An overview of the Alpino Treebank tools

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Introduction

In this document the various tools of the Alpino Treebank will be discussed. These tools include:

dtsearch
: find dependency structures using XPath queries

dtview
: display dependency structures

dtedit
: edit dependency structures with Thistle (obsolete)

dttred
: edit dependency structures with TrEd

dtxslt
: apply style sheet to dependency structures

Most of the Alpino Treebank tools support two types of annotated corpora. In the simple case, each dependency structure is represented by a single XML file. In addition, the tools support so-called *compact* annotated corpora, in which collections of XML files are concatenated and compressed. The following tools are relevant for compact corpora in particular:

act
: construct/extract compact annotated corpora

dtlist
: list file names of dependency structures

dtgrep
: search in dependency structures with grep

dtget
: write the specified dependency structure to standard output

Some tools are not yet documented:

mg_m_search
: use dtsearch on set of dependency structures returned by an MG query

wrappers and libraries for xquery
: tools to apply Xquery to Alpino dependency structures (normal and compact)

Chapter 2

DTSearch: find dependency structures using XPath queries

Use dtsearch to search the corpus. dtsearch uses XPath expressions for querying.

A basic example is:

```
$ dtsearch -v '//node[@cat="pp"]' /path/to/corpus
```

By using the `-v` option dtsearch will display the files that match the query using dtview. In this case, the XPath expression matches with any dependency structure which contains a node with category *pp*.

DTSearch will process directories recursively.

The usage information lists the possible output methods:

```
usage: dtsearch [options] <[-q] QUERY> <files, directories, ...>

options:
  -qQUERY, -eQUERY, --query=QUERY, --expr=QUERY
                        XPath expression to search for
  -l, --statistics      print stats for values of rel,cat,pos labels in
                        matching nodes
  -r, --root-labels     print stats for values of root label in matching nodes
  -s, --bracketed-sentence
                        show sentences with matching phrases
  -c, --matching-constituents
                        show matching constituents only, not the full sentence.
  -d, -v, --dtview      show matches using dtview.
  --stdin               read file arguments from stdin.  Any non-option
                        arguments on the commandline will be ignored.
  -h, --help            show this help message and exit
```

When no output method is selected dtsearch will print the filename.

2.1 Examples of using dtsearch

In order to be able to specify relevant XPath queries, it is neccessary to know a little bit about the way in which dependency structures are encoded in XML. This encoding is rather straightforward. Nodes in the dependency structure are encoded by a recursive XML element *node*. Nodes contain a variety of attributes. The most important attributes are:

cat
: The category of the node (only for non-leaves)

pos
The POS-tag of the node (only for leaves)

begin
The begin position of the node

end
The end position of the node

rel
The dependency relation

root
The root form of the node (leaves)

word
The surface form of the node (leaves)

index
The index of the node (for *secondary edges* or reentrancies)

There are more attributes, but these are the most important ones.

The following query lists all dependency structures which contain a subject node which itself contains a node with the root form *man*. Because of the -s option, all matching sentences are displayed, where the matched part of the sentence is given in square brackets.

```
$ dtsearch -s '//node[@rel="su" and node[@root="man"]]' g_suite

g_suite/192.xml [de man] zou op zijn te bellen
g_suite/193.xml [de man] zou zijn op te bellen
g_suite/194.xml dat [de man] zou zijn op te bellen
g_suite/195.xml dat [de man] zou op zijn te bellen
g_suite/196.xml dat [de man] op te bellen zou zijn
g_suite/298.xml [de man] is bang dat hij naar huis moet
g_suite/518.xml [de man] werd in de rede gevallen
```

Further examples are perhaps hard to understand without some knowledge of XPath. There are various tutorials and reference manuals for XPath available on the web. See for example: What is XPath?

List all sentences containing a "krijgen-passive" in the h_suite treebank:

```
$ dtsearch -s '//node[ node[@rel="hd" and @root="krijg"] and \
    node[@rel="su"]/@index=node[@rel="vc"]/node[@rel="obj2"]/@index ]' h_suite

h_suite/277.xml Ik krijg doorbetaald
h_suite/296.xml hij krijgt een microfoon onder de neus geduwd
h_suite/300.xml De ontslagen medewerkers krijgen tot 1 januari 2004 doorbetaald
h_suite/301.xml De ontslagen medewerkers krijgen hun salaris tot 1 januari 2004 doorbetaald
h_suite/306.xml Hij krijgt betaald voor zijn adviezen
h_suite/307.xml Hij krijgt een fortuin betaald voor zijn adviezen
h_suite/556.xml Je kreeg met de paplepel ingegoten dat je beleefd moest zijn
```

Sometimes it is convenient to get the matched portion of the sentence and nothing more. This is accomplished with the `-c` option. For instance, the following finds determiner phrases which contain an adverb as a daughter node.

```
$ dtsearch -c '//node[@cat="detp" and node[@pos="adv"]]' leuven_yellow_pages
leuven_yellow_pages/244.xml     iets meer dan bij ons
leuven_yellow_pages/275.xml     wat meer
leuven_yellow_pages/276.xml     zoveel meer
```

Quantitative information can be obtained with the `-l` option. For instance, this query finds which relations occur as sisters to the *tag* relation:

```
$ dtsearch -l '//node[ ../node[ @rel="tag"]]' cdb
rel:
     295 tag
     291 nucl
cat:
     228 smain
     104 sv1
      56 pp
      40 du
      30 conj
      27 ssub
      12 mwu
      10 np
       4 whq
       1 inf
       1 detp
       1 cp
       1 ap
pos:
      28 adv
      20 tag
       8 noun
       8 adj
       3 verb
       3 num
       1 comp
```

More examples, for the interested reader to try out:

Find topicalized secondary objects with category NP:

```
$ dtsearch -v '//node[../@cat="smain" and @rel="obj2" and \
                 not(@cat="pp") and @begin = ../@begin]' .
```

Find occurrences of extraposition of comparatives out of topicicalized constituents:

```
$ dtsearch -v '//node[@cat="smain" and node[node[@rel="obcomp"]/@end\
                > ../node[@rel="hd"]/@begin]/@begin = @begin]' .
```

It is also possible to search in the text of the sentence. To do this efficiently, we have developed mg_m_search, that should be used in such cases for very large treebanks. Searching for text can be done as follows:

```
$ dtsearch -v 'contains(//sentence,"tot zo")' .
```

This works, because there is atmost one sentence element - the string value of a node set in XPath is defined as the string value of the first element of that node set.

In addition to "contains()" there are various other XPATH functions that are useful in this context, such as "starts-with()" and "matches()". The latter is used for regular expression matching in XPath 2 (currently not supported).

```
$ dtsearch  '(//comment[. = "time_out" ] or //comment[. ="out_of_memory" ]' .
```

In this case, the query will also match in case there are multiple comments (these are indeed allowed).

Beware of shell quoting

Make sure you keep the shell (your command interpreter) from interpreting any special characters in the query. Use any of the following schemes:

```
dtsearch '//node[@cat="pp"]'  ...
dtsearch "//node[@cat='pp']"  ...
dtsearch "//node[@cat=\"pp\"]"  ...
```

The first two are the most convenient ones to type.

Chapter 3

DTView: graphical display of dependency structures

Use dtview to visualize Alpino Dependency structures on your screen.

Screenshot of DTView vannoord/alp/Alpino/dtview.png vannoord/alp/Alpino/dtview.png Screenshot of DTView

3.1 Basic Functionality

DTView can visualize dependency structures and highlight nodes that match an XPath expression. This expression can be specified on the commandline or in the text entry at the top of the window.

Apply
Apply the XPath expression for highlighting to the current tree. (This can also be achieved by pressing Enter in the text field)

Filelist
Toggle the visibility of the file list

Ext'd Attrs
Toggle the visibility of any extended attributes that might be present. In general trees that are not hand-corrected contain extra attributes.

Previous, Next
Go to the previous or next file in the file list.

Smaller, Bigger, Normal
Change the size of the displayed tree.

TrEd
Run the TrEd tree editor on the current file.

Thistle
Run the Thistle tree editor on the current file. (Obsolete, use TrEd instead)

Emacs
Run the Emacs text editor on the current file.

Open Selection
Your current X-selection is supposed to contain a file name. This file name is opened.

Quit
Exit the viewer

3.2 Keyboard navigation

Table 3.1: Keyboard shortcuts for DTView

Key(s)	**Description**
Control-q	exit
n, Page Down	next file
p, Page Up	previous file
Up	scroll canvas
Down	scroll canvas
Left	scroll canvas
Right	scroll canvas
e	toggle Extended Attributes
f	toggle Filelist
t	invoke TrEd editor
Keypad_Add	zoom in
Keypad_Subtract	zoom out
Keypad_Multiply	revert to original size

Chapter 4

DTXslt: Running stylesheets on a corpus

The dtxslt tool can be used in a similar fashion as programs such as *xsltproc* to apply a stylesheet to dependency structures.

```
Usage: dtxslt [options] <files, directories, ...>
  -s, --stylesheet=STYLESHEET      The styleheet to use for output.
  -q, --query=QUERY                XPath-expression to be used as query.  The
                                   expression should evaluate to a node set or
                                   a boolean.  With this option the stylesheet
                                   is only applied to documents matching QUERY.
  --param=<name>=<value>           Normal parameter for stylesheet
  --stringparam=<name>=<value>     String-parameter for stylesheet
  --stdin                          Read the arguments from standard input, one
                                   argument per line.  When this option is
                                   used, any files or directories specified on
                                   the command line will be discarded.
  -r, --recursive                  Process the directory tree recursively

Help options:
  -?, --help                       Show this help message
  --usage                          Display brief usage message
```

An example:

```
$ dtxslt -r -s stylesheets/print-sentence.xsl Machine/clef
```

This example applies the `print-sentence.xsl` stylesheet to every .xml file under `Machine/clef`. The `-r` flag ensures `dtxslt` will recursively walk through the directory structure looking for .xml files and compact corpora.

Look for examples of stylesheets in `Alpino/TreebankTools/stylesheets`. Another place that shows the use of stylesheets in the Alpino Treebank is the `create-sanity-check-stylesheet.py` script in the `misc-scripts` directory.

Chapter 5

DTTred: Editing dependency structures with TrEd

The `dttred` program is a simple script which starts the TrEd tree editor, with the appropriate settings for Alpino dependency structures. The program normally takes one or more arguments: the file names you want to edit.

Alpino-specific functionality can be found under the User-defined→Alpino menu.

Documentation for TrEd

Chapter 6

DTEdit: Editing dependency structures with Thistle

The `dtedit` program is a simple script which starts the Thistle editor, with the appropriate settings for Alpino dependency structures. The program normally takes one or more arguments: the file names you want to edit.

Please use the documentation of Thistle in the local installation directory of Thistle on your machine.

DTEdit is now obsolete. Please use `dttred` instead.

Compact annotated corpora are not supported by dtedit and dttred
If you want to edit a dependency structure from a compact annotated corpus, you must first extract the file (using ACT), use dtedit or dttred, and then pack the file back into the archive (again using ACT).

Chapter 7

ACT: Managing Compact Corpora

The Alpino corpora can be stored in what we call "Compact Corpora". They consist of two files per corpus: one file with the compressed data and a seperate file with the filename information. Together they're in a way similar to .zip files, but their compression ratio (because they typically consist of a lot of small files) is much better. The compression method is based on `dictzip` (see www.dict.org).

All of our tools work transparently on files in a normal directory structure and those in a compact corpus.

7.1 Creating compact corpora

The tool to manage compact corpora is called `act`, which stands for Alpino Corpus Tool. From the `act` point of view, a corpus is a directory that contains .xml files.

7.1.1 Compressing a single directory

Let's assume we have the following directory structure:

```
corpus_directory
|-- cdb
|   |-- 0.xml
|   |-- 1.xml
|   |-- 1.xml~
|   |-- 10.xml
|   |-- 100.xml
|   |-- 1000.xml
... ...
|   |-- 997.xml
|   |-- 998.xml
|   |-- 999.xml
|   |-- CVS
|   |   |-- Entries
|   |   |-- Repository
|   |   `-- Root
|   `-- Makefile
`-- compact_corpora
```

We want to create a compact corpus for `cdb` in the `compact_corpora` directory. Here are two ways to do this:

```
$ cd compact_corpora
$ act --create ../cdb
```

The same can be accomplished with:

```
$ act --create --targetdir=compact_corpora cdb
```

Both methods will put two new files in the `compact_corpora` subdirectory:

```
`-- compact_corpora
    |-- cdb.data.dz
    `-- cdb.index
```

Compact corpora will only contain .xml files. In our example above, `1.xml~`, the `CVS` directory structure, and `Makefile` are all ignored.

7.1.2 Compressing a directory tree recursively

Consider the following directory structure (only directories shown):

```
corpus_directory
|-- cdb
|   `-- CVS
|-- Machine
|   `-- clef
|       |-- AD19940103
|       |-- AD19940104
|       |-- AD19940105
|       |-- AD19940106
|       |-- AD19940107
|       `-- AD19940108
`-- compact_corpora
```

Assume only `cdb` and the directories starting with "AD" contain .xml files.

The following will create a compact version of the `Machine` directory structure under `compact_corpora/Machine`:

```
$ act --create --recursive --targetdir=compact_corpora/Machine Machine/
```

The target-directory will be created if necessary.

The default for `\--targetdir` is the current directory, so the following yields the same result as the previous example:

```
$ mkdir -p compact_corpora/Machine
$ cd compact_corpora/Machine
$ act --recursive ../../Machine
```

To replace the `Machine` subdirectory with a compact equivalent use the following:

```
$ cd Machine
$ act --remove --create --recursive .
```

The `\--remove` option will cause the source files to be removed after creating the compact corpus.

When using recursion to create compact corpora, the directories specified **should not** contain any xml files, i.e. they should (only) contain other directories.

7.2 Updating compact corpora

Scenario: there's a compact corpus and there are several files that have updates. Say, a couple of sentences have been reparsed.

Updating will work on a sparse directory structure, i.e. the directory structure only needs to contain the reparsed files. These files will be merged with the existing compact corpus or compact corpora.

```
act --recursive --update --targetdir clef newly_parsed_clef
```

7.2.1 Asymmetry of the `\--targetdir` option

When using `\--recursive`, the `\--targetdir` option provides a straight directory to directory mapping. With regular corpus directories (directories that contain xml files) however this is not possible, because when converting to a compact corpus the name of the last directory component has to map to a *filename*.

Let's illustrate this with two examples. The `cdb` directory is a regular corpus directory and (therefore) contains `.xml` files. The `cdb` compact corpus will be created/updated *below* the `compact_corpora` directory:

```
act --update cdb --targetdir=compact_corpora
```

The `clef` directory does not contain any `.xml` files, it contains other directories that contain the `.xml` files. In this case we have a direct mapping from directory to directory:

```
act --recursive --update clef --targetdir=compact_corpora/clef
```

Therefore the directory arguments given to `act \--recursive` should not contain any `.xml` files, otherwise the compact corpora may end up in places not intended.

7.3 Extracting compact corpora

To extract a compact corpus use the `\--extract` flag. At the moment of writing extracting only works non-recursively.

Extracting creates a new directory with `.xml` files for every compact corpus specified on the commandline. For example, the following extracts a collection of compact corpora to a specific directory (current directory is the default):

```
$ act --targetdir /lots/of/space/Machine/clef --extract Machine/clef/*.index
```

Use `\--force` to have `act` overwrite existing directories.

Chapter 8

DTList: Listing the contents of a compact corpus

To show the contents of a compact corpus use ``dtlist''. The compact corpus `cdb can be specified as cdb, as cdb.index or as cdb.data.dz.

```
$ dtlist compact/corpora/cdb.index
compact_corpora/cdb/0.xml
compact_corpora/cdb/1.xml
compact_corpora/cdb/2.xml
compact_corpora/cdb/3.xml
compact_corpora/cdb/4.xml
compact_corpora/cdb/5.xml
...
```

Note the complete path in the output and the numerical sorting.

Watch out with wildcards and compact corpora
The following will give **two** listings! One for cdb.data.dz, and one for cdb.index:

```
$ dtlist compact_corpora/cdb*
```

Chapter 9

DTGrep: grep in dependency structures

DTGrep can be used to search in dependency structures with regular expressions. The `dtgrep` program basically is a stripped down `grep` program that knows about Compact Corpora. DTGrep uses Perl-compatible regular expressions.

```
usage: dtgrep [options] arguments

options:
  --help                show this help message and exit
  -ePATTERN, --regexp=PATTERN
                        use PATTERN as a regular expression
  -i, --ignore-case     ignore case distinctions
  -v, --invert-match    select non-matching lines
  -l, --files-with-matches
                        only print FILE names containing matches
  -h, --no-filename     suppress the prefixing filename on output
  --stdin               read file arguments from stdin.  Any non-option
                        arguments on the commandline will be ignored.
```

Chapter 10

DTGet: Write dependency structure to standard output

The `dtget` tool simply takes a sequence of one or more filenames, and prints the corresponding dependency structures to standard output.