Web As Corpus Toolkit
User's and Hacker's Manual

Ramon Ziai & Niels Ott

November 1st, 2005

Originally written for Lexical Computing Ltd.
See `http://www.drni.de/wac-tk/` for further information on the project.

Contents

1 Introduction

The concept of the *Web As Corpus Toolkit* is to create a clean corpus from a list of URLs. Web pages are fetched, filtered, and finally the collection is converted to a single vertical file that serves as input for Word Sketch Engine and its web based user interface.

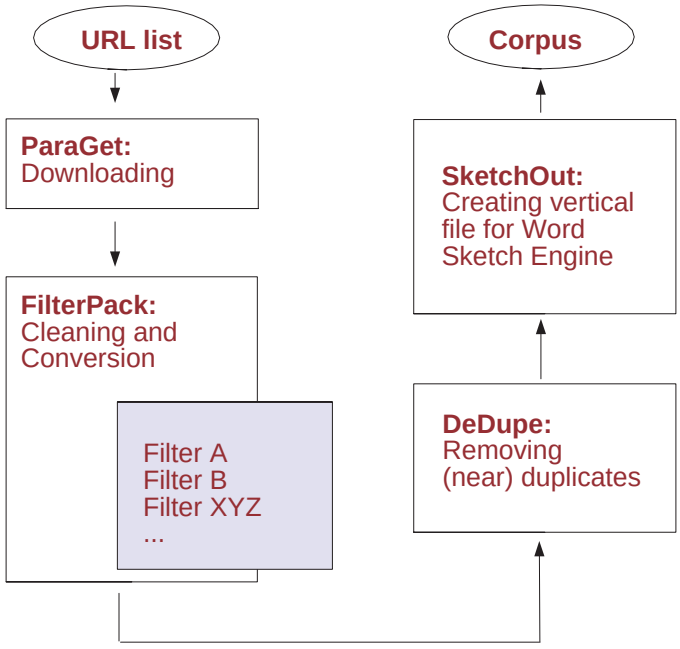

Figure 1: Processing steps from URL list to Corpus

In order to make those things happen, several steps of processing are required. Each step has a tool of its own. (See figure 1.) They are as follows:

1. *ParaGet* – Downloads all URLs from the list using parallel network connections.
2. *FilterPack* – Runs a bunch of filter modules on each document to get everything nice an clean.
3. *DeDupe* – Compares all documents to find duplicates and near-duplicates.
4. *SketchOut* – Collects all documents and converts them to one single vertical file as input for Word Sketch Engine.

Usage and configuration are described in detail in the sections below. If you just want to get it going, check out section 2.

2 Quickstart and Installation Guide

This is a guide for the impatient, containing only what you really need to know to run the whole thing. Reading the other sections is recommended, but not strictly necessary. The following assumes that you have already extracted the tarball into some directory, which we'll call *rootdir*.

2.1 Requirements

To run the *WaC Tools*, you will need to install a number of Perl modules not included in the standard distribution. To find out which modules your system is lacking, run the following command from the *rootdir*:

```
./modules
```

This will give you a list of modules you need to install – or if you're lucky it will just say that everything is alright. All these modules are available from CPAN[1] and installed in the following way:

Start CPAN shell:
`perl -MCPAN -e shell`
Do this for every module:
`install` *modulename*

Setting up CPAN is not covered here, see `http://www.cpan.org`

2.2 Configuration

You will at least need to change the following settings in the respective configuration files (located in *rootdir*`/conf`):

2.2.1 paraget.conf

`list`

The list of URLs you want to download. The file has to contain one URL per line.

`target`

The destination directory where all the files will be stored. Make sure you have write access.

2.2.2 filterpack.conf

`source`

The directory where the downloaded files were stored (`target` from above).

2.2.3 dedupe.conf

`source`

The directory where the downloaded and the filtered files were stored (`target` from above).

[1] Except for HTML::Strip::StripX which comes along with the *WaC Toolkit*. If this module fails, install HTML::Strip from CPAN first.

2.2.4 sketchout.conf

`source`

The directory where the downloaded and the filtered files were stored (`target` from above).

`vertical`

The vertical file that should be written, full path name required. Make sure you have write access.

2.3 Running the stuff

Now you're ready to run the various parts. Commands should be run in this order (Logs will be written to *rootdir*`/logs`):

```
./paraget
./filterpack
./dedupe
./sketchout
```

Depending on the size of your URL list, each step may take quite some time. If everything goes well, you find your corpus in the place you specified.

3 Computing Resources

The *WaC Toolkit* can require quite some resources from your machine. In development, approx. 82.000 URLs were downloaded. This required 5GB of disk space for the data directory. The final vertical file will be a lot smaller, yet POS tagging and lemmatization may increase size again.

There is no specific requirement for CPU power. Users will enjoy fast machines, though. As *FilterPack* is the most power consuming tool, it runs parallel. This way it can seize the power of multi-processor machines. For the example above, a 2x3Ghz machine was used and found useful.

Memory consumption is low. *DeDupe* is the tool with the most consumption and for the example above the worst case scenario for it would be to use approx. 300MB of RAM. For our modern world of computing, that's just peanuts, isn't it? If you are unsure about your system working properly with the *WaC Toolkit*, we suggest monitoring with `top` and `xosview`.

You will need at least Perl 5.8 to run the toolkit. Everything older lacks good Unicode support and may also fail to run parallel processes.

4 Configuration Files

Configuration files for all tools are stored in *rootdir*`/conf`. Their format is the typical one on Un*x machines: Pairs of names and values, number-signs denoting comments. There are no special escape characters. Everything to the right of the equal sign is taken as is. You find detailed documentation on each config file in the corresponding section of the tool.

5 Directory Structure and Data Files

5.1 Directories

The toolkit will work using a special directory structure. You do not have to create this structure. *ParaGet* will do it for you before it downloads from the web pages. You will find a directory for each top level domain that is in the URL list. Unclassified domains (like IP numbers) will end up in `xxx`. In each of those directories you will find the first letters of all server names, ignoring leading `www` sequences. The files in the directories are named after their document IDs. See sections 5.2.1 and 5.2.2

5.2 Data Files

5.2.1 Document IDs

Document IDs are created by *ParaGet* before downloading. They simply consist of the string `net-` plus a counter given as hexadecimal number. Document IDs are meant to be unique in your corpus!

5.2.2 File Formats

ParaGet will store data in files ending in `.dl`. The files contain two header parts and one body part. The first header part is the "private" data of the Toolkit while the second header is a mix of the original HTTP header returned by the web server and several interpretations done by simplistic HTML parsing.

After downloading, the first header will be quite small: You will find the URL where the document came from plus the document ID.

FilterPack will read the `*.dl` files and create `*.fil` equivalents with much more header information: Each filter module stores a message there, and the filter dependencies system does so, too. Both *FilterPack* and *DeDupe* will insert a header reporting on their success or failure. Further processing steps may rely on these headers.

The body part consists of the original data in the `*.dl` files. In `*.fil` files it might look very different. You will find the data as it looks like after all filter modules did process it. Several filter modules may introduce new tags to the body data. We call them *WaC tags*. These tags all are from the "wac" namespace and look like `<wac:p>`. Additionally the filters will keep track of those tags in the `Protected-Tags` header field. The `wac:` namespace part is removed by *SketchOut* and will not appear in the vertical file. For details on this topic see section 11.1.3.

6 Log Files

All log files are meant to be human readable and self explanatory. Therefore entries starting with a plus sign donate success of an action, those starting with a minus sign indicate failure.

Most kinds of log file analysis can be done by simple Un*x tools like `grep` and `wc -l`. They allow to built statistics that the Toolkit cannot or does not come up with. Note that in the current release, output lines might get mixed together as log output is currently not parallelized in a clean way.

7 ParaGet: The Downloader

7.1 Files

Startup command:	`paraget`
Configuration:	`conf/paraget.conf`
Messages go to:	`log/paraget.log`

7.2 Introduction

ParaGet will download your URL list. Additionally it takes over the task to generate document IDs. These IDs will accompany you and your tools throughout the whole process of creating your corpus. For details on directory structure and file formats, see sections 5.1 and 5.2.

ParaGet includes a first step of filtering, namely it does filter out documents which are too large or too small. URLs will be randomized before downloading starts in order to avoid "bombing" of single servers.

7.3 Configuration

The configuration file contains the following directives:

`list`

This specifies where to find the list of URLs you want to download. The file has to contain one URL per line.

`children`

The number of child processes *ParaGet* will create. Each process will open one network connection at a time, so this is also the number of connections the program will hold simultaneously.

`timeout`

Some servers will not answer to the requests of the tool. They will just remain silent. After `timeout` seconds, *ParaGet* will give up.

`target`

This is where the output goes to. The tool will create a number of subdirectories from the domain names in the input list. All URLs will end up as single documents in those subdirectories. Make sure that you have write permission.

`minsize`

Minimum document size in kilobytes. Set to -1 to disable the filter.

`maxsize`

Maximum document size in kilobytes. Set to -1 to disable this filter. Disabling is not recommended! You might download very large but useless files, e.g. ISO CD images.

7.4 Messages

The messages in the log file are self explanatory. As usual in this project, "good" messages start with a plus and "bad" ones with a minus. *ParaGet* outputs the HTTP response codes as well. There is one exception: Do not get confused by response codes over 900. Some codes are customized a bit:

`500` While officially a server error, the implementation of the underlying downloader uses this for timed out requests as well.
`901` The file is smaller than the minimum size required.
`902` The file is larger than the maximum size limit.

For more information on log files, refer to section 6.

8 FilterPack: All The Filtering

8.1 Files

Startup command:	`filterpack`
Configuration:	`conf/filterpack.conf`
Messages go to:	`log/filterpack.log`

8.2 Introduction

This program deals with the various filters implemented. After reading in the downloaded files, *FilterPack* automatically searches its config file for filters that should be applied. Each will be described shortly here and the configuration options explained. It is important to know that filters are applied in the order specified in the config file. If a filter returns failure, filtering stops and the file is ruled out. After *FilterPack* has applied all filters to a specific document or discarded it, it stores the filtered document in a `.fil` file.

8.3 General Configuration

This is what you can set in the configuration file:

`source`

This specifies where to find the downloaded files.

`children`

Specifies the number of child processes for parallel processing. Rule of thumb: Number of CPUs + 1

`savefailures`

Indicates whether you want to save files rejected by *FilterPack* or not. Possible values are 0 for "No" and 1 for "Yes".

`filters`

The comma-separated list of filters *FilterPack* will apply to each file. Order does matter!

Additionally you will find entries that by convention start with "fil.". These are used by the filter modules. They are described in section 8.5.

8.4 Dependencies

FilterPack modules can require module keywords that are in turn provided by other *FilterPack* modules. For example, *boilerplate* requires `cleaning`, which is provided by *plaintext-cleaner* and in turn provides `no-boilerplates`. This model ensures that filters don't get applied in the very wrong order and you don't end up with something very messy.

8.5 The Filters

8.5.1 content-type

Tries to determine the content type of the file and rules out all non-HTML stuff. Useful if you're not sure your URL list is HTML-only. Returns failure if content type is not `text/html`, success otherwise.

8.5.2 plaintext-cleaner

Cleans up superfluous whitespace and line breaks, also converts the latter to Un*x-style. Always returns success.

8.5.3 find-title

Finds out the title of the document, makes an attempt to get it from the document itself if no `<title>` tag is found. Returns failure if no title could be found, success otherwise.

8.5.4 insert-wactags

This module finds certain structures in HTML using a parser and inserts the *WaC tags* `<wac:heading>` and `</wac:heading>` (see section 11.1.3). Currently the code is in an alpha stadium and heavily conflicting with the boilerplate module.

8.5.5 boilerplate

Removes boilerplate (high tag density) areas from the document. Always returns success.

8.5.6 html-strip

Removes remaining HTML markup using the HTML::Strip module. Returns failure if HTML::Strip::StripX fails or its output is empty, success otherwise.

8.5.7 to-unicode

Converts document to Unicode if necessary. Guesses encoding if none is specified in the HTML header. Returns failure if the decoder chokes on the input, success otherwise.

Configuration options:

`fil.to-unicode.candidates`

The list of candidates to guess from if no encoding is found in the HTML header.

`fil.to-unicode.alwaysguess`

Should the module always guess instead of using the provided encoding? 0 for "No", 1 for "Yes".

8.5.8 de-entity

Converts all HTML entities to plain text (Unicode). Always returns success.

8.5.9 doctor-unicode

Tries to repair conversion accidents that occurred earlier. Always returns success.

8.5.10 non-text

Removes non-text ("ASCII furniture") from the document, such as vertical bars, sequences of hyphens, and other weird things. Always returns success.

8.5.11 size-matters

Rejects documents smaller than a given size. Returns failure if document is smaller than the threshold specified, success otherwise.

Configuration options:

`fil.size-matters.minsize`

Minimal document size in bytes.

8.5.12 paragraphs

Inserts paragraph *WaC tags* `<wac:p>` and `</wac:p>` into the document, using newline information and heading tags (if available). Always returns success.

8.5.13 sentences-plus

Inserts the sentence *WaC tags* `<wac:s>` and `</wac:s>` to mark the beginning and end of a sentence. Always returns success.

8.5.14 string-tokenizer

Tokenizes document using String::Tokenizer. Returns failure if the String::Tokenizer module fails, success otherwise.

Configuration options (mandatory):

`fil.string-tokenizer.delim_set`

The delimiter set used by the tokenizer as a single string.

`fil.string-tokenizer.insertglue`

Determines whether to insert glue tags (<g/>) between punctuation characters and words or not. 1 for "Yes", 0 for "No".

8.5.15 british-american

Classifies document language as British, American or Unknown. Always returns success.

8.5.16 generic-shell-filter

Provides the possibility to run any shell command as a filter. Returns failure if the command fails for some reason, success otherwise. **Warning:** This module is dangerous. Do not use. At the current stage of development, it is not parallelization safe! (See also: Sections 3 and 11.1.4.)

Configuration options:

`fil.generic-shell-filter.command`

The command you want to run.

`fil.generic-shell-filter.tmpfile`

Temporary file used for the output of the shell command.

8.5.17 passthru

This module does exactly nothing to your data. It can be used as reference implementation of a module. For module hacking, see section 11.1.

8.6 Messages

FilterPack uses the following "message tags": `START` for startup messages, `PREP` for preparation procedures, `FIL` for filter information and `FINAL` for final summary information. Additionally, it uses `READ` when reading files, `WRITE` when writing files and `CLOCK` for time information. As with the other programs, "-" indicates a negative and "+" indicates a positive log entry.

9 DeDupe: Removing (Near-)Duplicate Documents

9.1 Files

Startup command:	`dedupe`
Configuration:	`conf/dedupe.conf`
Messages go to:	`log/dedupe.log`

9.2 Introduction

DeDupe deals with detecting and removing (near) duplicate documents. During the process, a database of sentences is built up. If the proportion of duplicate sentences in a document is too hight, it gets dropped. *DeDupe* relies on *FilterPack* output, so make sure you ran *FilterPack* before.

9.3 Configuration

The following parameters are configurable:

`source`

Specifies the directory that contains the filtered documents.

`proportion`

Specifies the threshold (in percent) of allowed duplication in documents.

9.4 Messages

DeDupe uses the usual +/- log messages of this project. `PROC` is used to indicate whether a file will be considered or not, `FIL` shows whether a document is ruled out by *DeDupe* or not. In the end, a short summary is given, containing the total number of files considered and the number of files dropped due to duplication.

10 SketchOut: Creating Input for Word Sketch Engine

10.1 Files

Startup command:	`sketchout`
Configuration:	`conf/sketchout.conf`
Messages go to:	`log/sketchout.log`

10.2 Introduction

SketchOut will collect all your `.fil` files that were created by *FilterPack* and probably modified by *DeDupe*. It will create one single vertical file that servers as input for Word Sketch Engine.

Additionally, *SketchOut* is able to run an external command as a filter. Your data will be piped through this command before it is stored. This can be used e.g. for POS tagging in the same step. The tool will ignore files that were not accepted by *FilterPack*. If header information from *DeDupe* is present, it will be considered as well.

10.3 Header Fields

SketchOut will create `<doc>` tags in its output with the following arguments:

- `id` – The document ID
- `url` – The origin of the document
- `title` – Title of the web page or "n/a" if none given
- `author` – Author(s) of the web page or "n/a" if none given
- `lang` – Language information or "n/a" if none given

10.4 Configuration

The configuration file contains the following directives:

`source`

Where to find your directory tree with the input data (`.fil` files).

`vertical`

Path to the output file. This file will contain the vertical text. Make sure you have write permission.

`pipefilter`

An external command that reads data from `STDIN` and writes to `STDOUT`. Your data will be piped through this filter before it gets stored into the vertical file. Specify `-` to disable this feature.

10.5 Messages

The messages in the log file are self explanatory. As usual in this project, "good" messages start with a plus and "bad" ones with a minus. For more information on log files, refer to section 6.

11 Hack It Yourself

11.1 FilterPack Module Interface Specification

FilterPack maybe is the most important element of the *WaC Toolkit*. It gives you the power to plug in and out filters as you wish (unless dependencies are not violated). In case you want to extend the functionality of the package, it is most likely that a new filter is the point to start from. To make things easier, this section describes how to write a filter.

While reading this section, you should open the file *rootdir*`/fp-filters/passthru.pm` for reference. The *passthru* module does exactly nothing to the data. Its main purpose is to show the plain interface.

11.1.1 The "filterinfo" Subroutine

When your filter module is initialized, *FilterPack* calls the subroutine `filterinfo`. This subroutine has to implement one very basic thing: It has to *return a reference* to an unique subroutine that does the actual job of the filter (See section 11.1.2).

Your `filterinfo` subroutine will make use of several arguments. They are all passed by reference and some need to be changed!

`text_p` **(Arg. #1)**

This is a reference to a string. You are supposed to assign a text message to it that says what the filter is doing. Specify e.g. "Identifies the sex of the author".

`CONFp` **(Arg. #2)**

This is a reference to a hash. The hash contains the key/value pairs from the file

rootdir/conf/filterpack.conf. This way, your filter can make use of the global configuration system of *FilterPack*. The convention is to use keys like "fil.*filtername*.something". Although the configuration can be modified, changes will be discarded for safety reasons.

PROVp **(Arg. #3)**

This is a reference to a string. It is used for the dependencies system (see section 8.4). Specify zero or more keywords that your module *provides*, separated with a single blank space.

REQp **(Arg. #4)**

This is a reference to a string. It is used for the dependencies system (see section 8.4). Specify zero or more keywords that your module *requires*, separated with a single blank space.

For the dependency system, no further action but specifying the keywords has to be done. *FilterPack* will do the rest for you.

11.1.2 The Filter Action Subroutine

Now this is where the magic happens! But first of all: Make sure that the name of this subroutine is unique throughout the whole code you can imagine of. This is a bit hacky but it works. We recommend to use subroutine names like "*filtername*_filteraction".

Your subroutine has to *return 0 or 1* for failure or success. If you do return 0, *FilterPack* will stop any processing of the current document. The document will be excluded by *DeDupe* and *SketchOut* so it is not present in the final corpus.

Like the previous subroutine (section 11.1.1), the filter action subroutine is supplied with several arguments which are references to different data structures.

H1p **(Arg. #1)**

This is a reference to a hash. It contains the first header of the document as key/value pairs. You can e.g. access the document ID as `$H1p->{'DocumentID'}`. You can overwrite or introduce header fields. Be careful. For more information on the header system, consult section 5.2.2.

H2p **(Arg. #2)**

This is a reference to a hash. It contains key/value pairs of the original HTTP header plus information from simplistic parsing. The same things hold as for H1p.

BOp **(Arg. #3)**

This is a reference to a string. It contains the body part of the document. Do anything you like with it.

MSGp **(Arg. #4)**

This is a reference to a string. You are supposed to specify a message that indicates what your filter did and why it possibly returned 0. You could e.g. specify "Identified author to be male" or "Author is Klingon, excluding document" etc.

CONFp **(Arg. #5)**

This is a reference to a hash. See section 11.1.1.
The message your filter returns will be stored to the `*.fil` file automatically as a field of the first header. Additionally it will be sent to the log file of *FilterPack*.

11.1.3 Taking Care of the WaC Tags

FilterPack includes a system of transporting meta information within the data. This requires your attention as those *WaC tags* must not be destroyed by your filter. And your filter should of course not be disturbed by their presence.

The means by which the tags are protected is basically the *Protected-Tags* header field. To make your life a bit easier, you don't have to work directly on this field. A library does it for you. If you do introduce a new tag, do like this:

```
require('tag-protector.pm');
...
tagprotect_addtag($H1p, '<wac:x/>');
...
```

Assuming `$H1p` being the reference to the header and `<wac:x/>` the tag you want to introduce. All filter modules will then take care of your *WaC tag* and will leave it as is.

To make things work without problems, your filter module must of course act the same way. Nobody cares how your module does it but it must leave those tags alone and in situ. Again, the library provides a function that tells you the tags you have to take care of:

```
require('tag-protector.pm');
...
my @tags = tagprotect_gettags($H1p);
...
```

Having all the *WaC tags* stored in the list `@tags` you can work around them.

11.1.4 Keep in mind: Parallelization

Keep in mind that *FilterPack* runs with parallel processes. If you are using e.g. temporary files, you may get into deep trouble if several instances of your filter write to the same file at the same time.