## Installation

- *Using Linux at HW MACS*
- Requirements:
  - Java - *already installed*
  - SWI Prolog **-** *Version 5.7.11 already installed in Linux*
  - *If you are using a different version of SWI Prolog, you might need to make a small change to queryRespond translation.sh:*
    - *comment out pl -s callProcesses.pl*
    - *and remove comment from # swipl -s callProcesses.pl*
  - jena – uses the **arq** command line utility (only)
  - svn client and access to daisy-spsm (a Google Code repository)
- Download daisy from the Google Code repository
  - cd to appropriate directory (e.g. I used ~dsb5/CHAIN/code)
    - svn checkout https://daisy-spsm.googlecode.com/svn/trunk daisy-spsm --username DSBENTAL@gmail.com
  - this creates a root directory **daisy-spsm** (in current directory)
  - with subdirectories
    - documentation
    - output
    - queryRepair
    - queryRespond
    - spsm
- Compile S-Match/SPSM version so far used for CHAIn
  - cd daisy-spsm/spsm/s-match-20110317
  - ant jar (clean compile)
- Alternatively build new S-Match/SPSM version
  - preparation (if not already installed)
    - update packages: sudo apt-get update
    - install maven: sudo apt-get install maven
  - Goto folder daisy-spsm/spsm/s-match-source and run:
    - sh 2buildSMatch
    - sh 3createBinary
    - sh 4extractBinary
  - For testing purpose go to folder daisy-spsm/spsm/s-match/bin and run ./all-spsm.sh, you should now find a file results.txt in folder daisy-spsm/spsm/s-match/test-data/spsm
  - If no problems occurred, then edit file daisy-spsm/spsm/prolog-spsm.sh
    - comment line: export S_MATCH_HOME=$DAISY_HOME_LOCAL/spsm/s-match-20110317
    - uncomment line: #export S_MATCH_HOME=$DAISY_HOME_LOCAL/spsm/s-match
  - Optional: Clone newest version of S-Match by running:
    - install git: sudo apt-get install git
    - sh 0cloneSMatch before running the other bash files
    - Source files for S-Match V2.0 are included
    - Note:

- If you clone the newest source files, then command sh 4extractBinary might no longer work
- in 4extractBinary change export SMATCH_BINARY=s-match-2.0.0-SNAPSHOT to point to the new file
- Check folder s-match-utils/target/ for new name of file

- Install Jena
  - https://jena.apache.org/download/index.cgi
  - download e.g. apache-jena-2.12.0.tar.gz into some useful directory - I used ~dsb5/CHAIN/code/jena
  - unzip (with gunzip and then tar –xf) to create apache-jena-2.12.0
  - nb for quick handy info about Jena SPARQL installation and usage:
    - http://richard.cyganiak.de/blog/wp-content/uploads/2013/09/jena-sparql-cli-v1.pdf
- Change shell variables to point to the right directories
  - In daisy-spsm/queryRespond/**translation.sh**
  - Change the **ARQROOT** variable to point to Jena ARQ so it points to the version of arq that was installed in the apache- jena directory
    - i.e. the ARQROOT line should read:
    - export ARQROOT=/home/dsb5/CHAIN/code/jena/apache-jena-2.12.0
    - (or wherever jena has been installed)
  - Change the **DAISY_HOME** variable to point to the daisy-spsm directory
    - i.e. the DAISY_HOME line should read
    - export DAISY_HOME=/home/dsb5/CHAIN/code/daisy-spsm
    - (or wherever)

## First run

- cd daisy_spsm/queryRespond
- ./translation.sh (starts Prolog)
- respond_to_query. (in Prolog)
- nb No output is created at present….
- nb. This will work with files specified in the queryRespond/translation.sh file. These data files are the SEPA dataset in
  - daisy_spsm/queryRespond/datasets/sepa/*n3
  - and with a test query file (based on EUStatistics freshwater)
    - queryRespond/queries/test/query*pl
  - and a demo source dataset (based on EUstatististic freshwater)
    - daisy_spsm/queryRespond/datasets/demo-data/water.n3
- ~~This should complete and create a proper daisy_spsm/output/result.pl file~~
- ~~which should now look like the file **result.txt** in this directory.~~

## Running a different query

- Queries are kept in subdirectories of:
  - daisy_spsm/queryRespond/queries/
    - EUStatistics/
      - environmental Protection expenditure regions/
      - freshWaterResources/

        - NAPTAN/
        - query1/
        - query2/
        - SPA/
        - uk Environment Agency/
- Each set of queries consists of a file called
    - query.pl
- Edit daisy-spsm/queryRespond/translation.sh
    - and change the line
    - export DAISY_QUERY="${DAISY_HOME}/queryRespond/queries/test"
    - so as to point to a different query directory.
- If the query was derived from a different dataset altogether then change the line
    - export DAISY_SOURCE_DATA="${DAISY_HOME}/queryRespond/datasets/demo-dataset"
    - to point to the different dataset directory.
    - The source dataset is used to derive the initial set of rdf prefixes. The dataset name in this directory must be the same as the dataset name in the query; and all the names in the query must be rdf-predicates in the source dataset.
    - If you don't do this step then the code will still more or less work, but the source prefixes will either be wrong or missing. This typically wont matter anyway since the source and target prefixes are different.

## Running a different target data set

- Datasets are kept in subdirectories of:
    - daisy_spsm/queryRespond/datasets/
        - sepa/
            - i.e.
                - bathingWaterLocations.n3
                - classifications.n3
                - waterBodyMeasures.n3
                - bathingWaters.n3
                - surfaceWaterBodies.n3
                - waterBodyPressures.n3
        - EUStatistics/
            - i.e.
                - environmentalProtectioneEpenditure.n3
                - euPrefixes.txt
                - protectedAreas.n3
                - environmentProtectionRegions.n3
                - freshWaters.n3
                - transportNetwork.n3
- The original dataset used on download is the sepa dataset
    - daisy_spsm/queryRespond/datasets/sepa/
- To change the dataset, edit daisy-spsm/queryRespond/translation.sh
    - and change the line

    - export DAISY_DATA="${DAISY_HOME}/queryRespond/datasets/sepa"
    - so as to point to a different dataset directory.
- It should also be possible to run the program using a remote dataset on a Sparql endpoint but this isn't implemented.

## Convert between SWI Prolog and Sicstus

- This version runs in SWI Prolog.
- To use with Sicstus
    - *eg. at HW SICStus 4.2.3 already installed in Linux (also Windows)*
- In queryRespond/
- Edit **translation.sh** to remove the line
    - pl -s callProcesses.pl
    - and replace it with
    - sicstus --noinfo -l callProcesses.pl
- Edit **utils.pl**
    - Include the line
        - :- use_module(library(system3)).
        - :- use_module(library(sets)).
    - Remove the predicate definitions for
        - environ/2
        - system/1
        - directory_files/2
- If other system-specific predicates are used, they should be added to utils.
    - As far as possible changes are localised into utils.
    - utils loads up libraries that exist in Sicstus and not in SWI Prolog.
    - The names of Sicstus system3 library predicates are used, and SWI Prolog definitions are provided for them. environ/2, system/1, and directory_files/2
    - The Sicstus lists library is always loaded explicitly into all the modules. This doesn't seem to bother SWI Prolog, although SWI doesn't need it.
    - The Sicstus sets library is loaded into utils, and two predicates from it are given other names (my_union and is_list) because loading the Sicstus sets library seems to bother SWI Prolog and I'm trying to keep changes for the different versions in the one place.
- A flag is set for Sicstus in callProcess.pl to turn off warnings when redefining static predicates
    - :- set_prolog_flag(redefine_warnings, off).
    - The warnings are turned off because they are not just a warning, Sicstus stops and asks what to do every tim
    - The flag does not seem to affect SWI Prolog.