ≡

```
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 10, 11d, 12c, 13a, 15a, 17c, 20d.
Fragment referenced in 11a.
Uses: `FIGFILES` 12c.

⟨ *impliciete make regels* 18a ⟩ ≡

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

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

◇

Fragment defined by 13b, 14c, 18a.
Fragment referenced in 11a.

Copy the nuweb file into the html directory.

⟨ *expliciete make regels* 18b ⟩ ≡

```
m4_htmlsource : vunlpclient.w
        cp  vunlpclient.w m4_htmlsource
```

◇

Fragment defined by 12ab, 14a, 15b, 18bcde.
Fragment referenced in 11a.

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* 18c ⟩ ≡

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

◇

Fragment defined by 12ab, 14a, 15b, 18bcde.
Fragment referenced in 11a.

Copy the bibliography.

⟨ *expliciete make regels* 18d ⟩ ≡

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

◇

Fragment defined by 12ab, 14a, 15b, 18bcde.
Fragment referenced in 11a.

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

⟨ *expliciete make regels* 18e ⟩ ≡

```
m4_htmltarget : m4_htmlsource m4_4htfildest $(HTML_PS_FIG_NAMES) $(HTML_PST_NAMES) m4_htmlbibfil
        cp w2html /bin
        cd /bin && chmod 775 w2html
        cd m4_htmldocdir && /bin/w2html vunlpclient.w
```

◇

Fragment defined by 12ab, 14a, 15b, 18bcde.
Fragment referenced in 11a.

Create a script that performs the translation.

```
"w2html" 19a≡
      #!/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 19c ⟩

      ⟨ perform the task of w2html 19b ⟩

      ◇
```

Uses: `nuweb` 16c.

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 19b ⟩ ≡
      ⟨ run the html processors until the aux file remains unchanged 20a ⟩
      ⟨ remove the copy of the aux file 16b ⟩
      ◇
```

Fragment referenced in 19a.

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 19c ⟩ ≡
      nufil=$1
      trunk=${1%%.*}
      texfil=${trunk}.tex
      auxfil=${trunk}.aux
      oldaux=old.${trunk}.aux
      indexfil=${trunk}.idx
      oldindexfil=old.${trunk}.idx
      ◇
```

Fragment referenced in 19a.
Defines: `auxfil` 16a, 17a, 20a, `nufil` 16ac, 20b, `oldaux` 16ab, 17a, 20a, `texfil` 16ac, 20b, `trunk` 16ac, 20bc.
Uses: `indexfil` 16a, `oldindexfil` 16a.

⟨ *run the html processors until the aux file remains unchanged* 20a ⟩ ≡

```
while
  ! cmp -s $auxfil $oldaux
do
  if [ -e $auxfil ]
  then
   cp $auxfil $oldaux
  fi
  ⟨ run the html processors 20b ⟩
done
⟨ run tex4ht 20c ⟩

◇
```

Fragment referenced in 19b.
Uses: `auxfil` 16a, 19c, `oldaux` 16a, 19c.

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

⟨ *run the html processors* 20b ⟩ ≡

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

Fragment referenced in 20a.
Uses: `bibtex` 16c, `makeindex` 16c, `nufil` 16a, 19c, `texfil` 16a, 19c, `trunk` 16a, 19c.

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* 20c ⟩ ≡

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

Fragment referenced in 20a.
Uses: `bibtex` 16c, `makeindex` 16c, `trunk` 16a, 19c.

*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* 20d ⟩ ≡

```
MKDIR = mkdir -p

◇
```

Fragment defined by 10, 11d, 12c, 13a, 15a, 17c, 20d.
Fragment referenced in 11a.
Defines: `MKDIR` 21a.

⟨ *make targets* 21a ⟩ ≡
```
DIRS = ⟨ directories to create ? ⟩

$(DIRS) :
        $(MKDIR) $@

◇
```
Fragment defined by 14b, 17b, 21ab.
Fragment referenced in 11a.
Defines: DIRS 21b.
Uses: MKDIR 20d.

⟨ *make targets* 21b ⟩ ≡
```
sources : vunlpclient.w $(DIRS)
        $(NUWEB) vunlpclient.w

jetty : sources
        cd .. && mvn jetty:run

◇
```
Fragment defined by 14b, 17b, 21ab.
Fragment referenced in 11a.
Uses: DIRS 21a.

# 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

⟨ all targets 11c ⟩ Referenced in 11b.
⟨ class Client 4b ⟩ Referenced in 7a.
⟨ compile nuweb 15d ⟩ Referenced in 15c.
⟨ default settings 2 ⟩ Referenced in 7b.
⟨ default target 11b ⟩ Referenced in 11a.
⟨ description of class Client 4a ⟩ Referenced in 4b.
⟨ description of the script 3b ⟩ Referenced in 7a.
⟨ directories to create ? ⟩ Referenced in 21a.
⟨ expliciete make regels 12ab, 14a, 15b, 18bcde ⟩ Referenced in 11a.
⟨ filenames in nuweb compile script 16a ⟩ Referenced in 15c.
⟨ filenames in w2html 19c ⟩ Referenced in 19a.
⟨ impliciete make regels 13b, 14c, 18a ⟩ Referenced in 11a.
⟨ imports 8c ⟩ Referenced in 7a.
⟨ make targets 14b, 17b, 21ab ⟩ Referenced in 11a.
⟨ methods of class Client 4c, 5, 6 ⟩ Referenced in 4b.
⟨ parameters in Makefile 10, 11d, 12c, 13a, 15a, 17c, 20d ⟩ Referenced in 11a.
⟨ perform the task of w2html 19b ⟩ Referenced in 19a.
⟨ program parameters 7b ⟩ Referenced in 7a.
⟨ remove the copy of the aux file 16b ⟩ Referenced in 15d, 19b.
⟨ run tex4ht 20c ⟩ Referenced in 20a.
⟨ run the html processors 20b ⟩ Referenced in 20a.
⟨ run the html processors until the aux file remains unchanged 20a ⟩ Referenced in 19b.
⟨ run the processors until the aux file remains unchanged 17a ⟩ Referenced in 15d.
⟨ run the three processors 16c ⟩ Referenced in 17a.
⟨ script code 8a ⟩ Referenced in 7a.
⟨ set up logging 8b ⟩ Not referenced.
⟨ user-controllable settings 3a ⟩ Referenced in 7b.
⟨ VU python blurb 9 ⟩ Referenced in 7a.

## C.3 Variables

`all`: 4a, 11b.
`auxfil`: 16a, 17a, 19c, 20a.
`bibtex`: 16c, 20bc.
`Client`: 3b, 4b, 8a.
`DEFAULT_URL`: 3a, 4c.
`DIRS`: 21a, 21b.
`downloadlogfiles`: 4c.
`fig2dev`: 13b.
`FIGFILENAMES`: 13a.
`FIGFILES`: 12c, 13a, 17c.
`id`: 3b, 4a, 4c, 6.
`indexfil`: 16a, 17a, 19c.
`logfiles`: 4c.
`makeindex`: 16c, 20bc.
`MKDIR`: 20d, 21a.
`nufil`: 16a, 16c, 19c, 20b.
`nuweb`: 10, 15acd, 16c, 19a.
`oldaux`: 16a, 16b, 17a, 19c, 20a.
`oldindexfil`: 16a, 17a, 19c.
`pdf`: 11cd, 14b, 14c.
`PDFT_NAMES`: 13a, 14c.
`PDF_FIG_NAMES`: 13a, 14c.
`PHONY`: 11b, 14a.
`print`: 8a, 12a, 14b.
`PST_NAMES`: 13a.
`PS_FIG_NAMES`: 13a.