

Bilingual NLP pipeline

Paul Huygen <paul.huygen@huygen.nl>

1st March 2018
14:58 h.

Abstract

This is a description and documentation of the installation of the Newsreader-pipeline¹. It is an instrument to annotate Dutch or English documents with NLP tags. The documents have to be stored in *Newsreader Annotation Format* (NAF [1]).

Contents

1	Introduction	3
1.1	Modules of the pipeline	3
1.2	Reproducibility	3
2	Structure of the pipeline	6
2.1	Expected resources	6
3	Construct the infra-structure	6
3.1	File-structure	8
3.2	Download resources	11
3.3	Java	12
3.4	Maven	13
3.5	Maven	13
3.6	Python	14
3.6.1	Python packages	17
3.6.2	Pynaf hack	18
3.7	Perl	18
3.8	Spotlight	21
3.8.1	Set spotlight host and port	21
3.8.2	Install spotlight servers	21
3.8.3	Check/start the Spotlight server	22
3.9	Download materials	27
4	Shared libraries	28
4.1	Autoconf	28
4.2	libxml2 and libxslt	29
4.3	Alpino	29
4.4	Treetagger	30
4.5	Timbl and Ticcutils	33
4.6	Svmlib	34
4.7	The Boost library	35

1. <http://www.newsreader-project.eu/files/2012/12/NWR-D4-2-2.pdf>

5	Install the modules	35
5.1	Parameters in module-scripts	37
5.1.1	Tokeniser	38
5.1.2	Topic detection tool.	38
5.1.3	Morphosyntactic Parser and Alpino	38
5.1.4	Pos tagger	39
5.1.5	Named entity recognition (NERC)	39
5.1.6	Word-sense disambiguation (WSD)	39
5.1.7	NED	40
5.1.8	Dark-entity relinker	40
5.1.9	Heideltime	40
5.1.10	Ontotagger, Framenet-SRL and nominal events	41
5.1.11	NED-reranker	42
5.1.12	Wikify module	42
5.1.13	UKB	42
5.1.14	IMS-WSD	43
5.1.15	Semantic Role labelling	43
5.1.16	SRL server for English	43
5.1.17	srl-Dutch nominals	44
5.1.18	FBK-time, FBK-temprel, FBK-causalrel	44
5.1.19	Factuality	45
5.1.20	Opinion miner	45
5.1.21	Event coreference	46
5.1.22	Corefgraph	46
5.2	Constituent parser	47
6	Utilities	47
6.1	Language detection	47
6.2	Run-script and test-script	48
7	Miscellaneous	52
7.1	Locate the path to the script itself	52
7.2	Logging	53
A	How to read and translate this document	53
A.1	Read this document	53
A.2	Process the document	54
A.3	The Makefile for this project.	55
A.4	Get Nuweb	56
A.5	Pre-processing	56
A.5.1	Process ‘dollar’ characters	57
A.5.2	Run the M4 pre-processor	57
A.6	Typeset this document	57
A.6.1	Figures	57
A.6.2	Bibliography	59
A.6.3	Create a printable/viewable document	59
A.6.4	Create HTML files	62
A.7	Perform the installation	65
A.8	Test whether it works	66
A.9	Restore paths after transplantation	66
B	References	67
B.1	Literature	67

C Indexes	67
C.1 Filenames	67
C.2 Macro's	68
C.3 Variables	69

1 Introduction

This document describes the installation of a pipeline that annotates texts in order to extract knowledge. The pipeline has been set up as part of the newsreader ² project. It accepts and produces texts in the NAF (Newsreader Annotation Format) format.

Apart from describing the pipeline set-up, the document actually constructs the pipeline. The pipeline has been installed on a (Ubuntu) Linux computer.

The installation has been parameterised. The locations and names that you read (and that will be used to build the pipeline) have been read from variables in file `inst.m4` in the `nuweb` directory.

The installed pipeline is bi-lingual. It is capable to annotate Dutch and English texts. It recognizes the language from the “lang” attribute of the NAF element of the document. Some of the modules are specific for a single language, other modules support both languages. As a result, there must be two pathways to lead a document through the pipeline, one for English and one for Dutch.

The pipeline is a concatenation of independent software modules, each of which reads a NAF document from standard input and produces another NAF document on standard output.

The aim is, to install the pipeline from open-source modules that can e.g. be obtained from Github. However, that aim is only partially fulfilled. Some of the modules still contain elements that are not open-source or data that are not freely available. Because of lack of time, the current version of the installer installs the English pipeline from a frozen repository of the Newsreader Project.

The NLPP pipeline can be seen as constructed in three parts: 1) The software that is needed to run the pipeline, e.g. compilers and interpreters; 2) the modules themselves and 3) scripts to make the modules operate on a document.

1.1 Modules of the pipeline

Table 2 lists the modules in the pipeline. The column *source* indicates the origin of the module. The modules are obtained in one of the following ways:

1. If possible, the module is directly obtained from an open-source repository like Github.
2. Some modules have not been officially published in a repository. These modules have been packed in a tar-ball that can be obtained by the author. In table 2 this has been indicated as SNAPSHOT.

The modules themselves use other utilities like dependency-taggers and POS taggers. These utilities are listed in table 1.

1.2 Reproducibility

An important goal of this pipeline is, to achieve reproducibility. It means, that at some point in the future the annotation could be re-done on the document and it should produce a result that is identical as the result of the original annotation. In our case, reproducibility involves the following aspects:

- The annotated document ought to contain documentation about the annotation process: What modules have been applied, what was the version of the software of each module, Which resources have been used and what was the version of the resources.

2. <http://www.newsreader-project.eu>

Module	Version	Section	Source
KafNafParserPy	1.87	3.6.1	Github
Alpino	21088	4.3	RUG
Ticcutils	0.7	4.5	ILK
Timbl	6.4.6	4.5	ILK
Treetagger	3.2	4.4	Uni. München
Spotlight server	0.7	3.8	Spotlight

Table 1: List of the utilities to be installed. Column description: **directory:** Name of the subdirectory below *mod* in which it is installed; **Source:** From where the module has been obtained; **script:** Script to be included in a pipeline.

Module	Source	Resources	Section	Commit	Script	language
Tokenizer	Github	Java	5.1.1	1a5b...	tok	en/nl
Topic detection	Github	Java	5.1.2	31b0...	topic	en/nl
Morpho-syntactic parser	Github	Python, Alpino	5.1.3	2960...	mor	nl
POS-tagger	snapshot		??	...	pos	en
Named-entity rec/class	Github		5.1.5	e619...	nerc	en/nl
Dark-entity relinker	Github		5.1.8	d788...	nerc	en/nl
Constituent parser	snapshot		??	...	constpars	en
Word-sense disamb. nl	Github		5.1.6	eae1...	wsd	nl
Word-sense disamb. en	snapshot		5.1.14	...	ewsd	en
Named entity/DBP	snapshot		5.1.7	...	ned	en/nl
NED reranker	snapshot		5.1.11	...	nedrerscript	en
Wikify	snapshot		5.1.12	...	wikify	en
UKB	snapshot		5.1.13	...	ukb	en
Coreference-base	snapshot		??	...	coref-graph	en
Heideltime	Github		5.1.9	76ee...	heidelttime	nl
Onto-tagger	Github		5.1.10	31b0...	onto	nl
Semantic Role labeling nl	Github		5.1.17	0602...	srl	nl
Semantic Role labeling en	snapshot		??	...	eSRL	en
Nominal Event ann.	Github		5.1.10	31b0...	nomevent	nl
SRL dutch nominals	Github		5.1.17	fcf3...	srl-dutch-nominals	nl
Framenet-SRL	Github		5.1.10	31b0...	framesrl	nl
FBK-time	snapshot		??	...	FBK-time	en
FBK-temprel	snapshot		??	...	FBK-temprel	en
FBK-causalrel	snapshot		??	...	FBK-causalrel	en
Opinion-miner	Github		5.1.20	93cd...	opinimin	en/nl
Event-coref	Github		5.1.21	5c3e...	evcoref	en/nl
Factuality tagger	Github		5.1.19	412f...	factuality	en
Factuality tagger	Github		5.1.19	1f47...	factuality	nl

Table 2: List of the modules to be installed. Column description: **directory**: Name of the subdirectory below subdirectory *modules* in which it is installed; **source**: From where the module has been obtained; **commit**: Commit-name or version-tag **script**: Script to be included in a pipeline.

- The source code of the modules as well as resources like data-sets and programming languages should be available from open repository.
- The repositories of the resources should use some versioning system enabling to re-use the version that has been used originally.

A problem in some cases is, that we need to use utilities that are supplied by external parties, and we do not have control about their methods of publication and version management. Examples of such utilities are the compilers for programming languages like Java, Python and parsers like Alpino.

Therefore, we have the following policy to achieve reproducibility:

- Each of the modules writes in the output NAF its own version, and details about the used resources in sufficient detail to enable re-processing.
- It is assumed that when a programming language (e.g. Java, Python) is used, annotation can be reproducible when the major versions coincide.
- A script is constructed that reproducibly builds an environment for the pipeline on some software/hardware platform (e.g. Linux on X64 CPU), using utilities that have been stored in some non-open repository (to preclude copyright-problems).

2 Structure of the pipeline

The finished pipeline consists of:

- A directory that contains for each module an directory with the module in installed form.
- A script that reads an input naf file or plain text file from standard in and produces an annotated NAF file on standard out.
- A script that must be “sourced” in order to find the resources that the modules need to find.

The directory with the modules must be relocatable and immutable. That means that scripts in modules do not have write permissions on the module directory and that they have to find other files on path-descriptions relative to the current path of the script itself.

2.1 Expected resources

In order to run the modules expect the following:

- Instruction `java` invokes Java 1.8;
- Instruction `python` invokes Python 3.6;
- Instruction `Perl` invokes Perl 5;
- Variable `TMPDIR` points to a user-writable directory.

3 Construct the infra-structure

In this section we will generate a script that set up an infra-structure in which the pipeline can be exploited. An attempt is made to make as little as possible presumptions about the services that the host provides.

We need to set up the following:

- Java Version 1.8
- Maven (Gradle?)
- Python version 3.6
- Python packages
- Autoconf
- ...

Let us generate a script to do the work:

```
"../env/bin/make_infrastructure" 7a≡
#!/bin/bash
  < get location of the script (7b DIR ) 53a >
  cd $DIR
  source ../../progenv
  echo make_infrastructure 'date':
  echo ' '
  < next part (7c Initialize ) 7p >
  < init make_infrastructure 8e, ... >
  < next part (7d Java ) 7p >
  < set up Java 13a >
  < next part (7e Maven ) 7p >
  < set up Maven 14b >
  < next part (7f Python ) 7p >
  < set up Python 15b, ... >
  < next part (7g autoconf ) 7p >
  < set up autoconf 29a >
  < next part (7h Perl ) 7p >
  < conditionally install Perl 20a >
  < next part (7i Shared libs ) 7p >
  < install shared libs 29c >
  < next part (7j Alpino ) 7p >
  < install Alpino 30a >
  < next part (7k Spotlight ) 7p >
  < install the Spotlight server 22a >
  < next part (7l Treetagger ) 7p >
  < install the treetagger utility 30c, ... >
  < install the ticcutils utility 33h >
  < install the timbl utility 34a >
  < next part (7m Svmlib ) 7p >
  < install svmlib 35a >
  < next part (7n Boost ) 7p >
  < install boost 35b >

  ◇

  < make scripts executable 7o > ≡
    chmod 775 ../env/bin/make_infrastructure
  ◇
```

Fragment defined by 7o, 8c, 17a, 24f, 36a, 47e, 65c.

Fragment referenced in 65d.

```
< next part 7p > ≡
  echo ' '
  echo make_infrastructure 'date': @1
  echo ' '
  ◇
```

Fragment referenced in 7a.

Let us also make a script that cleans up the infra-structure after the installation.

```

"../env/bin/clean_infrastructure" 8a≡
    #!/bin/bash
    < get location of the script (8b DIR ) 53a >
    cd $DIR
    source ../../progenv
    < init make_infrastructure 8e, ... >
    < clean up after installation 14g >
    ◇

< make scripts executable 8c > ≡
    chmod 775 ../env/bin/clean_infrastructure
    ◇

```

Fragment defined by 7o, 8c, 17a, 24f, 36a, 47e, 65c.

Fragment referenced in 65d.

Before we begin, we can try whether commands that we need to use actually exist and stop execution otherwise.

```

< test presence of command 8d > ≡
    which @1 >/dev/null
    if
        [ $? -ne 0 ]
    then
        echo "Please install @1"
        exit 4
    fi
    ◇

```

Fragment referenced in 8e.

Uses: install 66a.

```

< init make_infrastructure 8e > ≡
    < test presence of command (8f git ) 8d >
    < test presence of command (8g tar ) 8d >
    < test presence of command (8h unzip ) 8d >
    < test presence of command (8i tcsh ) 8d >
    < test presence of command (8j hg ) 8d >
    ◇

```

Fragment defined by 8e, 11b.

Fragment referenced in 7a, 8a.

3.1 File-structure

Let us set up the pipeline in a directory-structure that looks like figure 1. The directories have the following functions.

socket: The directory in the host where the pipeline is to be implemented.

root: The root of the pipeline directory-structure.

nuweb: This directory contains this document and everything to create the pipeline from the open sources of the modules.

modules: Contains subdirectories with the NLP modules that can be applied in the pipeline.

bin: Contains for each of the applicable modules a script that reads NAF input, passes it to the module in the modules directory and produces the output on standard out. Furthermore,

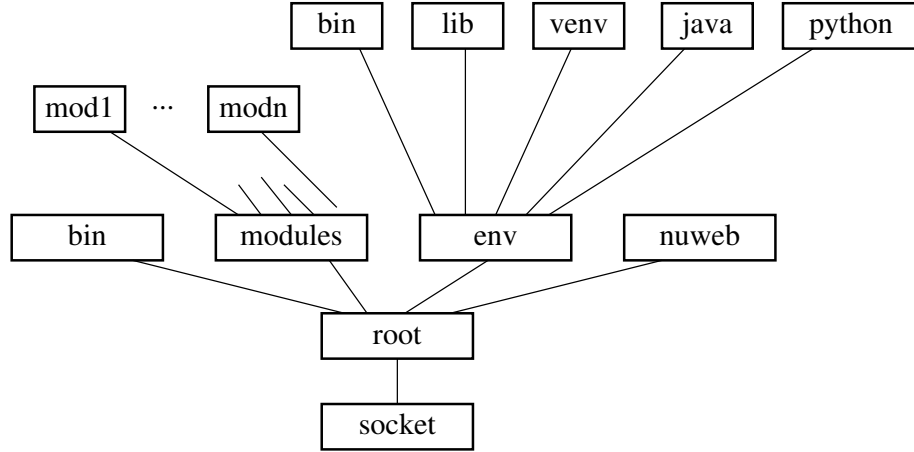


Figure 1: Directory-structure of the pipeline (see text).

the subdirectory contains the script `install_modules` that performs the installation, and a script `test` that shows that the pipeline works in a trivial case.

env: The programming environment. It contains a.o. the Java development kit, Python, the Python virtual environment (**venv**), libraries and binaries.

< directories to create 9a > \equiv
`../modules` \diamond

Fragment defined by 9abcd, 60b.
 Fragment referenced in 65b.

< directories to create 9b > \equiv
`../bin` `../env/bin` \diamond

Fragment defined by 9abcd, 60b.
 Fragment referenced in 65b.

< directories to create 9c > \equiv
`../env/lib` \diamond

Fragment defined by 9abcd, 60b.
 Fragment referenced in 65b.

< directories to create 9d > \equiv
`../env/etc` \diamond

Fragment defined by 9abcd, 60b.
 Fragment referenced in 65b.

It would be great if an installed pipeline could be moved to another directory while it would keep working. We are not yet sure whether this is possible. However, a minimum condition for this to work would be, that the location of the pipeline can be determined at run-time. To achieve this, let us place a script in the root-directory of the pipeline, that can find in run-time the absolute path to itself and that generates variables that point to the other directories.

```

"../progenv" 10a≡
# Source this script
< get location of the script (10b piperoot ) 53a>
< set variables that point to the directory-structure 10e, ... >
< set environment parameters 10c, ... >
if
  [ -e "$piperoot/progenvv" ]
then
  source $piperoot/progenvv
fi
export progenvset=0
◇

```

Uses: piperoot 10d.

```

< set environment parameters 10c> ≡
export LC_ALL=en_US.UTF-8
export LANG=en_US.UTF-8
export LANGUAGE=en_US.UTF-8
◇

```

Fragment defined by 10c, 21b, 30b, 31a, 34d, 35c.

Fragment referenced in 10a.

The full path to the sourced script can be found in variable BASH_SOURCE[0].

```

< find the nlpp root directory 10d> ≡
piperoot="$( cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )"
◇

```

Fragment never referenced.

Defines: piperoot 10abe, 13e, 16ac, 20c, 29ab, 34b, 35a, 36b, 52a.

Once we know piperoot, we know the path to the other directories of figure 1.

```

< set variables that point to the directory-structure 10e> ≡
export pipesocket=${piperoot%*/nlpp}
export nuwebdir=$piperoot/nuweb
export envdir=$piperoot/env
export envbindir=$envdir/bin
export envlibdir=$envdir/lib
export modulesdir=$piperoot/modules
export pipebin=$piperoot/bin
export javadir=$envdir/java
export jarsdir=$javadir/jars
◇

```

Fragment defined by 10ef, 11a, 14f.

Fragment referenced in 10a.

Uses: nuweb 61d, piperoot 10d.

Include a “snapshot” directory that contains non-open materials.

```

< set variables that point to the directory-structure 10f> ≡
export snapshotdir=$pipesocket/v4.0.0.0_nlpp_resources
◇

```

Fragment defined by 10ef, 11a, 14f.

Fragment referenced in 10a.

Add the environment `bin` directory to `PATH`:

```
< set variables that point to the directory-structure 11a > ≡
    export PATH=$envbindir:$pipebin:$PATH
◇
```

Fragment defined by 10ef, 11a, 14f.

Fragment referenced in 10a.

Defines: `PATH` 13e, 14bf, 20c.

3.2 Download resources

To enhance speed of the installation we start to download all resources that we can download at the beginning of the installation in a single blow as parallel processes. We park the resources in a directory `v4.0.0.0_nlpp_resources`, located in the directory where the root of NLPP also resides.

```
< init make_infrastructure 11b > ≡
    < download everything 11c, ... >
    wait
◇
```

Fragment defined by 8e, 11b.

Fragment referenced in 7a, 8a.

Hopefully there will be little to download.

Synchronize with a non-open snapshot-directory if possible. It is only possible if a valid `ssh` key resides in file `nrkey` in the directory in which the `nlpp` root directory resides.

```
< download everything 11c > ≡
    mkdir -p $pipesocket/v4.0.0.0_nlpp_resources
    if
        [ -e /home/paul/temp/nrkey ]
    then
        cd $pipesocket
        ( rsync -e "ssh -i /home/paul/temp/nrkey" -
          rLt newsreader@kyoto.let.vu.nl:v4.0.0.0_nlpp_resources . ) &
    fi
◇
```

Fragment defined by 11c, 28a.

Fragment referenced in 11b.

Download other stuff using `wget`. The following macro downloads a resource into the snapshot-directory if it is not already there.

```
< need to wget 11d > ≡
    if
        [ ! -e $pipesocket/v4.0.0.0_nlpp_resources/@1 ]
    then
        cd $pipesocket/v4.0.0.0_nlpp_resources
        ( wget @2 ) &
    fi
◇
```

Fragment referenced in 13f, 18c, 28b, 33c.

3.3 Java

We need to have a Java JDK version 1.8 installed. In other words, when we issue the instruction `javac -version` within the pipeline environment, the response must be something like `javac 1.8.0_131`. We assume that if we find a correct Java 1.8, there will also be a proper `java`. Let us first test whether that is the case. If it is not the case, we can install java if a proper tarball is present in the “snapshot directory”.

Let us perform the two tests:

Do we have a proper Java?

```

< check presence of javac in 1.8 12a > ≡
    javac -version 2>&1 | grep 1.8 >/dev/null
    if
        [ $? == 0 ]
    then
        @1="True"
    else
        @1="False"
    fi
    ◇

```

Fragment referenced in 13a.

Do we have a tarball to install Java? (in fact, the following macro can be used to check the presence of any tarball in the snapshot directory).

```

< check whether a tarball is present in the snapshot 12b > ≡
    if
        [ -e $pipesocket/v4.0.0.0_nlpp_resources/@1 ]
    then
        @2="True"
    else
        @2="False"
    fi
    ◇

```

Fragment referenced in 13a, 14b, 15b, 19b.

Now do it:

```

< set up Java 13a > ≡
  < check presence of javac in 1.8 (13b java_OK ) 12a >
  if
    [ ! "$java_OK" == "True" ]
  then
    < check whether a tarball is present in the snapshot (13c jdk-8u131-linux-x64.tar.gz,13d tarball_present ) 12b >
    if
      [ ! "$tarball_present" == "True" ]
    then
      echo "Please install Java 1.8 JDK"
      exit 4
    fi
    mkdir -p $javadir
    cd $javadir
    tar -xzf $pipesocket/v4.0.0.0_nlpp_resources/jdk-8u131-linux-x64.tar.gz
    < set up java environment 13e >
  fi
  ◇

```

Fragment referenced in 7a.

Adapt the PATH variable and set JAVA_HOME. Set these variables in the script that will be sourced in the running pipeline and set them in this script because we are going to need Java.

```

< set up java environment 13e > ≡
  echo 'export JAVA_HOME=$envdir/java/jdk1.8.0_131' >> $piperoot/progenvv
  echo 'export PATH=$JAVA_HOME/bin:$PATH' >> $piperoot/progenvv
  export JAVA_HOME=$envdir/java/jdk1.8.0_131
  export PATH=$JAVA_HOME/bin:$PATH
  ◇

```

Fragment referenced in 13a.

Uses: PATH 11a, piperoot 10d.

3.4 Maven

Currently we need version 3.0.5 to compile the Java sources in some of the modules.

3.5 Maven

Some Java-based modules can best be compiled with [Maven](#). So download and install Maven:

```

< download stuff 13f > ≡
  < need to wget (13g apache-maven-3.0.5-bin.tar.gz,13h http://apache.rediris.es/maven/maven-3/3.0.5/binaries) 13f >
  ◇

```

Fragment defined by 13f, 18c, 28b, 33c.

Fragment referenced in 28a.

First check whether maven is already present in the correct version.

```

< check presence of maven in 3.0.5 14a > ≡
    mvn -version | grep "Maven 3.0.5" >/dev/null
    if
        [ $? == 0 ]
    then
        @1="True"
    else
        @1="False"
    fi
    ◇

```

Fragment referenced in 14b.

```

< set up Maven 14b > ≡
    < check presence of maven in 3.0.5 (14c mvn_OK ) 14a >
    if
        [ ! "$mvn_OK" == "True" ]
    then
        < check whether a tarball is present in the snapshot (14d apache-maven-3.0.5-bin.tar.gz, 14e tarball_present ) 14a >
        if
            [ ! "$tarball_present" == "True" ]
        then
            echo "Please install Maven version 3.0.5"
            exit 4
        fi
        cd $envdir
        tar -xzf /home/paul/temp/v4.0.0.0_nlpp_resources/apache-maven-3.0.5-bin.tar.gz
        export MAVEN_HOME=$envdir/apache-maven-3.0.5
        export PATH=${MAVEN_HOME}/bin:${PATH}
    fi
    ◇

```

Fragment referenced in 7a.

```

< set variables that point to the directory-structure 14f > ≡
    export MAVEN_HOME=$envdir/apache-maven-3.0.5
    export PATH=${MAVEN_HOME}/bin:${PATH}
    ◇

```

Fragment defined by 10ef, 11a, 14f.

Fragment referenced in 10a.

Uses: PATH 11a.

When the installation has been finished, we do not need maven anymore.

```

< clean up after installation 14g > ≡
    cd $envdir
    rm -rf apache-maven-3.0.5
    ◇

```

Fragment referenced in 8a.

3.6 Python

Several modules in the pipeline run on Python version 3.6. If the command `python` does not invoke that version, we can try install ActivePython, of which we have a tarball in the snapshot.

Versioning in Python is very confusing. It is the [official Python policy](#) that `/usr/bin/env python` points to Python version 2 but that scripts with a shabang of `#!/usr/bin/env python` should be executable by Python version 2 as well as Python version 3.

Our policy will be as follows:

1. When installing, make sure that command `python3` starts a python 3.6 executable. If this is not the case, install ActivePython version 3.6.
2. Generate a virtual environment.
3. Make sure that in our environmen command `python` executes python from the virtual environment.

```

< check presence of python3 in 3.6 15a > ≡
python3 --version 2>&1 | grep "Python 3.6" >/dev/null
if
[ $? == 0 ]
then
  @1="True"
else
  @1="False"
fi
◇

```

Fragment referenced in [15b](#).

```

< set up Python 15b > ≡
< check presence of python3 in 3.6 (15c python_OK ) 15a >
if
[ ! "$python_OK" == "True" ]
then
  < check whether a tarball is present in the snapshot (15d ActivePython-3.6.0.3600-linux-x86_64-glibc-2.3.6-40)
  if
  [ ! "$tarball_present" == "True" ]
  then
    echo "Please install Python version 3.6"
    exit 4
  fi
  < install ActivePython 16a >
fi
◇

```

Fragment defined by [15b](#), [18ab](#).

Fragment referenced in [7a](#).

Unpack the tarball in a temporary directory and install active python in the `env` subdirectory of `nlpp`. Activepython has a few peculiarities:

- It installs things in subdirectories `bin` and `lib` of the installation-directory (in our case subdirectory `env`).
- It installs scripts with names `python3` and `pip3`. We will make symbolic links from these scripts to `python` resp. `pip`.
- It writes self-starting scripts with a “shabang” containing the full absolute path to the `python3` script. In an attempt to make Active-python relocatable we will rewrite the Shabangs to have them contain `#!/usr/bin/env python`.

```

< install ActivePython 16a > ≡
    pytinsdir='mktemp -d -t activepyt.XXXXXX'
    cd $pytinsdir
    tar -xzf $pipesocket/v4.0.0.0_nlpp_resources/ActivePython-3.6.0.3600-linux-x86_64-
    glibc-2.3.6-401834.tar.gz
    accdir='ls -l'
    cd $acdir
    ./install.sh -I $envdir
    cd $piperoot
    rm -rf $pytinsdir
    < create python script and pip script 16b >
    < rewrite ActivePython shabangs 16c >

```

◇

Fragment referenced in 15b.

Uses: install 66a, piperoot 10d.

```

< create python script and pip script 16b > ≡
    cd $envbindir
    rm python
    ln -s python3 python
    rm pip
    ln -s pip3 pip

```

◇

Fragment referenced in 16a.

To rewrite the shabangs of the ActivePython scripts do as follows:

1. Create a temporary directory.
2. Generate an AWK script that replaces the shabang line with a correct one.
3. Generate a script that moves a script from `env/bin` to the temporary directory and then applies the AWK script.
4. Apply the generated script on the scripts in `env/bin`.

```

< rewrite ActivePython shabangs 16c > ≡
    transfile='mktemp -t trans.XXXXXX'
    rm -rf $transfile
    < apply script tran on the scripts in (16d $envbindir, 16e $transfile ) 17c >
    cd $piperoot
    rm -rf $transfile

```

◇

Fragment referenced in 16a.

```

"../env/bin/tran" 16f≡
    #!/bin/bash
    < get location of the script (16g trandir ) 53a >
    workfil=$1
    tempfil=$2
    mv $workfil $tempfil
    gawk -f $strandir/chasbang.awk $tempfil>$workfil
    chmod 775 $workfil

```

◇


```

< make scripts executable 17a > ≡
    chmod 775 ../env/bin/tran
    ◇

```

Fragment defined by 7o, 8c, 17a, 24f, 36a, 47e, 65c.
 Fragment referenced in 65d.

```

"../env/bin/chasbang.awk" 17b ≡
    #!/usr/bin/gawk -f
    BEGIN { shabang="#!/usr/bin/env python3"}

    /\^#\!.*python.*/ { print shabang
                        next
                      }

    {print}
    ◇

```

Uses: `print` 59b.

The following looks complicated. The `find` command applies the `file` command on the files in the `env/bin` directory. The `grep` command filters out the names of the files that are scripts. it produces a filename, followed by a colon, followed by a description of the type of the file. The `gawk` command prints the filenames only and the `xargs` command applies the `tran` script on the file.

```

< apply script tran on the scripts in 17c > ≡
    find @1 -type f -exec file {} + \
    | grep "Python script" | gawk '{print $1}' FS=':' \
    | xargs -iaap $envbindir/tran aap @2
    ◇

```

Fragment referenced in 16c.
 Uses: `print` 59b.

3.6.1 Python packages

In order to be reproducible, we must make sure that Python packages are installed in the correct version. Therefore, we will install the packages beforehand and do not leave that to the install-scripts of the modules. Descriptions of the packages can be found on <https://pypi.python.org>. Install the following packages:

package	version	module
KafNafParserPy	1.87	
lxml	3.8.0	
pyyaml	3.12	
requests	2.18.1	networkx
networkx	1.11	corefbase

```

⟨ set up Python 18a ⟩ ≡
    pip install KafNafParserPy==1.87
    pip install lxml==3.8.0
    pip install networkx==1.11
    pip install pyyaml==3.12
    pip install requests==2.18.1
    pip install six==1.10.0.
    pip install pynaf
    ◇

```

Fragment defined by 15b, 18ab.

Fragment referenced in 7a.

Uses: install 66a.

3.6.2 Pynaf hack

I could not get pynaf to work properly for EHU-corefgraph.v30. Therefore, I made a hack to make it work.

```

⟨ set up Python 18b ⟩ ≡
    cd $envlibdir/python3.6/site-packages/pynaf
    mv __init__.py old.__init__.py
    cp $snapshotdir/pynafhack ./__init__.py
    ◇

```

Fragment defined by 15b, 18ab.

Fragment referenced in 7a.

3.7 Perl

One of the modules uses perl and needs XML::LibXML. However, installation of that package seems to be tricky and seems to depend on the availability of obscure stuff. So, we proceed as follows. First test whether Perl version 5 is present on the host. If that is not the case, check whether we have a tarball named 20160520_nlpp_perllib.tgz in the snapshot. If that is the case, install Perl from scratch and unpack the tarball. Otherwise, fail, and tell the user to install Perl and XML::LibXML.

Install Perl locally, to be certain that Perl is available and to enable to install packages that we need (in any case: XML::LibXML).

```

⟨ download stuff 18c ⟩ ≡

    ⟨ need to wget (18d perl-5.22.1.tar.gz, 18e http://www.cpan.org/src/5.0/perl-5.22.1.tar.gz ) 11d ⟩

    ◇

```

Fragment defined by 13f, 18c, 28b, 33c.

Fragment referenced in 28a.

```

< check presence of perl in 5 19a > ≡
perl -v 2>&1 | grep "perl 5," >/dev/null
if
  [ $? == 0 ]
then
  @1="True"
else
  @1="False"
fi
◇

```

Fragment referenced in 19b.

```

< set up Perl 19b > ≡
< check presence of perl in 5 (19c perl_OK ) 19a >
if
  [ "$perl_OK" == "True" ]
then
  < check whether XML::LibXML is installed (19d lib_OK ) 19g >
  if
    [ ! "$lib_OK" == "True" ]
  then
    perl_OK="False"
  fi
fi
if
  [ ! "$perl_OK" == "True" ]
then
  < check whether a tarball is present in the snapshot (19e 20160520_nlpp_perllib.tgz, 19f tarball_present ) 12b >
  if
    [ ! "$tarball_present" == "True" ]
  then
    echo "Please install Perl version 3.6 and XML::LXML"
    exit 4
  fi
  < install perl 20c, ... >
fi
◇

```

Fragment never referenced.

```

< check whether XML::LibXML is installed 19g > ≡
perl -MXML::LibXML -e 1 2>/dev/null
if
  [ $? == 0 ]
then
  @1="True"
else
  @1="False"
fi
◇

```

Fragment referenced in 19b.

Install Perl if either Perl is not installed or the installed Perl has a version below 5.

```

< conditionally install Perl 20a > ≡
installPerl=1
which perl >/dev/null
res=$?
if
[ $res -gt 0 ]
then
installPerl=0
else
perlversion='perl -e 'print $];''
if
[ "${perlversion%%.*}" -lt 5 ]
then
installPerl=0
fi
fi
if
[ installPerl -gt 0 ]
then
< install Perl 20b >
fi
◇

```

Fragment referenced in 7a.

Uses: `print` 59b.

```

< install Perl 20b > ≡
tmpdir='mktemp -d -t perl.XXXXXX'
cd $tmpdir
tar -xzf $pipesocket/v4.0.0.0_nlpp_resources/perl-5.22.1.tar.gz
cd perl-5.22.1
./Configure -des -Dprefix=$envdir/perl
make
make test
make install
cd $progroot
rm -rf $tmpdir
◇

```

Fragment referenced in 20a.

Uses: `install` 66a.

Make sure that modules use the correct Perl

```

< install perl 20c > ≡
echo 'export PERL_HOME=$envdir/perl' >> $piperoot/progenvv
echo 'export PATH=$PERL_HOME/bin:$PATH' >> $piperoot/progenvv
export PERL_HOME=$envdir/perl
export PATH=$PERL_HOME/bin:$PATH
◇

```

Fragment defined by 20c, 21a.

Fragment referenced in 19b.

Uses: `PATH` 11a, `piperoot` 10d.

Unpack the poor-man tarball with LibXML:

```

⟨ install perl 21a ⟩ ≡
    cd $envdir/perl/lib
    tar -xzf $pipesocket/v4.0.0.0_nlpp_resources/20160520_nlpp_perllib.tgz
    ◇

```

Fragment defined by 20c, 21a.

Fragment referenced in 19b.

3.8 Spotlight

A Spotlight server occupies a lot of memory and we need two of them, one for each language. We may be lucky and have a spotlight server running somewhere. Nevertheless, let us be prepared to be able to install a server ourselves.

3.8.1 Set spotlight host and port

Maybe we do not have to use the built-in spotlight. Tell the modules about this:

```

⟨ set environment parameters 21b ⟩ ≡
    export SPOTLIGHTHOST=localhost
    ◇

```

Fragment defined by 10c, 21b, 30b, 31a, 34d, 35c.

Fragment referenced in 10a.

3.8.2 Install spotlight servers

Install Spotlight in the way that Itziar Aldabe (<mailto:itziar.aldabe@ehu.es>) described:

The NED module works for English, Spanish, Dutch and Italian. The module returns multiple candidates and correspondences for all the languages. If you want to integrate it in your Dutch or Italian pipeline, you will need:

1. The jar file with the dbpedia-spotlight server. You need the version that Aitor developed in order to correctly use the "candidates" option. You can copy it from the English VM. The jar file name is `dbpedia-spotlight-0.7-jar-with-dependencies-candidates.jar`
2. The Dutch/Italian model for the dbpedia-spotlight. You can download them from: <http://spotlight.sztaki.hu/downloads/>
3. The jar file with the NED module: `ixa-pipe-ned-1.0.jar`. You can copy it from the English VM too.
4. The file: `wikipedia-db.v1.tar.gz`. You can download it from: <http://ixa2.si.ehu.es/ixa-pipes/models/wikipedia-db.v1.tar.gz>. This file contains the required information to do the mappings between the wikipedia-entries. The zip file contains three files: `wikipedia-db`, `wikipedia-db.p` and `wikipedia-db.t`

To start the dbpedia server: Italian server:

```
java -jar -Xmx8g dbpedia-spotlight-0.7-jar-with-dependencies-candidates.jar \
    it http://localhost:2050/rest
```

Dutch server:

```
java -jar -Xmx8g dbpedia-spotlight-0.7-jar-with-dependencies-candidates.jar nl http://localhost:2
```

We set 8Gb for the English server, but the Italian and Dutch Spotlight will require less memory.

So, that's how we did it. However, currently the url `http://spotlight.sztaki.hu/downloads/` seems to be unreachable. Therefore, as a temporary measure, we just unpack a complete spotlight installation.

First, get the Spotlight model data that we need:

Unpack the spotlight ball as a subdirectory of `env`

```
< install the Spotlight server 22a > ≡
    cd $envdir
    tar -xzf $pipesocket/$snapshotdirectory/20180301_spotlight.tgz
    ◇
```

Fragment referenced in [7a](#).

3.8.3 Check/start the Spotlight server

The macro `check/start spotlight` does the following:

1. Check whether spotlight runs on the default spotlighthost.
2. If that is not the case, and the defaulthost is not `localhost`, check whether Spotlight runs on `localhost`.
3. If a running spotlightserver is still not found, start a spotlightserver on `localhost`.

Start Spotlight, if it doesn't run already. Spotlight ought to run on `localhost` unless variable `spotlighthost` exists. In that case, check whether a Spotlight server can be contacted on that host. Otherwise, change `spotlighthost` to `localhost` and check whether a Spotlight server runs there. If that is not the case, start up a Spotlight server on `localhost`.

The following script, `check_start_spotlight`, has three optional arguments:

language: Default is exported variable `naflang` if it exists, or `en`.

spotlighthost: Name of a host that probably runs a Spotlightserver. Default: exported variable `spotlighthost` if it exists, or `localhost`.

spotlightport: Default: exported variable `spotlightport` if it exists or either 2020 or 2060 for English resp. Dutch.

```
"../bin/check_start_spotlight" 22b≡
    #!/bin/bash
    < get location of the script (22c DIR ) 53a >
    cd $DIR
    source ../progenv
    < get cmdline-arguments for check_start_spotlight 23a >
    < set default arguments for Spotlight 23b >
    ◇
```

File defined by [22b](#), [24a](#).

The code to obtain command-line arguments has been obtained from [Stackoverflow](#). The following fragment reads the arguments `-l language`, `-h spotlighthost` and `-p spotlightport`:

```

⟨ get commandline-arguments for check_start_spotlight 23a ⟩ ≡
    while [[ $# > 1 ]]
    do
        key="$1"

        case $key in
            -l|--language)
                naflang="$2"
                shift # past argument
                ;;
            -h|--spothost)
                spotlighthost="$2"
                shift # past argument
                ;;
            -p|--spotport)
                spotlightport="$2"
                shift # past argument
                ;;
            *)
                # unknown option
                ;;
        esac
        shift # past argument or value
    done
    ◇

```

Fragment referenced in 22b.

Uses: `naflang` 50b.

Fill in default values when they cannot be found in exported variables nor in command-line arguments.

```

⟨ set default arguments for Spotlight 23b ⟩ ≡
    if
        [ "$spotlighthost" == "" ]
    then
        spotlighthost=localhost
    fi
    if
        [ "$spotlightport" == "" ]
    then
        if
            [ "$naflang" == "nl" ]
        then
            spotlightport=2060
        else
            spotlightport=2020
        fi
    fi
    ◇

```

Fragment referenced in 22b.

Uses: `naflang` 50b.

```

"../bin/check_start_spotlight" 24a≡
  < check listener on host, port (24b $spotlighthost,24c $spotlightport ) 25b>
  if
    [ $spotlightrunning -ne 0 ]
  then
    if
      [ ! "$spotlighthost" == "localhost" ]
    then
      export spotlighthost="localhost"
      < check listener on host, port (24d $spotlighthost,24e $spotlightport ) 25b>
    fi
  fi
  if
    [ $spotlightrunning -ne 0 ]
  then
    < start the Spotlight server on localhost 26b, ... >
  fi
  echo $spotlighthost:$spotlightport
  ◇

```

File defined by 22b, 24a.

```

< make scripts executable 24f> ≡
  chmod 775 ../bin/check_start_spotlight
  ◇

```

Fragment defined by 7o, 8c, 17a, 24f, 36a, 47e, 65c.

Fragment referenced in 65d.

Use function `check_start_spotlight` to find and exploit a running Spotlight-server or to die (with exit code 5) if no server can be found or created. The macro uses implicitly the exported variables `spotlighthost` and `spotlightport` if they exist.

```

< find a spotlightserver or exit 24g> ≡
  spothostport='/home/paul/temp/nlpp/bin/check_start_spotlight -l $naflang'
  export spotlighthost='echo $spothisport | gawk -F ":" '{print $1}''
  export spotlightport='echo $spothisport | gawk -F ":" '{print $2}''
  echo "Spotlight server found on $spothisport." >&2
  if
    [ "$spotlighthost" == "none" ]
  then
    echo "No Spotlight-server found."
    exit 5
  fi
  ◇

```

Fragment referenced in 52a.

Uses: `naflang` 50b, `print` 59b.

Set the port-number and the language resource for Spotlight, dependent of the language that the user gave as argument.


```

< get spotlight language parameters 25a > ≡
    if
        [ "$naflang" == "nl" ]
    then
        spotlightport=2060
    else
        spotlightport=2020
    fi
◇

```

Fragment never referenced.

Uses: **naflang** 50b.

The following macro has a hostname and a port-number as arguments. It checks whether something in the host listens on the port and sets variable **success** accordingly:

```

< check listener on host, port 25b > ≡
    exec 6<>/dev/tcp/@1/@2 2>/dev/null
    spotlightrunning=$?
    exec 6<&-
    exec 6>&-
◇

```

Fragment referenced in 24a, 27b.

If variable **spotlighthost** does not exist, set it to localhost. Test whether a Spotlightserver runs on **spotlighthost**. If that fails and **spotlighthost** did not point to localhost, try localhost.

If the previous attempts were not succesfull, start the spotlightserver on localhost.

If some spotlightserver has been contacted, set variable **spotlightrunning**. Otherwise exit. At the end variable **spotlighthost** ought to contain the address of the Spotlight-host.

```

< try to obtain a running spotlightserver 25c > ≡
    < test whether spotlighthost runs (25d $spotlighthost ) 26a >
    if
        [ ! $spotlightrunning ]
    then
        if
            [ "$spotlighthost" != "localhost" ]
        then
            export spotlighthost=localhost
            < test whether spotlighthost runs (25e $spotlighthost ) 26a >
        fi
    fi
    if
        [ ! $spotlightrunning ]
    then
        < start the Spotlight server on localhost 26b, ... >
        < test whether spotlighthost runs (25f $spotlighthost ) 26a >
    fi
    if
        [ ! $spotlightrunning ]
    then
        echo "Cannot start spotlight"
        exit 4
    fi
◇

```

Fragment never referenced.

Test whether the Spotlightserver runs on a given host. The “spotlight-test” does not really test Spotlight, but it tests whether something is listening on the port and host where we expect Spotlight. I found the test-construction that is used here on [Stackoverflow](#). If the test is positive, set variable `spotlightrunning` to 0. Otherwise, unset that variable.

```

< test whether spotlighthost runs 26a > ≡
    exec 6<>/dev/tcp/@1/2060
    if
        [ $? -eq 0 ]
    then
        export spotlightrunning=0
    else
        spotlightrunning=
    fi
    exec 6<&-
    exec 6>&-
    ◇

```

Fragment referenced in [25c](#).

When trying to start the Spotlight-server on localhost, take care that only one process does this. So we do this:

1. Try to acquire a lock without waiting for it.
2. If we got the lock, run the Spotlight java program in background.
3. If we got the lock, release it.
4. If we did not get the lock, wait for the lock to be released by the process that started the spotlight-server.

But first, we specify the resources for the Spotlight-server.

```

< start the Spotlight server on localhost 26b > ≡
    if
        [ "$naflang" == "nl" ]
    then
        spotresource="nl"
    else
        spotresource="en_2+2"
    fi
    spotlightjar=dbpedia-spotlight-0.7-jar-with-dependencies-candidates.jar
    ◇

```

Fragment defined by [26b](#), [27a](#).

Fragment referenced in [24a](#), [25c](#).

Uses: `naflang` [50b](#).

```

< start the Spotlight server on localhost 27a > ≡
local oldd='pwd'
cd /home/paul/temp/nlpp/env/spotlight
$envbindir/sematree acquire spotlock 0
gotit=$?
if
[ $gotit == 0 ]
then
java -jar -Xmx8g $spotlightjar $spotresource \
    http://localhost:$spotlightport/rest &
    < wait until the spotlight server is up or faulty 27b >
    $envbindir/sematree release spotlock
else
    < wait until the spotlight server is up or faulty 27b >
fi
cd $oldd
◇

```

Fragment defined by 26b, 27a.

Fragment referenced in 24a, 25c.

When the Spotlight server has been started, it takes up to a minute until it really listens on its port. When there is something wrong, it will never listen, of course. Therefore, we give it three minutes. If after that time still nothing listens, we set `spotlighthost` to `none`, indicating that something has gone wrong.

```

< wait until the spotlight server is up or faulty 27b > ≡
trial=0
maxtrials=12
while
    trial=$((trial+1))
    < check listener on host, port (27c $spotlighthost, 27d $spotlightport ) 25b >
    [ $spotlightrunning -ne 0 ] && [ $trial -le $maxtrials ]
do
    sleep 10
done
if
[ $spotlightrunning -ne 0 ]
then
    export spotlighthost="none"
fi
◇

```

Fragment referenced in 27a.

Start the Spotlight if it is not already running. First find out what the host is on which we may expect to find a listening Spotlight.

Variable `spotlighthost` contains the address of the host where we expect to find Spotlight. If the expectation does not come true, and the Spotlighthost was not localhost, test whether Spotlight can be found on localhost. If the spotlight-server cannot be found, start it up on localhost.

3.9 Download materials

This installer needs to download a lot from different sources:

- Most of the NLP-modules will be built up from their sources in Github. The sources must be cloned.

- Many modules need external resources, e.g. the Alpino tagger. Often these utilities must be downloaded from a location specified by the supplier.
- Many modules use extra resources like model-data, that must be obtained separately.
- Some of the resources are not publicly available. They must be obtained from a pass-word protected URL.
-

Usually downloads are slow, and the duration is only little determined by the resources in the installing computer, but by the network and the performance of the systems from which we download. Therefore, we may speed up by first downloading things, if possible in parallel processes.

We put the following the beginning of the install-script:

```
< download everything 28a > ≡
  < download stuff 13f, ... >
  echo Waiting for downloads to complete ...
  wait
  echo Download completed
  ◇
```

Fragment defined by 11c, 28a.

Fragment referenced in 11b.

4 Shared libraries

When we do not want to rely on what the host can present to us, we need to make our own shared libraries. For the present, we will generate the shared libraries `libxslt` and `libxml2`. We do the following:

1. install autoconf, needed to compile the libs.
2. install libxslt
3. install libxml2

4.1 Autoconf

Gnu autoconf is a system to help configure the Makefiles for a software package. Software packages that use this, supply a file `configure`, `configure.in` or `configure.ac`. To compile and install a package from source we can then perform 1) `./configure --prefix=<environment>`; 2) `make`; 3) `make install`.

Get autoconf:

```
< download stuff 28b > ≡
  < need to wget (28c autoconf-2.69.tar.gz, 28d http://ftp.gnu.org/gnu/autoconf/autoconf-2.69.tar.gz ) 11d >
  ◇
```

Fragment defined by 13f, 18c, 28b, 33c.

Fragment referenced in 28a.

Install autoconf:

< set up autoconf 29a > ≡

```
autoconfdir='mktemp -d -t autoconf.XXXXXX'
cd $autoconfdir
tar -xzf $pipesocket/v4.0.0.0_nlpp_resources/autoconf-2.69.tar.gz
cd autoconf-2.69
./configure --prefix=$envdir
make
make install
cd $piperoot
rm -rf $autoconfdir
◇
```

Fragment referenced in 7a.

Uses: install 66a, piperoot 10d.

4.2 libxml2 and libxslt

Compilation and installation of libxml2 and libxslt goes similar, according to the following template:

< install libxml2 or libxslt 29b > ≡

```
shtmpdir='mktemp -d -t shl.XXXXXX'
cd $shtmpdir
git clone @1
packagedir='ls -1'
cd $packagedir
./autogen.sh --prefix=$envdir
make
make install
cd $piperoot
rm -rf $shtmpdir
◇
```

Fragment referenced in 29c.

Uses: install 66a, piperoot 10d.

< install shared libs 29c > ≡

```
< install libxml2 or libxslt (29d git://git.gnome.org/libxml2 ) 29b >
< install libxml2 or libxslt (29e git://git.gnome.org/libxslt ) 29b >
◇
```

Fragment referenced in 7a.

4.3 Alpino

Install Alpino as a utility because it is so big, and hard to install on different platforms. Users may choose to install the utilities (and Alpino) by hand and then still install the modules with the script from this file.

Alpino cannot be obtained from an open source repository and there does not seem to be a repository where all the older versions are stored. Therefore, if possible, we will use a copy from

our secret archive if that is available. If that is not available, we will download the latest version of Alpino.

```

< install Alpino 30a > ≡
  alpinosrc=Alpino-x86_64-Linux-glibc-2.19-21088-sicstus.tar.gz
  cd $envdir
  if
  [ -d "Alpino" ]
  then
    echo "Not installing Alpino, because of existing directory $envdir/Alpino"
  else
    if
      [ ! -e "$pipesocket/v4.0.0.0_nlpp_resources/$alpinosrc" ]
    then
      echo "Try to install the latest Alpino."
      alpinosrc=latest.tar.gz
      cd $pipesocket/v4.0.0.0_nlpp_resources
      wget http://www.let.rug.nl/vannoord/alp/Alpino/versions/binary/latest.tar.gz
      if
        [ $? -gt 0 ]
      then
        echo "Cannot install Alpino. Please install Alpino in $envdir/Alpino"
        exit 4
      fi
    fi
    cd $envdir
    tar -xzf $pipesocket/v4.0.0.0_nlpp_resources/$alpinosrc
  fi
  ◇

```

Fragment referenced in 7a.

Uses: install 66a.

```

< set environment parameters 30b > ≡
  export ALPINO_HOME=$envdir/Alpino
  ◇

```

Fragment defined by 10c, 21b, 30b, 31a, 34d, 35c.

Fragment referenced in 10a.

Defines: ALPINO_HOME Never used.

4.4 Treetagger

Installation of Treetagger goes as follows (See [Treetagger's homepage](#)):

1. Download and unpack the Treetagger tarball. This generates the subdirectories `bin`, `cmd` and `doc`
2. Download and unpack the tagger-scripts tarball

The location where Treetagger comes from and the location where it is going to reside:

```

< install the treetagger utility 30c > ≡
  TREETAGDIR=treetagger
  TREETAGGER_HOME=$envdir/$TREETAGDIR
  TREETAG_BASIS_URL=http://www.cis.uni-muenchen.de/%7Eschmid/tools/TreeTagger/data/
  ◇

```

Fragment defined by 30c, 31bcde, 32a, 33ab.

Fragment referenced in 7a.

Defines: TREETAGGER_HOME 31a, 32ce.

```

⟨ set environment parameters 31a ⟩ ≡
    export TREETAGGER_HOME=$envdir/treetagger
    ◇

```

Fragment defined by 10c, 21b, 30b, 31a, 34d, 35c.

Fragment referenced in 10a.

Uses: TREETAGGER_HOME 30c.

The source tarball, scripts and the installation-script:

```

⟨ install the treetagger utility 31b ⟩ ≡
    TREETAGSRC=tree-tagger-linux-3.2.1.tar.gz
    TREETAGSCRIPTS=tagger-scripts.tar.gz
    TREETAG_INSTALLSCRIPT=install-tagger.sh
    ◇

```

Fragment defined by 30c, 31bcde, 32a, 33ab.

Fragment referenced in 7a.

Uses: install 66a.

Parametersets:

```

⟨ install the treetagger utility 31c ⟩ ≡
    DUTCHPARS_UTF_GZ=dutch-par-linux-3.2-utf8.bin.gz
    DUTCH_TAGSET=dutch-tagset.txt
    DUTCHPARS_2_GZ=dutch2-par-linux-3.2-utf8.bin.gz
    ◇

```

Fragment defined by 30c, 31bcde, 32a, 33ab.

Fragment referenced in 7a.

Download everything in the target directory:

```

⟨ install the treetagger utility 31d ⟩ ≡
    mkdir -p $envdir/$TREETAGDIR
    cd $envdir/$TREETAGDIR
    wget $TREETAG_BASIS_URL/$TREETAGSRC
    wget $TREETAG_BASIS_URL/$TREETAGSCRIPTS
    wget $TREETAG_BASIS_URL/$TREETAG_INSTALLSCRIPT
    wget $TREETAG_BASIS_URL/$DUTCHPARS_UTF_GZ
    wget $TREETAG_BASIS_URL/$DUTCH_TAGSET
    wget $TREETAG_BASIS_URL/$DUTCHPARS_2_GZ
    ◇

```

Fragment defined by 30c, 31bcde, 32a, 33ab.

Fragment referenced in 7a.

Run the install-script:

```

⟨ install the treetagger utility 31e ⟩ ≡
    chmod 775 $TREETAG_INSTALLSCRIPT
    ./$TREETAG_INSTALLSCRIPT
    ◇

```

Fragment defined by 30c, 31bcde, 32a, 33ab.

Fragment referenced in 7a.

The scripts in the cmd subdirectory contain absolute paths. We can make the treetagger directory-structure location-independent by using relative paths, eg relative to TREETAGGER_HOME

```

<install the treetagger utility 32a> ≡
  <make treetagger location-independent 32b>
  ◇

```

Fragment defined by 30c, 31bcde, 32a, 33ab.

Fragment referenced in 7a.

It works as follows:

Many of the scripts in the `cmd` subdirectory contain lines like:

```
BIN=<absolute path>/bin
```

We read one of those scripts and extract the contents of `<absolute path>` into variable `indicator`. Then we replace in all scripts occurrences of this text with `${TREETAGGER_HOME}`.

```

<make treetagger location-independent 32b> ≡
  <extract the absolute path from one of the scripts 32c>
  <replace the absolute paths 32e>
  ◇

```

Fragment referenced in 32a.

```

<extract the absolute path from one of the scripts 32c> ≡
  cmdir=${TREETAGGER_HOME}/cmd
  probefile='grep -l "^BIN=" ${cmdir}/* | head -n 1'
  indicator='cat $probefile | gawk '/^BIN=/ {<matchscript 32d>}' '
  ◇

```

Fragment referenced in 32b.

Uses: `TREETAGGER_HOME` 30c.

```

<matchscript 32d> ≡
  match($0, /^BIN=(.*treetagger)\.bin/, arr); print arr[1]◇

```

Fragment referenced in 32c.

Uses: `print` 59b.

```

<replace the absolute paths 32e> ≡
  sedcommand="s|$indicator|${TREETAGGER_HOME}|g"
  tempfile='mktemp -t mytemp.XXXXXX'
  for file in ${cmdir}/*
  do
    mv $file $tempfile
    cat $tempfile | sed $sedcommand >$file
  done
  rm -rf $tempfile
  ◇

```

Fragment referenced in 32b.

Uses: `TREETAGGER_HOME` 30c.

Make the treetagger utilities available for everybody.


```

< install the treetagger utility 33a > ≡
    chmod -R o+rx $envdir/$TREETAGDIR/bin
    chmod -R o+rx $envdir/$TREETAGDIR/cmd
    chmod -R o+r $envdir/$TREETAGDIR/doc
    chmod -R o+rx $envdir/$TREETAGDIR/lib
    ◇

```

Fragment defined by 30c, 31bcde, 32a, 33ab.

Fragment referenced in 7a.

Remove the tarballs:

```

< install the treetagger utility 33b > ≡
    rm $TREETAGSRC
    rm $TREETAGSCRIPTS
    rm $TREETAG_INSTALLSCRIPT
    rm $DUTCHPARS_UTF_GZ
    rm $DUTCH_TAGSET
    rm $DUTCHPARS_2_GZ
    ◇

```

Fragment defined by 30c, 31bcde, 32a, 33ab.

Fragment referenced in 7a.

4.5 Timbl and Ticcutils

Timbl and Ticcutils are installed from their source-tarballs. The installation is not (yet?) completely reproducibe because it uses the C-compiler that happens to be available on the host. Installation involves:

1. Download the tarball in a temporary directory.
2. Unpack the tarball.
3. cd to the unpacked directory and perform `./configure`, `make` and `make install`. Note the argument that causes the files to be installed in the `lib` and the `bin` sub-directories of the `env` directory.

```

< download stuff 33c > ≡
    < need to wget (33d timbl-6.4.6.tar.gz,33e http://software.ticc.uvt.nl/timbl-6.4.6.tar.gz ) 11d >
    < need to wget (33f ticcutils-0.7.tar.gz,33g http://software.ticc.uvt.nl/ticcutils-0.7.tar.gz ) 11d >
    ◇

```

Fragment defined by 13f, 18c, 28b, 33c.

Fragment referenced in 28a.

```

< install the ticcutils utility 33h > ≡
    URL=http://software.ticc.uvt.nl/ticcutils-0.7.tar.gz
    TARB=ticcutils-0.7.tar.gz
    DIR=ticcutils-0.7
    < unpack ticcutils or timbl 34b >
    ◇

```

Fragment referenced in 7a, 34c.

```

< install the timbl utility 34a > ≡
    TARB=timbl-6.4.6.tar.gz
    DIR=timbl-6.4.6
    < unpack ticcutils or timbl 34b >
    ◇

```

Fragment referenced in 7a, 34c.

```

< unpack ticcutils or timbl 34b > ≡
    SUCCES=0
    ticbeldir='mktemp -t -d tickbel.XXXXXX'
    cd $ticbeldir
    tar -xzf $pipesocket/v4.0.0.0_nlpp_resources/$TARB
    cd $DIR
    sh ./bootstrap.sh
    ./configure --prefix=$envdir
    make
    make install
    cd $piperoot
    rm -rf $ticbeldir
    ◇

```

Fragment referenced in 33h, 34a.

Uses: install 66a, piperoot 10d.

```

< re-install modules after the transplantation 34c > ≡
    < install the ticcutils utility 33h >
    < install the timbl utility 34a >
    ◇

```

Fragment never referenced.

4.6 Svmlib

Svmlib is needed by module svmwsd. That module can install svmlib by itself, but for now we try installation in the prog-environment. We set variable `SVMLIB_HOME` to indicate where the module is located.

```

< set environment parameters 34d > ≡
    export SVMLIB_HOME=$envdir/svmlib
    ◇

```

Fragment defined by 10c, 21b, 30b, 31a, 34d, 35c.

Fragment referenced in 10a.

Defines: `SVMLIB_HOME` 35a.

```

< install svmllib 35a > ≡
    export SVMLIB_HOME=${envdir}/svmllib
    tempdir='mktemp -d -t svmllib.XXXXXX'
    cd $tempdir
    wget https://github.com/cjlin1/libsvm/archive/master.zip
    unzip master.zip
    rm master.zip
    oridir='ls -1 | head -1'
    mv $oridir ${SVMLIB_HOME}
    cd ${SVMLIB_HOME}/python
    rm -rf $tempdir
    make
    cd $piperoot
    ◇

```

Fragment referenced in 7a.

Uses: piperoot 10d, SVMLIB_HOME 34d.

4.7 The Boost library

I have no idea how Boost works. Neither can I find out how to test whether boost has been installed already. So we install libboost according to [this manual](#) and hope for the best.

```

< install boost 35b > ≡
    cd $envdir
    tar -xzf $pipesocket/v4.0.0.0_nlpp_resources/20160103_boost_1_54_bin.tgz
    ◇

```

Fragment referenced in 7a.

Zet de boost libraries in LD_LIBRARY_PATH.

```

< set environment parameters 35c > ≡
    export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${envdir}/boost_1_54_0/stage/lib
    ◇

```

Fragment defined by 10c, 21b, 30b, 31a, 34d, 35c.

Fragment referenced in 10a.

5 Install the modules

We make a separate script to install the modules. By default, the modules will be installed in subdirectory `modules` of the NLPP root directory, but this is not necessarily so.

The script `install_modules` installs modules that are not yet present.

```

"../env/bin/install_modules" 35d≡
    #!/bin/bash
    < get location of the script (35e DIR ) 53a >
    cd $DIR
    source ../../progenv
    < variables of the module-installer 53b >
    < functions of the module-installer 36b, ... >
    < install the modules 38c, ... >
    ◇

```

```

<make scripts executable 36a> ≡
    chmod 775 ../env/bin/install_modules
    ◇

```

Fragment defined by 7o, 8c, 17a, 24f, 36a, 47e, 65c.

Fragment referenced in 65d.

Installing a module from Github is very simple:

- Skip installation if the module is already present. Otherwise:
- Clone the module in subdirectory `modules`.
- `cd` to that module and perform script `install`.

```

<functions of the module-installer 36b> ≡
    function gitinst (){
        url=$1
        dir=$2
        commitset=$3
        echo "Install $dir" >&2
        cd $piperoot/modules
        if
            [ -e $dir ]
        then
            echo "Not installing existing module $dir"
        else
            git clone $url
            cd $dir
            git checkout $commitset
            ./install
        fi
    }
    ◇

```

Fragment defined by 36bc.

Fragment referenced in 35d.

Uses: `install` 66a, `piperoot` 10d.

```

<functions of the module-installer 36c> ≡
    function ballinst (){
        ball=$1
        if
            [ -e $snapshotdir/$ball ]
        then
            cd $modulesdir
            tar -xzf $snapshotdir/$ball
        else
            echo "Cannot find $snapshotdir/$ball"
            exit 1
        fi
    }
    ◇

```

Fragment defined by 36bc.

Fragment referenced in 35d.

For each module we generate a script in the `bin` subdirectory to make the module easier to use. The script does the following:

1. Find the directory of itself.

2. Run script `run` in the directory of the module, that can be found as `../<modulename>/run`.

```

< contents of shorthand-script 37a > ≡
#!/bin/bash
< get location of the script (37b thisdir ) 53a >
scriptname=${0##*/}
scriptpath=$thisdir/$scriptname
cd ${thisdir}
source ../progenv
< set the naflang parameter 37c >
cat | ../modules/@1/run
◇

```

Fragment referenced in 38dg, 39adj, 40be, 41adfh, 42beh, 43be, 44f, 46beh, 47b.

```

< set the naflang parameter 37c > ≡
if
  [ -z "${naflang}" ]
then
  naffile='mktemp -t naf.XXXXXX'
  cat >$naffile
  naflang='cat $naffile | python $envbindir/langdetect.py'
  export naflang
  cat $naffile | $scriptpath
  result=$?
  rm $naffile
  exit $result
fi
◇

```

Fragment referenced in 37a, 45c.

Uses: `naflang` 50b.

5.1 Parameters in module-scripts

Some modules need parameters. All modules need a language specification. The language can be passed as exported variable `naflang`, but it can also be passed as argument `-l`. Furthermore, some modules need contact with a Spotlight server. With the arguments `-h` and `-b` the host and port of a running Spotlight-server can be passed.

Let us assess a “Parameter-passing” hierarchy for `run` scripts. Basically a “run” script uses default values encoded in the `run` script itself. These values can be overruled by environment parameters. Both default and environment parameter settings can be overruled by options that are provided to the `run` commands.

Let us adhere to the policy that we use short one-letter options in `run` scripts, that can be parsed with `getopts`.

The code to obtain command-line arguments in Bash has been obtained from [Stackoverflow](#). The following fragment reads the arguments `-l language`, `-h spotlighthost` and `-p spotlightport`:

```

< start of module-script 38a > ≡
  < get location of the script (38b DIR ) 53a >
  cd $DIR
  source ../progenv
  ◇

```

Fragment never referenced.

5.1.1 Tokeniser

The tokenizer is the simplest of the modules. It needs Java version 1.8. On installation it compiles a Java JAR file, and this is used in the run script.

```

< install the modules 38c > ≡
  gitinst https://github.com/PaulHuygen/ixa-pipe-tok.git ixa-pipe-
  tok 1a5b0f76e13315f9a1a75525e93d0789ccf9383c
  ◇

```

Fragment defined by 38cfi, 39cfi, 40adg, 41c, 42adg, 43adg, 44e, 45ab, 46adg, 47a.
 Fragment referenced in 35d.

```

"../bin/tok" 38d ≡
  < contents of shorthand-script (38e ixa-pipe-tok ) 37a >
  ◇

```

5.1.2 Topic detection tool.

The topic detection tool uses Java.

```

< install the modules 38f > ≡
  gitinst https://github.com/PaulHuygen/ixa-pipe-topic.git ixa-pipe-
  topic 31b0554d22e3a38f980086cfc7c4627d425344e9
  ◇

```

Fragment defined by 38cfi, 39cfi, 40adg, 41c, 42adg, 43adg, 44e, 45ab, 46adg, 47a.
 Fragment referenced in 35d.

```

"../bin/topic" 38g ≡
  < contents of shorthand-script (38h ixa-pipe-topic ) 37a >
  ◇

```

5.1.3 Morphosyntactic Parser and Alpino

The morphosyntactic parser is in fact a wrapper around Alpino. We have installed Alpino in section ???. The morpho-syntactic parser expects Alpino to be located in \$envdir/Alpino.

```

< install the modules 38i > ≡
  gitinst https://github.com/PaulHuygen/morphosyntactic_parser_nl.git morphosyntactic_parser_nl 29601f8
  ◇

```

Fragment defined by 38cfi, 39cfi, 40adg, 41c, 42adg, 43adg, 44e, 45ab, 46adg, 47a.
 Fragment referenced in 35d.

```
"../bin/mor" 39a≡
  ⟨ contents of shorthand-script (39b morphosyntactic_parser_nl ) 37a ⟩
◇
```

5.1.4 Pos tagger

Use the pos-tagger from EHU for English documents.

```
⟨ install the modules 39c ⟩ ≡
  gitinst https://github.com/PaulHuygen/ixa-pipe-pos.git ixa-pipe-
  pos f6de665c2b155828e9ca4d30b38451c5faaaeb9c
◇
```

Fragment defined by 38cfi, 39cfi, 40adg, 41c, 42adg, 43adg, 44e, 45ab, 46adg, 47a.
 Fragment referenced in 35d.

```
"../bin/pos" 39d≡
  ⟨ contents of shorthand-script (39e ixa-pipe-pos ) 37a ⟩
◇
```

5.1.5 Named entity recognition (NERC)

```
⟨ install the modules 39f ⟩ ≡
  gitinst https://github.com/ixa-ehu/ixa-pipe-nerc ixa-pipe-
  nerc e6197f960aeb383dd9c67cd329f8716dff96d7a0
◇
```

Fragment defined by 38cfi, 39cfi, 40adg, 41c, 42adg, 43adg, 44e, 45ab, 46adg, 47a.
 Fragment referenced in 35d.

```
"../bin/nerc" 39g≡
  ⟨ contents of shorthand-script (39h ixa-pipe-nerc ) 37a ⟩
◇
```

5.1.6 Word-sense disambiguation (WSD)

```
⟨ install the modules 39i ⟩ ≡
  gitinst https://github.com/PaulHuygen/svm_wsd.git svm_wsd eae13e95c215bc359431c50349a8aaf0270307c2
◇
```

Fragment defined by 38cfi, 39cfi, 40adg, 41c, 42adg, 43adg, 44e, 45ab, 46adg, 47a.
 Fragment referenced in 35d.

```
"../bin/wsd" 39j≡
  ⟨ contents of shorthand-script (39k svm_wsd ) 37a ⟩
◇
```

5.1.7 NED

The NED module is rather picky about the structure of the NAF file. In any case, it does not accept a file that has been produced by the ontotagger. Hence, in a pipeline NED should be executed before the ontotagger.

The NED module wants to consult the Dbpedia Spotlight server, so that one has to be installed somewhere. For this moment, let us suppose that it has been installed on localhost.

```
<install the modules 40a> ≡
    gitinst https://github.com/ixa-ehu/ixa-pipe-ned.git ixa-pipe-
    ned 211c3a0f4da7a8007fdc1493494fc2036169efd7
    ◇
```

Fragment defined by 38cfi, 39cfi, 40adg, 41c, 42adg, 43adg, 44e, 45ab, 46adg, 47a.
Fragment referenced in 35d.

```
"../bin/ned" 40b≡
    <contents of shorthand-script (40c ixa-pipe-ned ) 37a>
    ◇
```

5.1.8 Dark-entity relinker

The “Dark Entity Relinker” tries to link “Dark entities” (named entities that have not been recognized) to the link of a known entity with a similar name structure that has been found in the same text.

```
<install the modules 40d> ≡
    gitinst https://github.com/cltl/entity-relink-pipeline.git entity-relink-
    pipeline d788b0c402343f9290d3c84c0501377a63e3d98a
    ◇
```

Fragment defined by 38cfi, 39cfi, 40adg, 41c, 42adg, 43adg, 44e, 45ab, 46adg, 47a.
Fragment referenced in 35d.

```
"../bin/dere1" 40e≡
    <contents of shorthand-script (40f entity-relink-pipeline ) 37a>
    ◇
```

5.1.9 Heideltime

The code for Heideltime can be found in [Github](#). This repo contains an adapted Jar file.

Use Heideltime via a wrapper, `ixa-pipe-time`, obtained from [Github](#).

Although suggested otherwise, Heideltime seems not to use Treetagger. It works

```
<install the modules 40g> ≡
    gitinst https://github.com:PaulHuygen/ixa-pipe-time.git ixa-pipe-
    time 76eed04f7332c41e289ca97b8ddec604235291bb
    ◇
```

Fragment defined by 38cfi, 39cfi, 40adg, 41c, 42adg, 43adg, 44e, 45ab, 46adg, 47a.
Fragment referenced in 35d.


```
"../bin/heideltime" 41a≡
  ⟨ contents of shorthand-script (41b ixa-pipe-time ) 37a ⟩
◇
```

5.1.10 Ontotagger, Framenet-SRL and nominal events

- Een directory voor drie modules.
- Verwacht module **vua-resources** in een parallele directory.

The three modules ontotagger (aka “predicatematrix”), Framenet-SRL and nominal event detection are based on the same software packages and resources. The three modules need the same jar **ontotagger-1.0-jar-with-dependencies.jar**, they need resources from the **cltl/vua-resources** Github repository and they are going to execute a script that resides in the scripts directory of the **cltl/OntoTagger** repository. So, what we have to do is:

1. Install from the **cltl/OntoTagger** repository.
2. Create the jar and put it in an appropriate place.
3. install from the **cltl/vua-resources** repository.
4. generate a script for each of the modules.

In fact, items 2 and 3 are performed by script **install.sh** from the **OntoTagger** repository.

⟨ install the modules 41c ⟩ ≡

```
gitinst https://github.com:PaulHuygen/OntoTagger.git OntoTagger 31b0554d22e3a38f980086cfc7c4627d42
◇
```

Fragment defined by 38cfi, 39cfi, 40adg, 41c, 42adg, 43adg, 44e, 45ab, 46adg, 47a.

Fragment referenced in 35d.

The “Ontotagger” script:

```
"../bin/onto" 41d≡
  ⟨ contents of shorthand-script (41e OntoTagger ) 37a ⟩
◇
```

The “Nominal Event Coreference” script:

```
"../bin/nomevent" 41f≡
  ⟨ contents of shorthand-script (41g Nominal_Events ) 37a ⟩
◇
```

The “Framenet SRL” script:

```
"../bin/framesrl" 41h≡
  ⟨ contents of shorthand-script (41i Framenet_SRL ) 37a ⟩
◇
```

5.1.11 NED-reranker

install the modules 42a \equiv

```
gitinst https://github.com:PaulHuygen/NWRDomainModel.git NWRDomainModel 6af97d8e25e451654af2a88956
```

◇

Fragment defined by 38cfi, 39cfi, 40adg, 41c, 42adg, 43adg, 44e, 45ab, 46adg, 47a.

Fragment referenced in 35d.

"../bin/nedrer" 42b \equiv

contents of shorthand-script (42c NWRDomainModel) 37a

◇

5.1.12 Wikify module

Wikify needs spotlight.

install the modules 42d \equiv

```
gitinst https://github.com:PaulHuygen/ixa-pipe-wikify.git ixa-pipe-wikify dc9e085dc3733d5f8367335f3980ac95dc127d17
```

◇

Fragment defined by 38cfi, 39cfi, 40adg, 41c, 42adg, 43adg, 44e, 45ab, 46adg, 47a.

Fragment referenced in 35d.

"../bin/wikify" 42e \equiv

contents of shorthand-script (42f ixa-pipe-wikify) 37a

◇

5.1.13 UKB

The UKB WSD module is up to now only available from closed repositories. There exists a repository [ukb](#) in Git, but this does not seem to include the scripts to process NAF. Therefore, we need to have the repo available beforehand.

install the modules 42g \equiv

```
# UKB
ballinst 20170906_EHU-ukb.v30.tgz
```

◇

Fragment defined by 38cfi, 39cfi, 40adg, 41c, 42adg, 43adg, 44e, 45ab, 46adg, 47a.

Fragment referenced in 35d.

"../bin/m4_ukbcript" 42h \equiv

contents of shorthand-script (42i EHU-ukb.v30) 37a

◇

5.1.14 IMS-WSD

install the modules 43a \equiv

```
gitinst https://github.com:PaulHuygen/it_makes_sense_WSD.git it_makes_sense_WSD 58c3ec30d6f7f8e8a4f5c
◇
```

Fragment defined by 38cfi, 39cfi, 40adg, 41c, 42adg, 43adg, 44e, 45ab, 46adg, 47a.
Fragment referenced in 35d.

"../bin/ewsd" 43b \equiv

contents of shorthand-script (43c it_makes_sense_WSD) 37a \diamond

5.1.15 Semantic Role labelling

install the modules 43d \equiv

```
gitinst m4_nl_srlgit vua-srl-nl 9066185f4538543e887a1f61b679967cbe6eae8
◇
```

Fragment defined by 38cfi, 39cfi, 40adg, 41c, 42adg, 43adg, 44e, 45ab, 46adg, 47a.
Fragment referenced in 35d.

"../bin/srl" 43e \equiv

contents of shorthand-script (43f vua-srl-nl) 37a \diamond

5.1.16 SRL server for English

As far as I know, the English SRL for Newsreader, **EHU-srl-server**, is not yet open-source. Therefore, we still have to rely on the v3.0 version from the Newsreader repository.

This module has been set up as client-server application, making it less suitable for this general pipeline-structure. It means that the server ought to have been started before processing documents.

For now, we only implement the client. The client checks whether some process listens on port 5005 and aborts if that is not the case.

install the modules 43g \equiv

```
# eSRL-server
ballinst 20170906_EHU-srl-server.tgz
◇
```

Fragment defined by 38cfi, 39cfi, 40adg, 41c, 42adg, 43adg, 44e, 45ab, 46adg, 47a.
Fragment referenced in 35d.

Generate script to start the eSRL server.

```
"../env/bin/start_eSRL_server" 44a≡
#!/bin/bash
< get location of the script (44b thisdir ) 53a >
cd $thisdir
cd ../../
source ./progenv
moddir=$modulesdir/EHU-srl-server
cd $moddir
pidFile=$envdir/SRLServer.pid
echo $$ > ${pidFile}
exec java -Xms2500m -cp ${moddir}/IXA-EHU-srl-3.0.jar ixa.srl.SRLServer en
◇
```

```
"../env/bin/stop_eSRL_server" 44c≡
#!/bin/bash
< get location of the script (44d thisdir ) 53a >
cd $thisdir
cd ../../
source ./progenv
moddir=$modulesdir/EHU-srl-server
cd $moddir
pidFile=$envdir/SRLServer.pid
if
[ -e ${pidFile} ]
then
SRLServerPID=$(cat ${pidFile})
kill -9 $SRLServerPID
echo 'SRL Server with PID=$SRLServerPID' stopped'
rm -f ${pidFile}
else
echo 'SRL Server not running'
fi
◇
```

5.1.17 srl-Dutch nominals

```
< install the modules 44e > ≡
gitinst https://github.com/newsreader/vua-srl-dutch-nominal-events vua-srl-dutch-
nominal-events fcf3985e1ee380ba5392c3532c9e6d0f243d8247
◇
```

Fragment defined by 38cfi, 39cfi, 40adg, 41c, 42adg, 43adg, 44e, 45ab, 46adg, 47a.
 Fragment referenced in 35d.

```
"../bin/srl-dutch-nominals" 44f≡
< contents of shorthand-script (44g vua-srl-dutch-nominal-events ) 37a >
◇
```

5.1.18 FBK-time, FBK-temprel, FBK-causalrel

The three modules FBK-time, FBK-temprel, FBK-causalrel are, as far as I know, not open-source yet. So, now we need to install from snapshot.

```

< install the modules 45a > ≡
ballinst 20170831_FBK-time.v30.tgz
ballinst 20170831_FBK-temprel.v30.tgz
ballinst 20170816_FBK-causalrel.v30.tgz
◇

```

Fragment defined by 38cfi, 39cfi, 40adg, 41c, 42adg, 43adg, 44e, 45ab, 46adg, 47a.
 Fragment referenced in 35d.

5.1.19 Factuality

We have module `vua_factuality` to identify event-factuality in English texts and module `multilingual_factuality` to identify event-factuality in non-English texts.

```

< install the modules 45b > ≡

```

```

gitinst https://github.com:PaulHuygen/multilingual_factuality.git multilingual_factuality 1f47e7bf4c5
gitinst https://github.com:PaulHuygen/vua_factuality.git vua_factuality 412f9917eb7a6e395c05af84abfc7
◇

```

Fragment defined by 38cfi, 39cfi, 40adg, 41c, 42adg, 43adg, 44e, 45ab, 46adg, 47a.
 Fragment referenced in 35d.

The shorthandsript runs the module in `vua_factuality` for english documents and it runs the module in `multilingual_factuality` for documents in other languages.

```

"../bin/factuality" 45c≡
#!/bin/bash
< get location of the script (45d thisdir ) 53a >
scriptname=${0##*/}
scriptpath=$thisdir/$scriptname
cd ${thisdir}
cd ..
source ./progenv
< set the naflang parameter 37c >
cd ${thisdir}
if
[ "${naflang}" == "en" ]
then
cat | ../modules/vua_factuality/run
else
cat | ../modules/multilingual_factuality/run
fi
◇

```

5.1.20 Opinion miner

The opinion-miner needs models that are not yet available from an open repository. The installer expects the variable `opinion_models_ball_path` to contain the full path to the tarball with the opinion-models.

```

<install the modules 46a> ≡
    export opinion_models_ball_path=${snapshotdir}/models_opinion_miner_deluxePP.tgz
    gitinst https://github.com/PaulHuygen/opinion_miner_deluxePP opinion_miner_deluxePP 93cd9e4a35f5964ed
    ◇

```

Fragment defined by 38cfi, 39cfi, 40adg, 41c, 42adg, 43adg, 44e, 45ab, 46adg, 47a.
 Fragment referenced in 35d.
 Defines: opinion_models_ball_path Never used.

```

"../bin/opinimin" 46b≡
    <contents of shorthand-script (46c opinion_miner_deluxePP ) 37a>
    ◇

```

5.1.21 Event coreference

The event-coreference module is language-independent. It is a module in a jar-file that can be built with the Github <http://github.com:PaulHuygen/EventCoreference.git> repo. The module uses resources from the `vua-resources` Github repo.

```

<install the modules 46d> ≡
    gitinst http://github.com:PaulHuygen/EventCoreference.git EventCoreference 5c3ec554eb3c5a56c448b804f2
    ◇

```

Fragment defined by 38cfi, 39cfi, 40adg, 41c, 42adg, 43adg, 44e, 45ab, 46adg, 47a.
 Fragment referenced in 35d.

```

"../bin/evcoref" 46e≡
    <contents of shorthand-script (46f EventCoreference ) 37a>
    ◇

```

5.1.22 Corefgraph

The corefgraph module is currently still a hacked version of the module that can be found in the newsreader vs. 3.0 repository. It is stored in the snapshot-directory.

So, install the module from there.

```

<install the modules 46g> ≡
    # EHU-corefgraph
    ballinst 20170831_EHU-corefgraph.v30.tgz
    ◇

```

Fragment defined by 38cfi, 39cfi, 40adg, 41c, 42adg, 43adg, 44e, 45ab, 46adg, 47a.
 Fragment referenced in 35d.

```

"../bin/coref-graph" 46h≡
    <contents of shorthand-script (46i EHU-corefgraph.v30 ) 37a>
    ◇

```

5.2 Constituent parser

```

<install the modules 47a> ≡
  gitinst https://github.com/ixa-ehu/ixa-pipe-parse.git ixa-pipe-
  parse bf8e91f829e963fe16684a12ad241f8b1aab251d
  ◇

```

Fragment defined by 38cfi, 39cfi, 40adg, 41c, 42adg, 43adg, 44e, 45ab, 46adg, 47a.
 Fragment referenced in 35d.

```

"../bin/constpars" 47b≡
  <contents of shorthand-script (47c ixa-pipe-parse ) 37a>
  ◇

```

6 Utilities

6.1 Language detection

The following script `../env/bin/langdetect.py` discerns the language of the NAF document that it reads from standard in. If it cannot find the language, it prints `unknown`. The macro `set the language variable` uses this script to set variable `naflang`. All pipeline modules expect that this variable has been set.

```

"../env/bin/langdetect.py" 47d≡
  #!/usr/bin/env python
  # langdetect -- Detect the language of a NAF document.
  #
  import xml.etree.ElementTree as ET
  import sys
  import re
  xmldoc = sys.stdin.read()
  #print xmldoc
  root = ET.fromstring(xmldoc)
  # print root.attrib['lang']
  lang = "unknown"
  for k in root.attrib:
    if re.match(".*lang$", k):
      language = root.attrib[k]
  print(language)
  ◇

```

Uses: `print` 59b.

```

<make scripts executable 47e> ≡
  chmod 775 ../env/bin/langdetect.py
  ◇

```

Fragment defined by 7o, 8c, 17a, 24f, 36a, 47e, 65c.
 Fragment referenced in 65d.

The module-scripts depend on the existence of variable `naflang`. In most cases this is not a problem because the scripts run in a surrounding script that sets `naflang`. However, a users may occasionally run a module-script stand-alone e.g. to debug. In that case, we can read the language from the NAF, set variable `naflang`, and then run the module-script in a subshell. We assume that variable `scriptpath` contains the path of the script itself.

The macro does the following if **naflang** has not been set:

1. Save the content of standard in to a temporary file.
2. Run **langdetect** with the temporary file as input and set the **naflang** variable.
3. Run the script **\$scriptpath** (i.e. itself) with the temporary file as input.
4. Remove the temporary file.
5. Exit itself with the errorcode of the sub-script that it has run.

```
<run in subshell when naflang is not known 48a> ≡
if
  [ -z "${naflang+x}" ]
then
  naffile='mktemp -t naf.XXXXXX'
  cat >$naffile
  naflang='cat $naffile | python $envbindir/langdetect.py'
  export naflang
  cat $naffile | $scriptpath
  result=$?
  rm $naffile
  exit $result
fi
◇
```

Fragment never referenced.

Uses: **naflang** 50b.

```
<run only if language is English or Dutch 48b> ≡
if
  [ ! "$naflang" == "nl" ] && [ ! "$naflang" == "en" ]
then
  exit 6
fi
◇
```

Fragment never referenced.

Uses: **naflang** 50b.

6.2 Run-script and test-script

The script **nlpp** reads a NAF document from standard in and produces an annotated NAF on standard out. The script **test** annotates either a test-document that resides in the nuweb directory or a user-provided document and leaves the intermediate results in its working directory **nlpp/test**, so that, in case of problems, it is easy traceable what went wrong.

The annotation process involves a sequence in which an NLP module reads a file that contains the output from a previous module (or the input NAF file), processes it and writes the result in another file.

The following function, **runmodule**, performs the action of a single module in the sequence. It needs three arguments: 1) the name of the NAF file that the previous module produced or the input file; 2) the name of directory in which the module resides and 3) the name of the output NAF.

The function uses variable **moduleresult** to decide whether it is really going to annotate. If this variable is "false" (i.e., not equal to zero), this means that one of the previous modules failed, and it is of no use to process the input file. In that case, the function leaves **moduleresult** as it is and does not process the input-file. Otherwise, it will process the input-file and it sets **moduleresult** to the result of the processing module.


```

⟨function to run a module 49a⟩ ≡
    export moduleresult=0

    function runmodule {
        local infile=$1
        local modulecommand=$modulesdir/$2/run
        local outfile=$3
        if
            [ $moduleresult -eq 0 ]
        then
            cat $infile | $modulecommand > $outfile
            moduleresult=$?
            if
                [ $moduleresult -gt 0 ]
            then
                failmodule=$modulecommand
                echo "Failed: module $modulecommand; result $moduleresult" >&2
                exit $moduleresult
            else
                echo "Completed: module $modulecommand; result $moduleresult" >&2
            fi
        fi
    }

```

◇

Fragment referenced in 52ab.

Defines: `moduleresult` 52ab, `runmodule` 49b, 50a.

Use the function to annotate a NAF file that `infile` points to and write the result in a file that `outfile` points to:

```

⟨annotate dutch document 49b⟩ ≡
    runmodule $infile      ixa-pipe-tok      tok.naf
    runmodule tok.naf      ixa-pipe-topic    top.naf
    runmodule top.naf      morphosyntactic_parser_nl    pos.naf
    runmodule pos.naf      ixa-pipe-nerc      nerc.naf
    runmodule nerc.naf     svm_wsd            wsd.naf
    runmodule wsd.naf      ixa-pipe-ned       ned.naf
    runmodule ned.naf      entity-relink-pipeline    derel.naf
    runmodule derel.naf    ixa-pipe-time      times.naf
    runmodule times.naf    OntoTagger          onto.naf
    runmodule onto.naf     vua-srl-nl         srl.naf
    runmodule srl.naf      Nominal_Events     nomev.naf
    runmodule nomev.naf    vua-srl-dutch-nominal-events    psrl.naf
    runmodule psrl.naf     Framenet_SRL      fsrl.naf
    runmodule fsrl.naf     multilingual_factuality fact.naf
    runmodule fact.naf     EventCoreference   $outfile

```

◇

Fragment referenced in 50b.

Uses: `runmodule` 49a.

Similar for an English naf:

```

⟨ annotate english document 50a ⟩ ≡
  runmodule $infile      ixa-pipe-tok tok.naf
  runmodule tok.naf      ixa-pipe-topic top.naf
  runmodule top.naf      ixa-pipe-pos pos.naf
  runmodule pos.naf      ixa-pipe-parse parse.naf
  runmodule parse.naf    ixa-pipe-nerc nerc.naf
  runmodule nerc.naf     svm_wsd wsd.naf
  runmodule wsd.naf      ixa-pipe-ned ned.naf
  runmodule ned.naf      entity-relink-pipeline derel.naf
  runmodule derel.naf    NWRDomainModel nedr.naf
  runmodule nedr.naf     ixa-pipe-wikify wikif.naf
  runmodule wikif.naf    EHU-ukb.v30 ukb.naf
  runmodule ukb.naf      it_makes_sense_WSD wsd.naf
  runmodule wsd.naf      EHU-corefgraph.v30 corefg.naf
  runmodule corefg.naf   EHU-srl-server esrl.naf
  runmodule esrl.naf     FBK-time.v30 ftime.naf
  runmodule ftime.naf    FBK-temprel.v30 ftemp.naf
  runmodule ftemp.naf    FBK-causalrel.v30 fcausal.naf
  runmodule fcausal.naf  EventCoreference evcoref.naf
  runmodule evcoref.naf  vua_factuality fact.naf
  ◇

```

Fragment referenced in 50b.

Uses: `runmodule` 49a.

Determine the language and select one of the above macro's to annotate the document. In fact, consider the document as an English document unless `naflang` is "nl"

```

⟨ annotate 50b ⟩ ≡
  naflang='cat $infile | /home/paul/temp/nlpp/env/bin/langdetect.py'
  export naflang
  if
    [ "$naflang" == "nl" ]
  then
    ⟨ annotate dutch document 49b ⟩
  else
    ⟨ annotate english document 50a ⟩
  fi
  ◇

```

Fragment referenced in 52ab.

Defines: `naflang` 23ab, 24g, 25a, 26b, 37c, 45c, 48ab, 51.

Use the above "annotate" macro in a test script and in a run script. The scripts set a working directory and put the input-file in it, and then annotate it.

The test-script uses a special test-directory and leaves it behind when it is finished. If the user specified a language, the script copies a NAF testfile from the nuweb directory as input-file. Otherwise, the script expects the test-directory to be present, with an input-file (named `in.naf`) in it.

$\langle \text{get a testfile and set naflang or die 51} \rangle \equiv$

```

cd $workdir
naflang=""
if
  [ "$1" == "en" ]
then
  cp $nuwebdir/test.en.in.naf $infile
  export naflang="en"
else
  if
    [ "$1" == "nl" ]
  then
    cp $nuwebdir/test.nl.in.naf $infile
    export naflang="nl"
  fi
fi
if
  [ -e $infile ]
then
  if
    [ "$naflang" == "" ]
  then
    naflang='cat $infile | python $envbindir/langdetect.py'
  fi
else
  echo "Please supply test-file $workdir/$infile or specify language"
  exit 4
fi
◇

```

Fragment referenced in 52a.

Uses: naflang 50b.

This is the test-script:

```

"../bin/test" 52a≡
#!/bin/bash
DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )"
rdir=$(dirname "$DIR")
source $rdir/progenv
oldd='pwd'
workdir=$piperoot/test
mkdir -p $workdir
cd $workdir
infile=in.naf
outfile=out.naf
< get a testfile and set naflang or die 51 >
< find a spotlightserver or exit 24g >
< function to run a module 49a >
< annotate 50b >
if
[ $moduleresult -eq 0 ]
then
echo Test succeeded.
else
echo Something went wrong.
fi
exit $moduleresult
◇

```

Uses: moduleresult 49a, piperoot 10d.

The run-script `nlpp` reads a “raw” naf from standard in and produces an annotated naf on standard out. It creates a temporary directory to store intermediate results from the modules and removes this directory afterwards.

```

"../bin/nlpp" 52b≡
#!/bin/bash
oldd='pwd'
workdir='mktemp -d -t nlpp.XXXXXX'
cd $workdir
cat >$workdir/$infile
< function to run a module 49a >
< annotate 50b >
if
[ $moduleresult -eq 0 ]
then
cat $outfile
fi
cd $oldd
rm -rf $workdir
exit $moduleresult
◇

```

Uses: moduleresult 49a.

7 Miscellaneous

7.1 Locate the path to the script itself

The following macro finds the directory in which the script itself or the sourced script itself is located.

```

⟨ get location of the script 53a ⟩ ≡
    @1="$( cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )"
    ◇

```

Fragment referenced in 7a, 8a, 10a, 16f, 22b, 35d, 37a, 38a, 44ac, 45c.

7.2 Logging

Write log messages to standard out if variable LOGLEVEL is equal to 1.

```

⟨ variables of the module-installer 53b ⟩ ≡
    LOGLEVEL=1
    ◇

```

Fragment referenced in 35d.

```

⟨ logmess 53c ⟩ ≡
    if
    [ $LOGLEVEL -gt 0 ]
    then
    echo @1
    fi
    ◇

```

Fragment never referenced.

A How to read and translate this document

This document is an example of *literate programming* [2]. 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 macro's.

< a scrap 87e > \equiv

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_nlpp.w`. Figure 2 shows pathways to translate it into print-

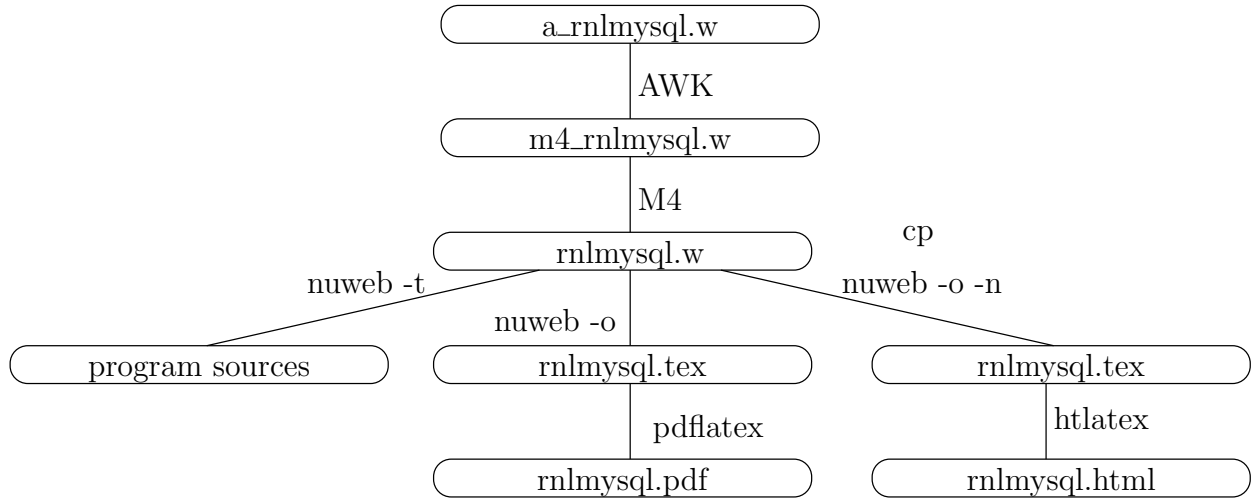


Figure 2: Translation of the raw code of this document into printable/viewable documents and into program sources. The figure shows the pathways and the main files involved.

able/viewable documents and to extract the program sources. Table 3 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 <code>xml/html</code>

Table 3: 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 55a> ≡
    NUWEB=../env/bin/nuweb
    ◇

```

Fragment defined by 55a, 56a, 58ab, 60c, 62b, 65a.

Fragment referenced in 55b.

Uses: nuweb 61d.

A.3 The Makefile for this project.

This chapter assembles the Makefile for this project.

```

"Makefile" 55b≡
    <default target 55c>

    <parameters in Makefile 55a, ... >

    <impliciete make regels 58c, ... >
    <expliciete make regels 56b, ... >
    <make targets 55d, ... >
    ◇

```

The default target of make is all.

```

<default target 55c> ≡
    all : <all targets 55e>
    .PHONY : all
    ◇

```

Fragment referenced in 55b.

Defines: all Never used, PHONY 59a.

```

<make targets 55d> ≡
    clean:
        ../env/bin/clean_infrastructure
    ◇

```

Fragment defined by 55d, 59b, 60a, 63c, 65bd, 66abc.

Fragment referenced in 55b.

The default is, to install nlpp.

```

<all targets 55e> ≡
    install◇

```

Fragment referenced in 55c.

Uses: install 66a.

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

\langle *parameters in Makefile 56a* $\rangle \equiv$
`.SUFFIXES: .pdf .w .tex .html .aux .log .php`

◇

Fragment defined by 55a, 56a, 58ab, 60c, 62b, 65a.
 Fragment referenced in 55b.
 Defines: SUFFIXES Never used.
 Uses: pdf 59b.

A.4 Get Nuweb

An annoying problem is, that this program uses nuweb, a utility that is seldom installed on a computer. Therefore, we are going to install that first if it is not present. Unfortunately, nuweb is hosted on sourceforge and it is difficult to achieve automatic downloading from that repository. Therefore I copied one of the versions on a location from where it can be downloaded with a script.

Put the nuweb binary in the nuweb subdirectory, so that it can be used before the directory-structure has been generated.

\langle *expliciete make regels 56b* $\rangle \equiv$

```
nuweb: $(NUWEB)

$(NUWEB): ../nuweb-1.58
    mkdir -p ../env/bin
    cd ../nuweb-1.58 && make nuweb
    cp ../nuweb-1.58/nuweb $(NUWEB)
```

◇

Fragment defined by 56bd, 57ab, 59a, 60d, 62c, 63b.
 Fragment referenced in 55b.
 Uses: nuweb 61d.

\langle *clean up 56c* $\rangle \equiv$

```
rm -rf ../nuweb-1.58
```

◇

Fragment never referenced.
 Uses: nuweb 61d.

\langle *expliciete make regels 56d* $\rangle \equiv$

```
../nuweb-1.58:
    cd .. && wget http://kyoto.let.vu.nl/~huygen/nuweb-1.58.tgz
    cd .. && tar -xzf nuweb-1.58.tgz
```

◇

Fragment defined by 56bd, 57ab, 59a, 60d, 62c, 63b.
 Fragment referenced in 55b.
 Uses: nuweb 61d.

A.5 Pre-processing

To make usable things from the raw input `a_nlpp.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.5.1 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 57a> ≡
m4_nlpp.w : a_nlpp.w
          gawk '{if(match($$0, "@%")) {printf("%s", substr($$0,1,RSTART-
1))} else print}' a_nlpp.w \
          | gawk '{gsub(/[\][\]$$/ , "$$");print}' > m4_nlpp.w
```

◇

Fragment defined by 56bd, 57ab, 59a, 60d, 62c, 63b.

Fragment referenced in 55b.

Uses: print 59b.

A.5.2 Run the M4 pre-processor

```
<expliciete make regels 57b> ≡
nlpp.w : m4_nlpp.w inst.m4
        m4 -P m4_nlpp.w > nlpp.w
```

◇

Fragment defined by 56bd, 57ab, 59a, 60d, 62c, 63b.

Fragment referenced in 55b.

A.6 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.6.1 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 58a > ≡
 FIGFILES=fileschema directorystructure

◇

Fragment defined by 55a, 56a, 58ab, 60c, 62b, 65a.
 Fragment referenced in 55b.
 Defines: FIGFILES 58b.

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 58b > ≡
 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 55a, 56a, 58ab, 60c, 62b, 65a.
 Fragment referenced in 55b.
 Defines: FIGFILENAMES Never used, PDFT_NAMES 60a, PDF_FIG_NAMES 60a, PST_NAMES Never used,
 PS_FIG_NAMES Never used.
 Uses: FIGFILES 58a.

Create the graph files with program `fig2dev`:

< impliciete make regels 58c > ≡
 %.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 58c, 63a.
 Fragment referenced in 55b.
 Defines: `fig2dev` Never used.

A.6.2 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 `nlpp.bib`. To create this file, copy the auxiliary file to another file `auxfil.aux`, but replace the argument of the command `\bibdata{nlpp}` 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 59a > ≡
    bibfile : nlpp.aux /home/paul/bin/mkportbib
              /home/paul/bin/mkportbib nlpp litprog

    .PHONY : bibfile
◇
```

Fragment defined by 56bd, 57ab, 59a, 60d, 62c, 63b.

Fragment referenced in 55b.

Uses: PHONY 55c.

A.6.3 Create a printable/viewable document

Make a PDF document for printing and viewing.

```
< make targets 59b > ≡
    pdf : nlpp.pdf

    print : nlpp.pdf
           lpr nlpp.pdf

    view : nlpp.pdf
           evince nlpp.pdf

◇
```

Fragment defined by 55d, 59b, 60a, 63c, 65bd, 66abc.

Fragment referenced in 55b.

Defines: pdf 56a, 60a, print 17bc, 20a, 24g, 32d, 47d, 57a, 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`.

```

< make targets 60a > ≡
    nlpp.pdf : nlpp.w $(W2PDF) $(PDF_FIG_NAMES) $(PDFT_NAMES)
              chmod 775 $(W2PDF)
              $(W2PDF) $*

```

◇

Fragment defined by 55d, 59b, 60a, 63c, 65bd, 66abc.

Fragment referenced in 55b.

Uses: pdf 59b, PDFT_NAMES 58b, PDF_FIG_NAMES 58b.

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.

```

< directories to create 60b > ≡
    ../nuweb/bin ◇

```

Fragment defined by 9abcd, 60b.

Fragment referenced in 65b.

Uses: nuweb 61d.

```

< parameters in Makefile 60c > ≡
    W2PDF=../nuweb/bin/w2pdf
    ◇

```

Fragment defined by 55a, 56a, 58ab, 60c, 62b, 65a.

Fragment referenced in 55b.

Uses: nuweb 61d.

```

< expliciete make regels 60d > ≡
    $(W2PDF) : nlpp.w $(NUWEB)
              $(NUWEB) nlpp.w
    ◇

```

Fragment defined by 56bd, 57ab, 59a, 60d, 62c, 63b.

Fragment referenced in 55b.

```

"../nuweb/bin/w2pdf" 60e≡
    #!/bin/bash
    # w2pdf -- compile a nuweb file
    # usage: w2pdf [filename]
    # 20180301 at 1458h: Generated by nuweb from a_nlpp.w
    NUWEB=../env/bin/nuweb
    LATEXCOMPILER=pdflatex
    < filenames in nuweb compile script 61b >
    < compile nuweb 61a >

```

◇

Uses: nuweb 61d.

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

```

⟨ compile nuweb 61a ⟩ ≡
    NUWEB=/home/paul/temp/nlpp/env/bin/nuweb
    ⟨ run the processors until the aux file remains unchanged 62a ⟩
    ⟨ remove the copy of the aux file 61c ⟩
    ◇

```

Fragment referenced in 60e.

Uses: nuweb 61d.

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 L^AT_EX 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 61b ⟩ ≡
    nufil=$1
    trunk=${1%.*}
    texfil=${trunk}.tex
    auxfil=${trunk}.aux
    oldaux=old.${trunk}.aux
    indexfil=${trunk}.idx
    oldindexfil=old.${trunk}.idx
    ◇

```

Fragment referenced in 60e.

Defines: auxfil 62a, 64ab, indexfil 62a, 64a, nufil 61d, 64ac, oldaux 61c, 62a, 64ab, oldindexfil 62a, 64a, texfil 61d, 64ac, trunk 61d, 64acd.

Remove the old copy if it is no longer needed.

```

⟨ remove the copy of the aux file 61c ⟩ ≡
    rm $oldaux
    ◇

```

Fragment referenced in 61a, 63e.

Uses: oldaux 61b, 64a.

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

```

⟨ run the three processors 61d ⟩ ≡
    $NUWEB $nufil
    $LATEXCOMPILER $texfil
    makeindex $trunk
    bibtex $trunk
    ◇

```

Fragment referenced in 62a.

Defines: bibtex 64cd, makeindex 64cd, nuweb 10e, 55a, 56bcd, 60bce, 61a, 62b, 63d.

Uses: nufil 61b, 64a, texfil 61b, 64a, trunk 61b, 64a.

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 62a ⟩ ≡

```
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 61d ⟩
  if [ $LOOPCOUNTER -ge 10 ]
  then
    cp $auxfil $oldaux
  fi;
done
◇
```

Fragment referenced in 61a.

Uses: auxfil 61b, 64a, indexfil 61b, oldaux 61b, 64a, oldindexfil 61b.

A.6.4 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`.

To create a HTML doc, we do the following:

1. Create a directory `../nuweb/html` for the HTML document.
2. Put the nuweb source in it, together with style-files that are needed (see variable `HTML-SOURCE`).
3. Put the script `w2html` in it and make it executable.
4. Execute the script `w2html`.

Make a list of the entities that we mentioned above:

⟨ parameters in Makefile 62b ⟩ ≡

```
htmldir=../nuweb/html
htmlsource=nlpp.w nlpp.bib html.sty artikel3.4ht w2html
htmlmaterial=$(foreach fil, $(htmlsource), $(htmldir)/$(fil))
htmltarget=$(htmldir)/nlpp.html
◇
```

Fragment defined by 55a, 56a, 58ab, 60c, 62b, 65a.

Fragment referenced in 55b.

Uses: nuweb 61d.

Make the directory:

⟨ expliciete make regels 62c ⟩ ≡

```
$(htmldir) :
  mkdir -p $(htmldir)
◇
```

Fragment defined by 56bd, 57ab, 59a, 60d, 62c, 63b.

Fragment referenced in 55b.

The rule to copy files in it:

```
< implicate make regels 63a > ≡
    $(htmldir)/% : % $(htmldir)
    cp $< $(htmldir)/
```

◇

Fragment defined by 58c, 63a.

Fragment referenced in 55b.

Do the work:

```
< expliciete make regels 63b > ≡
    $(htmltarget) : $(htmlmaterial) $(htmldir)
    cd $(htmldir) && chmod 775 w2html
    cd $(htmldir) && ./w2html nlpp.w
```

◇

Fragment defined by 56bd, 57ab, 59a, 60d, 62c, 63b.

Fragment referenced in 55b.

Invoke:

```
< make targets 63c > ≡
    htm : $(htmldir) $(htmltarget)
```

◇

Fragment defined by 55d, 59b, 60a, 63c, 65bd, 66abc.

Fragment referenced in 55b.

Create a script that performs the translation.

```
"w2html" 63d≡
    #!/bin/bash
    # w2html -- make a html file from a nuweb file
    # usage: w2html [filename]
    # [filename]: Name of the nuweb source file.
    # 20180301 at 1458h: Generated by nuweb from a_nlpp.w
    echo "translate " $1 >w2html.log
    NUWEB=/home/paul/temp/nlpp/env/bin/nuweb
    < filenames in w2html 64a >
```

```
< perform the task of w2html 63e >
```

◇

Uses: nuweb 61d.

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 63e > ≡
    < run the html processors until the aux file remains unchanged 64b >
    < remove the copy of the aux file 61c >
```

◇

Fragment referenced in 63d.

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 L^AT_EX 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 64a⟩ ≡
    nufil=$1
    trunk=${1%.*}
    texfil=${trunk}.tex
    auxfil=${trunk}.aux
    oldaux=old.${trunk}.aux
    indexfil=${trunk}.idx
    oldindexfil=old.${trunk}.idx
    ◇

```

Fragment referenced in 63d.

Defines: `auxfil` 61b, 62a, 64b, `nufil` 61bd, 64c, `oldaux` 61bc, 62a, 64b, `texfil` 61bd, 64c, `trunk` 61bd, 64cd.

Uses: `indexfil` 61b, `oldindexfil` 61b.

```

⟨run the html processors until the aux file remains unchanged 64b⟩ ≡
    while
        ! cmp -s $auxfil $oldaux
    do
        if [ -e $auxfil ]
        then
            cp $auxfil $oldaux
        fi
        ⟨run the html processors 64c⟩
    done
    ⟨run tex4ht 64d⟩
    ◇

```

Fragment referenced in 63e.

Uses: `auxfil` 61b, 64a, `oldaux` 61b, 64a.

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

```

⟨run the html processors 64c⟩ ≡
    $NUWEB -o -n $nufil
    latex $texfil
    makeindex $trunk
    bibtex $trunk
    htlatex $trunk
    ◇

```

Fragment referenced in 64b.

Uses: `bibtex` 61d, `makeindex` 61d, `nufil` 61b, 64a, `texfil` 61b, 64a, `trunk` 61b, 64a.

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 64d⟩ ≡
    tex '\def\filename{{nlpp}{idx}{4dx}{ind}} \input idxmake.4ht'
    makeindex -o $trunk.ind $trunk.4dx
    bibtex $trunk
    htlatex $trunk
    ◇

```

Fragment referenced in 64b.

Uses: `bibtex` 61d, `makeindex` 61d, `trunk` 61b, 64a.

A.7 Perform the installation

Run nuweb, but suppress the creation of the L^AT_EX 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 65a ⟩ ≡
    MKDIR = mkdir -p
```

◇

Fragment defined by 55a, 56a, 58ab, 60c, 62b, 65a.

Fragment referenced in 55b.

Defines: MKDIR 65b.

```
⟨ make targets 65b ⟩ ≡
    DIRS = ⟨ directories to create 9a, ... ⟩
```

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

◇

Fragment defined by 55d, 59b, 60a, 63c, 65bd, 66abc.

Fragment referenced in 55b.

Defines: DIRS 65d.

Uses: MKDIR 65a.

```
⟨ make scripts executable 65c ⟩ ≡
    chmod -R 775 ../bin/*
    chmod -R 775 ../env/bin/*
```

◇

Fragment defined by 7o, 8c, 17a, 24f, 36a, 47e, 65c.

Fragment referenced in 65d.

The target “sources” unpacks the nuweb file and creates the program scripts, i.e. the scripts that will apply modules on a NAF file and the script `install_modules` that installs the modules themselves and that creates the software environment the the modules need.

```
⟨ make targets 65d ⟩ ≡
    sources : nlpp.w $(DIRS) $(NUWEB)
              $(NUWEB) nlpp.w
              ⟨ make scripts executable 7o, ... ⟩
```

◇

Fragment defined by 55d, 59b, 60a, 63c, 65bd, 66abc.

Fragment referenced in 55b.

Uses: DIRS 65b.

The “install” target performs the complete installation.

```

< make targets 66a > ≡
    install : sources
              ../env/bin/make_infrastructure
              ../env/bin/install_modules

```

◇

Fragment defined by 55d, 59b, 60a, 63c, 65bd, 66abc.

Fragment referenced in 55b.

Defines: **install** 8d, 13a, 14b, 15b, 16a, 18a, 19b, 20b, 29ab, 30a, 31b, 34b, 36b, 55e, 66b.

A.8 Test whether it works

The targets **testnl** and **testen** perform the test-script to test the dutch resp. english pipeline.

```

< make targets 66b > ≡

    testnl : install test.nl.in.naf
              rm -rf ../test
              mkdir ../test
              cd ../test && ../bin/test nl

    testen : install test.en.in.naf
              rm -rf ../test
              mkdir ../test
              cd ../test && ../bin/test en

```

◇

Fragment defined by 55d, 59b, 60a, 63c, 65bd, 66abc.

Fragment referenced in 55b.

Defines: **testen** Never used, **testnl** Never used.

Uses: **install** 66a.

A.9 Restore paths after transplantation

When an existing installation has been transplanted to another location, many path indications have to be adapted to the new situation. The scripts that are generated by nuweb can be repaired by re-running nuweb. After that, configuration files of some modules must be modified.

```

< make targets 66c > ≡
    transplant :
              touch a_nlpp.w
              $(MAKE) sources
              ../env/bin/transplant

```

◇

Fragment defined by 55d, 59b, 60a, 63c, 65bd, 66abc.

Fragment referenced in 55b.

B References

B.1 Literature

References

- [1] Rodrigo Agerri, Itziar Aldabe, Zuhaitz Beloki, Egoitz Laparra1, Maddalen Lopez de Lacalle1, German Rigau, Aitor Soroa, Antske Fokkens, Ruben Izquierdo, Marieke van Erp, Piek Vossen, Christian Girardi, and Anne-Lyse Minard. Event detection, version 2, deliverable d4.2.2. Technical report, University of the Basque Country, IXA NLP group, feb 2015. <http://www.newsreader-project.eu/files/2012/12/NWR-D4-2-2.pdf>.
- [2] Donald E. Knuth. Literate programming. Technical report STAN-CS-83-981, Stanford University, Department of Computer Science, 1983.

C Indexes

C.1 Filenames

"../bin/check_start_spotlight" Defined by 22b, 24a.
 "../bin/constpars" Defined by 47b.
 "../bin/coref-graph" Defined by 46h.
 "../bin/derel" Defined by 40e.
 "../bin/evcoref" Defined by 46e.
 "../bin/ewsd" Defined by 43b.
 "../bin/factuality" Defined by 45c.
 "../bin/framesrl" Defined by 41h.
 "../bin/heideltime" Defined by 41a.
 "../bin/m4_ukbcript" Defined by 42h.
 "../bin/mor" Defined by 39a.
 "../bin/ned" Defined by 40b.
 "../bin/nedrer" Defined by 42b.
 "../bin/nerc" Defined by 39g.
 "../bin/nlpp" Defined by 52b.
 "../bin/nomevent" Defined by 41f.
 "../bin/onto" Defined by 41d.
 "../bin/opinimin" Defined by 46b.
 "../bin/pos" Defined by 39d.
 "../bin/srl" Defined by 43e.
 "../bin/srl-dutch-nominals" Defined by 44f.
 "../bin/test" Defined by 52a.
 "../bin/tok" Defined by 38d.
 "../bin/topic" Defined by 38g.
 "../bin/wikify" Defined by 42e.
 "../bin/wsd" Defined by 39j.
 "../env/bin/chasbang.awk" Defined by 17b.
 "../env/bin/clean_infrastructure" Defined by 8a.
 "../env/bin/install_modules" Defined by 35d.
 "../env/bin/langdetect.py" Defined by 47d.
 "../env/bin/make_infrastructure" Defined by 7a.
 "../env/bin/start_eSRL_server" Defined by 44a.
 "../env/bin/stop_eSRL_server" Defined by 44c.
 "../env/bin/tran" Defined by 16f.
 "../nuweb/bin/w2pdf" Defined by 60e.
 "../progenv" Defined by 10a.
 "Makefile" Defined by 55b.
 "w2html" Defined by 63d.

C.2 Macro's

- ⟨all targets 55e⟩ Referenced in 55c.
- ⟨annotate 50b⟩ Referenced in 52ab.
- ⟨annotate dutch document 49b⟩ Referenced in 50b.
- ⟨annotate english document 50a⟩ Referenced in 50b.
- ⟨apply script tran on the scripts in 17c⟩ Referenced in 16c.
- ⟨check listener on host, port 25b⟩ Referenced in 24a, 27b.
- ⟨check presence of javac in 1.8 12a⟩ Referenced in 13a.
- ⟨check presence of maven in 3.0.5 14a⟩ Referenced in 14b.
- ⟨check presence of perl in 5 19a⟩ Referenced in 19b.
- ⟨check presence of python3 in 3.6 15a⟩ Referenced in 15b.
- ⟨check whether a tarball is present in the snapshot 12b⟩ Referenced in 13a, 14b, 15b, 19b.
- ⟨check whether XML::LibXML is installed 19g⟩ Referenced in 19b.
- ⟨clean up 56c⟩ Not referenced.
- ⟨clean up after installation 14g⟩ Referenced in 8a.
- ⟨compile nuweb 61a⟩ Referenced in 60e.
- ⟨conditionally install Perl 20a⟩ Referenced in 7a.
- ⟨contents of shorthand-script 37a⟩ Referenced in 38dg, 39adgj, 40be, 41adfh, 42beh, 43be, 44f, 46beh, 47b.
- ⟨create python script and pip script 16b⟩ Referenced in 16a.
- ⟨default target 55c⟩ Referenced in 55b.
- ⟨directories to create 9abcd, 60b⟩ Referenced in 65b.
- ⟨download everything 11c, 28a⟩ Referenced in 11b.
- ⟨download stuff 13f, 18c, 28b, 33c⟩ Referenced in 28a.
- ⟨expliciete make regels 56bd, 57ab, 59a, 60d, 62c, 63b⟩ Referenced in 55b.
- ⟨extract the absolute path from one of the scripts 32c⟩ Referenced in 32b.
- ⟨filenames in nuweb compile script 61b⟩ Referenced in 60e.
- ⟨filenames in w2html 64a⟩ Referenced in 63d.
- ⟨find a spotlightserver or exit 24g⟩ Referenced in 52a.
- ⟨find the nlpp root directory 10d⟩ Not referenced.
- ⟨function to run a module 49a⟩ Referenced in 52ab.
- ⟨functions of the module-installer 36bc⟩ Referenced in 35d.
- ⟨get a testfile and set naflang or die 51⟩ Referenced in 52a.
- ⟨get commandline-arguments for check_start_spotlight 23a⟩ Referenced in 22b.
- ⟨get location of the script 53a⟩ Referenced in 7a, 8a, 10a, 16f, 22b, 35d, 37a, 38a, 44ac, 45c.
- ⟨get spotlight language parameters 25a⟩ Not referenced.
- ⟨impliciete make regels 58c, 63a⟩ Referenced in 55b.
- ⟨init make_infrastructure 8e, 11b⟩ Referenced in 7a, 8a.
- ⟨install ActivePython 16a⟩ Referenced in 15b.
- ⟨install Alpino 30a⟩ Referenced in 7a.
- ⟨install boost 35b⟩ Referenced in 7a.
- ⟨install libxml2 or libxslt 29b⟩ Referenced in 29c.
- ⟨install Perl 20b⟩ Referenced in 20a.
- ⟨install perl 20c, 21a⟩ Referenced in 19b.
- ⟨install shared libs 29c⟩ Referenced in 7a.
- ⟨install svmlib 35a⟩ Referenced in 7a.
- ⟨install the modules 38cfi, 39cfi, 40adg, 41c, 42adg, 43adg, 44e, 45ab, 46adg, 47a⟩ Referenced in 35d.
- ⟨install the Spotlight server 22a⟩ Referenced in 7a.
- ⟨install the ticcutils utility 33h⟩ Referenced in 7a, 34c.
- ⟨install the timbl utility 34a⟩ Referenced in 7a, 34c.
- ⟨install the treetagger utility 30c, 31bcde, 32a, 33ab⟩ Referenced in 7a.
- ⟨logmess 53c⟩ Not referenced.
- ⟨make scripts executable 7o, 8c, 17a, 24f, 36a, 47e, 65c⟩ Referenced in 65d.
- ⟨make targets 55d, 59b, 60a, 63c, 65bd, 66abc⟩ Referenced in 55b.
- ⟨make treetagger location-independent 32b⟩ Referenced in 32a.
- ⟨matchscript 32d⟩ Referenced in 32c.
- ⟨need to wget 11d⟩ Referenced in 13f, 18c, 28b, 33c.
- ⟨next part 7p⟩ Referenced in 7a.
- ⟨parameters in Makefile 55a, 56a, 58ab, 60c, 62b, 65a⟩ Referenced in 55b.

<perform the task of w2html 63e> Referenced in 63d.
 <re-install modules after the transplantation 34c> Not referenced.
 <remove the copy of the aux file 61c> Referenced in 61a, 63e.
 <replace the absolute paths 32e> Referenced in 32b.
 <rewrite ActivePython shabangs 16c> Referenced in 16a.
 <run in subshell when naflang is not known 48a> Not referenced.
 <run only if language is English or Dutch 48b> Not referenced.
 <run tex4ht 64d> Referenced in 64b.
 <run the html processors 64c> Referenced in 64b.
 <run the html processors until the aux file remains unchanged 64b> Referenced in 63e.
 <run the processors until the aux file remains unchanged 62a> Referenced in 61a.
 <run the three processors 61d> Referenced in 62a.
 <set default arguments for Spotlight 23b> Referenced in 22b.
 <set environment parameters 10c, 21b, 30b, 31a, 34d, 35c> Referenced in 10a.
 <set the naflang parameter 37c> Referenced in 37a, 45c.
 <set up autoconf 29a> Referenced in 7a.
 <set up Java 13a> Referenced in 7a.
 <set up java environment 13e> Referenced in 13a.
 <set up Maven 14b> Referenced in 7a.
 <set up Perl 19b> Not referenced.
 <set up Python 15b, 18ab> Referenced in 7a.
 <set variables that point to the directory-structure 10ef, 11a, 14f> Referenced in 10a.
 <start of module-script 38a> Not referenced.
 <start the Spotlight server on localhost 26b, 27a> Referenced in 24a, 25c.
 <test presence of command 8d> Referenced in 8e.
 <test whether spotlighthost runs 26a> Referenced in 25c.
 <try to obtain a running spotlightserver 25c> Not referenced.
 <unpack ticcutils or timbl 34b> Referenced in 33h, 34a.
 <variables of the module-installer 53b> Referenced in 35d.
 <wait until the spotlight server is up or faulty 27b> Referenced in 27a.

C.3 Variables

all: 55c.
 ALPINO_HOME: 30b.
 auxfil: 61b, 62a, 64a, 64b.
 bibtex: 61d, 64cd.
 DIRS: 65b, 65d.
 fig2dev: 58c.
 FIGFILENAMES: 58b.
 FIGFILES: 58a, 58b.
 indexfil: 61b, 62a, 64a.
 install: 8d, 13a, 14b, 15b, 16a, 18a, 19b, 20b, 29ab, 30a, 31b, 34b, 36b, 55e, 66a, 66b.
 makeindex: 61d, 64cd.
 MKDIR: 65a, 65b.
 moduleresult: 49a, 52ab.
 naflang: 23ab, 24g, 25a, 26b, 37c, 45c, 48ab, 50b, 51.
 nufil: 61b, 61d, 64a, 64c.
 nuweb: 10e, 55a, 56bcd, 60bce, 61a, 61d, 62b, 63d.
 oldaux: 61b, 61c, 62a, 64a, 64b.
 oldindexfil: 61b, 62a, 64a.
 opinion_models_ball_path: 46a.
 PATH: 11a, 13e, 14bf, 20c.
 pdf: 56a, 59b, 60a.
 PDFT_NAMES: 58b, 60a.
 PDF_FIG_NAMES: 58b, 60a.
 PHONY: 55c, 59a.
 piperoot: 10ab, 10d, 10e, 13e, 16ac, 20c, 29ab, 34b, 35a, 36b, 52a.

print: 17bc, 20a, 24g, 32d, 47d, 57a, 59b.
PST_NAMES: 58b.
PS_FIG_NAMES: 58b.
runmodule: 49a, 49b, 50a.
SUFFIXES: 56a.
SVMLIB_HOME: 34d, 35a.
testen: 66b.
testnl: 66b.
texfil: 61b, 61d, 64a, 64c.
TREETAGGER_HOME: 30c, 31a, 32ce.
trunk: 61b, 61d, 64a, 64cd.
view: 59b.