# Flexible NLP Pipelines for Digital Humanities Research

**Janneke M. van der Zwaan**
j.vanderzwaan@esciencecenter.nl
Netherlands eScience Center, The Netherlands

**Wouter Smink**
w.a.c.smink@utwente.nl
University of Twente, The Netherlands

**Anneke Sools**
a.m.sools@utwente.nl
University of Twente, The Netherlands

**Gerben Westerhof**
g.j.westerhof@utwente.nl
University of Twente, The Netherlands

**Bernard Veldkamp**
b.p.veldkamp@utwente.nl
University of Twente, The Netherlands

**Sytske Wiegersma**
s.wiegersma@utwente.nl
University of Twente, The Netherlands

## Introduction

A lot of Digital Humanities (DH) research involves applying Natural Language Processing (NLP) tasks, such as, sentiment analysis, named entity recognition, or topic modeling. A large amount of NLP software is already available. On the one hand, there are frameworks that bundle software for different tasks and languages (e.g., NLTK [Bird et al, 2009], or xtas1), and on the other hand there are tools that target specific tasks (e.g., gensim, Rehurek and Sojka, 2010). As long as researchers do not need to combine tools from different packages, it is usually relatively easy to write scripts that perform the task. However, for innovative research, combining tools often is required, especially when working with non-English text. This abstract presents work in progress on NLP Pipeline (**nlppln**), an open source tool that improves access to NLP software by facilitating combining NLP functionality from different software packages2.

**nlppln** is based on Common Workflow Language (CWL), a standard for describing data analysis workflows and tools (Amstutz et al, 2016). The main advantage of using a standard is that any existing NLP tool can be integrated into a workflow, as long as it can be run as a command line tool. This flexibility is missing from existing frameworks for creating NLP pipelines, such as DKPro (Eckart de Castilho, and Gurevych, 2015) using the UIMA framework (Ferrucci, and Lally, 2004). In addition to improved reuse of existing software, CWL increases research reproducibility, as it provides a standardized, formal record of all steps taken in a processing pipeline. Finally, CWL workflows are modular. This means that individual processing steps can easily be swapped in and out.

To demonstrate how NLP tools can be combined using nlppln, we show what need to be done to create a pipeline that removes named entities from a directory of text files. This is a common NLP task, that can be used as part of a data anonymization procedure.

## The Software

An NLP pipeline or workflow is a sequence of natural language processing steps. A 'step' represents a specific NLP task, that is executed by a single tool. Tools require input and produce output. The basic units in CWL are command line tools (i.e., tools that can be run from the command line). In order to be able to run a command line tool, CWL needs a specification. The **nlppln** software helps creating those specifications. In addition, **nlppln** provides functionality to convert existing NLP tools written in Python to command line tools. Finally, the software helps users to combine (existing and new) processing steps to pipelines.

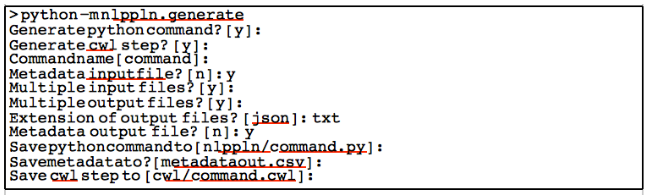
In the next section, we explain how **nlppln** facilitates creating NLP steps, and in "Constructing Pipelines" we demonstrate the creation of an NLP pipeline for data anonymization.

### Generating Steps

**nlppln** allows users to generate CWL specifications for existing NLP tools. To simplify the generation of CWL specifications, we use a convention for NLP tasks. The convention assumes that there can be two types of input parameters: a list of files for which the command should be executed, and/or a file containing metadata about the texts in the corpus. Output parameters consist of a directory where output files are stored (usually there is one output file for every input file) and/or a file in which metadata is stored. So far, almost all steps that are currently available in **nlppln** follow this convention. Be that as it may, we would like to emphasize that it is possible to deviate from this convention; for example, when existing NLP functionality requires different parameters (e.g., the name of a directory containing the input files instead of a list of input files). This does however mean that the user has to adapt the CWL specification by hand.

In addition to CWL specifications, **nlppln** allows users to generate boilerplate Python command line tools. A boilerplate command line tool contains generic functionality, such as opening input files and saving output files, but lacks implementation of the specific NLP task. The generated Python command can be used to turn existing NLP functionality into command line tools, and to create Python command line tools for new NLP tasks.

Python commands and associated CWL steps are generated using a command line tool that requires the user to answer a sequence of yes/no questions. Listing 1 shows what that looks like for a (hypothetical) command 'command', that takes as input a metadata file and multiple input files, and produces as output multiple text files and metadata.



```
>python-mnlppln.generate
Generatepythoncommand?[y]:
Generate cwl step? [y]:
Commandname[command]:
Metadata inputfile?[n]:y
Multiple input files?[y]:
Multipleoutput files?[y]:
Extension of output files?[json]:txt
Metadata output file?[n]:y
Savepythoncommandto[nlppln/command.py]:
Savemetadatato?[metadataout.csv]:
Save cwl step to [cwl/command.cwl]:
```

Listing 1: Generating a CWL specification and associated boilerplate Python command using **nlppln.**

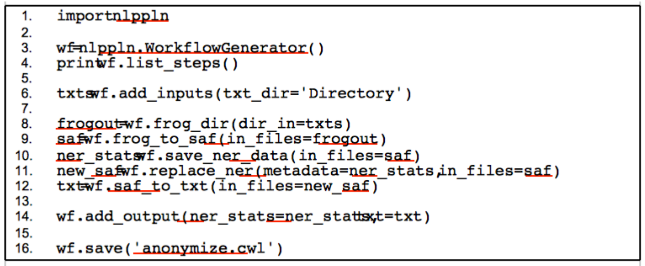
## Constructing Pipelines

To combine text processing steps into a CWL pipeline, **nlppln** provides an interface that allows users to write a simple Python script. We demonstrate this functionality by creating a pipeline that replaces named entities in a collection of text documents. Named entities are objects in text referred to by proper names, such as persons, organizations, and locations. In the example pipeline, named entities will be replaced with their named entity type (i.e., PER (person), ORG (organization), LOC (location), or UNSP (unspecified)). The pipeline can be used as part of a data anonymization procedure.

The pipeline consists of the following steps:

1. Extract named entities from text documents using frog (van den Bosch et al, 2007), an existing parser/tagger for Dutch
2. Convert frog output to SAF, a generic representation for text data3
3. Aggregate data about named entities that occur in the text files
4. Replace named entities with their named entity type in the SAF documents
5. Convert SAF documents to text

All steps required for this pipeline are available through **nlppln**. Listing 2 shows the script that creates a CWL workflow for this pipeline. After importing **nlppln** (line 1), a new WorkflowGenerator object is created (line 3), and the available NLP steps are listed (line 4). Next, the script specifies the workflow inputs (line 6). In this case, there is a single input, which is a directory containing text files. This directory is the input of the first step, which is frog_dir (line 8). The output argument txts contains the internal CWL name of the input parameter (line 6). By assigning its value to the input argument dir_in of frog_dir (line 8), the output is connected to the input. Steps 1 to 5 from the pipeline description correspond to lines 8 to 12 in listing 2. After the remaining steps steps of the workflow are added (lines 9–12), the workflow outputs are specified (line 14). Finally, the workflow is saved to a CWL file (line 16).



```
1.   importnlppln
2.
3.   wf=nlppln.WorkflowGenerator()
4.   printwf.list_steps()
5.
6.   txts=wf.add_inputs(txt_dir='Directory')
7.
8.   frogout=wf.frog_dir(dir_in=txts)
9.   saf=wf.frog_to_saf(in_files=frogout)
10.  ner_stats=wf.save_ner_data(in_files=saf)
11.  new_saf=wf.replace_ner(metadata=ner_stats,in_files=saf)
12.  txt=wf.saf_to_txt(in_files=new_saf)
13.
14.  wf.add_output(ner_stats=ner_stats,txt=txt)
15.
16.  wf.save('anonymize.cwl')
```

Listing 2: Python script for constructing the pipeline to replace named entities in text files.

## Conclusion

To help DH researchers to (re)use and combine existing NLP software, we presented **nlppln**, an open source Python package for creating flexible and reusable NLP pipelines in CWL. nlppln comes with ready-to-use NLP steps, facilitates creating new steps, and helps combining steps into standardized workflows that are portable across different software and hardware environments. Compared to existing frameworks for creating NLP pipelines, CWL and

**nlppln** add flexibility and improved research reproducibility.

**nlppln** is a work in progress. An important challenge that needs to be addressed is the fact that there is no standard data format for representing text and/or information extracted from text. This means that we will have to add NLP steps that convert different data formats (cf. Eckart de Castilho, 2016)). For future work, we plan to implement additional NLP steps and pipelines, including functionality that targets more languages. We would also like to add visualizations of pipelines and allow users to run pipelines directly from **nlppln**.

## Bibliography

**Amstutz, P., Crusoe, M. R., Tijanić, N., Chapman, B., Chilton, J., Heuer, M., Kartashov, A., Leehr, D., Ménager, H., Nedeljkovich, M., Scales, M., Soiland-Reyes, S., and Stojanovic, L.** (2016). *Common Workflow Language, v1.0,*.

**Bird, S., Loper, E., and Klein, E.** (2009) *Natural Language Processing with Python*. O'Reilly Media Inc.

**van den Bosch, A., B Busser, B., Dealemans, G. J., and Canisius, S.** (2007) An efficient memory-based morphosyntactic tagger and parser for Dutch. In *Proceedings of the 17th Meeting of Computational Linguistics in the Netherlands*, pages 191–206, 2007.

**Eckart de Castilho, R.** (2016). Interoperability = f(community, division of labour). In *Proceedings of the Workshop on Cross-Platform Text Mining and Natural Language Processing Interoperability (INTEROP 2016) at LREC 2016*, pages 24–28, 2016.

**Eckart de Castilho, R., and Gurevych, I. (**2014). A broad-coverage collection of portable NLP components for building shareable analysis pipelines. In *Proceedings of the Workshop on Open Infrastructures and Analysis Frameworks for HLT (OIAF4HLT) at COLING 2014,* pages 1–11.

**Ferrucci, D., and Lally., A.** (2004) UIMA: an architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering* 10.3-4, pages 327–348.

**Rehurek, R., and Sojka, P.** (2010). Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50.