1. ADQL from Scripts

Hendrik Heinl *(heinl@ari.uni-heidelberg.de)*

Agenda

- Make and Makefiles
- STILTS
- Scripting ADQL queries

Open a browser on `http://docs.g-vo.org/ais/html`

2. Make and Makefile

`make` is a tool to control the generation of executables and non-sourcefiles of a program. Users can define rules which set how a target file shall be build and which dependencies have to fulfilled beforehand.

Rules:

▷ 1
```
target: [dependencies]
  recipe
```

Here you see the principle syntax of a make rule. Dependencies are optional and are only used if needed for building a target. A specific make rule could look like this:

▷ 2
```
table1.vot: table2.vot
  python magicpipeline.py

table2.vot:
  ./fortran95
```

Here table2.vot is a prerequisite to table1.vot. `make` will look at the rule for table2.vot before processing the recipe of table1. For instance the fortran code in the recipe of table2 may write a file that is used by magicpipeline.py in the recpipe of the rule of table1.vot.

3. The Makefile

For the usage of make a file named `Makefile` in the working folder is needed. The build can be started from the command-line with `make`.

▷ 3
```
target1.txt:
  touch target1.txt
```

Defining dependencies:

▷ 4
```
targetA.txt: targetB.txt
  touch targetA.txt

targetB.txt:
  touch targetB.txt
```

By default, `make` will start to build the first target. If the target has dependencies defined, those will be build first. Targets that are not part of dependency chains and that are not the first in the makefile will be ignored.

▷ 5
```
target1.txt: target2.txt
  touch target1.txt
target2.txt:
  touch target2.txt
target3.txt: target2.txt
  touch target3.txt
```

Target files that already exist in the work folder will considered to be fulfilled. `make` does not check for the content though, it simply checks the existance.

In the examples `touch` is used to simply create a file in the work folder.

4. make - calling targets

`make targetname` can be used to override the default of starting with the first target, but instead call the build of a specific target independently from its position in the make file. This is extremely useful for cleaning the work folder from leftovers from older builds.

▷ 6
```
clean:
  rm *.txt
```

5. make - recipes

`make` recipes describe how to build a specific target. In general, shell programming can be used (bash on *IX machines). Furthermore, full programms can be executed which then can read files and built a target file.

▷ 7
```
ra.txt:
  echo 123g.2345 > ra.txt
de.txt:
  echo -54.8765 > de.txt
pipe: ra.txt de.txt
  python pipe.py
```

Here the first targets write ra and de into two different files.Then a python script merges them to a proper position and writes the result into the file position.csv.

pipe.py:

```
import string
def make_position():
  a = open('ra.txt', 'r')
  b = open('de.txt', 'r')
  ra = a.readline().rstrip(string.whitespace)
  de = b.readline().rstrip(string.whitespace)
  return ra + ' , ' + de
if __name__=="__main__":
  print make_position()
  f=open('position.csv', 'w')
  f.write(make_position())
```

Line breaks:

Some commands are executed with a few arguments. This can make recipes a littly messy. For increased readability, line breaks can be inserted but must be escaped with a \.

6. STILTS

STILTS is a set of command-line tools based on STIL and forms the command-line counterpart of TOPCAT. With STILTS tabular data such as astronomical catalogs can be processed. STILTS supports several formats as FITS, VOTable, CDF, GBIN, CSV and ASCII. Built in function are:

- crossmatching
- plotting
- column calculation and rearrangement
- data and metadata manipulation and display
- statistical and histogramm calculation
- access to remote data services including Virtual Observatory

7. ADQL in STILTS

TAP is one of the VO protocols supported by STILTS: Performing an ADQL query with STILS could look like this:

▷ 8
```
STILTS tapquery \
 tapurl='http://gaia.esac.esa.int/tap-server/tap' \
 omode=out out=t1.vot \
 ofmt=vot \
 delete=never \
 adql="SELECT \
 TOP 10000 \
 source_id, ra, dec, parallax, pmra, pmdec\
  FROM tgas_source\
 WHERE \
 1=CONTAINS(POINT('ICRS', ra, dec),\
 CIRCLE('ICRS', 56.75, 24.1167, 10.5 ))"
```

A brief explanation of the STILTS arguments:

- `tapquery` calls the TAP query functions
- `tapurl` is needed to identify the service that shall be queried
- omode defines the outputmode. Here we chose VOTable.
- out sets the file that shall contain the results
- `delete` is used to set when a finished job shall be deleted on the server
- `adql` is the ADQL query. By default, a STILTS TAP query is in asychronous mode. After starting the query STILTS will return a hyperlink pointing to the job on the remote service. If "delete=never" is set, this link can be used to return to the job at a later time and after the job has finished, to download the results.

8. Tapupload with STILTS

▷ 9
```
STILTS tapquery \
 tapurl='http://dc.zah.uni-heidelberg.de/tap' \
 omode=out nupload=1 \
 upload1=t1.vot upname1=tab \
 out=t2.vot ofmt=vot \
 adql="SELECT \
 mine.*,\
 tm.raj2000, tm.dej2000, tm.jmag, tm.hmag, tm.e_jmag, tm.e_hmag\
 FROM tap_upload.tab AS mine\
 JOIN twomass.data AS tm\
 ON 1=CONTAINS(\
 POINT('ICRS', tm.raj2000, tm.dej2000),\
 CIRCLE('ICRS', mine.ra, mine.dec, 3/3600.))"
```

The STILTS tapupload arguments:

- `nupload` sets number of uploads
- `upload1` sets the name of the file for the upload
- `upname1` sets the alias of the uploaded file that can be used to refer to it in the ADQL query.