README for SemRep Processing

Dongwook Shin, Ph.D.  
Date: August 15 2019

Java package: gov.nih.nlm.skr.semrepProcess

**Data format:**

Input: XML format compliant with MEDLINE BASELINE converts XML files into the corresponding ASCII format (a separate directory) and generating input ASCII file for semrep processing. The input file format for Semrep is:

First line: PMID- pmid\_number

Second line: TI - title

Third line: AB - abstract

EXAMPLE:

PMID- 30

TI - Lysosomal hydrolases of the epidermis. I. Glycosidases.

AB - Seven distinct glycosidases (EC 3.2) have been characterized in guinea-pig epidermis. Their properties indicate them to be of lysosomal origin. The 'profile' of

the epidermal glycosidases is significantly different from that reported for whole skin, the activities of beta-galactosidase and beta-acetylglucosaminidase being very

high and those of the remaining enzymes relatively low in epidermis.

**Process**

1. Create two databases; one for PREPROCESS and the other for SemMedDatabase
   1. The database schema for each database is:

CREATE TABLE FACT\_DATA (

`FACT\_DATA\_ID` int(10) unsigned NOT NULL auto\_increment,

PMID varchar(20) NOT NULL,

EXIST\_XML tinyint(1) default 0,

EXIST\_SEMREP tinyint(1) default 0,

XML\_DATA MEDIUMTEXT,

SEMREP\_DATA MEDIUMTEXT,

TITLE varchar(999),

ABSTRACT MEDIUMTEXT,

PRIMARY KEY (`FACT\_DATA\_ID`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1 ;

create index pmid\_index\_btree using btree on FACT\_DATA (PMID);

* 1. The database schema for SemMedDB is described in appendix A.

This database name can be configured in ROOT/semrep.properties file as follows:

connectionString=jdbc:mysql://indsrv2.nlm.nih.gov

preddatabase=PREPROCESS

semmeddatabase=semmedVER50

dbusername=user name for the database

dbpassword=password for the database

Connection string is the connection string for MySQL database that Is to the Semrep output.

Please note that the user has to have the read/write privilege for the database.

1. Check if there is a jar file ./lib/semrepProcess.jar. If not, run the following command and generate it.
   1. % ant main –f build\_semrepProcess.xml

If a Java program in ./src directory is modified for some reason, make sure to run the “ant” command and have the latest semrepProcess.jar file.

1. Set the parameters appropriately in the file ROOT/semrep.properties. The parameters relevant to Semrep processing in addition to those described in 1.b are as follows:

semrepLoadingProgram=resources/semrepping\_40WithSH.pl

semrepentityLoadingProgram=resources/semrepping\_entity\_40.pl

perlScript=/usr/bin/perl

schedulerAccount=yourUMLSAccount

schedulerPassword=yourUMLSPassword

semrepLoadingProgram is the Perl script that is used in loading SemRep output into the database, whereas semrepentityLoadingProgram is the Perl script that loads Entity into ENTITY table. As ENTITY table is optional in SemMed database, it is not loaded while predications and sentences are loaded. Instead, it is loaded separately.

schedulerAccout and schedulerPassword are the account information that is used to login to UMLS.

1. Run gov.nih.nlm.skr.semrepProcess.Semrepping. This class takes (1) the MEDLINE\_BASELINE directory as input, (2) converting XML files into corresponding ASCII format, (3) generating input ASCII file for Semrep processing and (4) putting Semrep result. The shell script ‘*semrepping.sh*” does this step automatically. The arguments need to be:
   1. First argument – MEDLINE\_BASELINE directory
   2. Second argument – ASCII conversion directory from MEDLINE\_BASELINE
   3. Third argument – directory for Medline format
   4. Forth argument – directory for Semrep output

* In this process, PMID, title and abstract are inserted into the FACT\_DATA table for the corresponding row. This is because if Semrep step fails for some citations in Step 3 (4), the Normalized format and MEDLINE format are recreated from these tables and resubmitted in step 5.
* The reason that this class needs ASCII converted XML directory is that it extracts metadata information from XML structure and inserts it into CITATIONS table in the SemMed database before any sentence and predication is inserted.
* From SemMedDB version 4.0, SENTENCE table is dependent on CITATIONS table and unless the metadata info for a PMID is created in CITATIONS table, the sentences are not inserted into SENTENCE table.
* Semrep results will be loaded into SemMed database using the perl semrep loading program. It is configured in ROOT/factuality\_semrep.properties file as:

semrepLoadingProgram=resources/semrepping\_40WithSH.pl

perlScript=/usr/bin/perl

* This step can be performed using ROOT/semrepping.sh

1. gov.nih.nlm.skr.semrepProcess.semrepingFromDB. This class is looking at the PREPROCESS database and finds the PMIDs whose EXIST\_SEMREP is 0 instead of 1. If EXIST\_SEMREP is 0, it means that the citation has not been semrepped yet (either because it was not semrepped at all or semrepping has failed) and needs to be semrepped in this step. SemreppingFromDB class is scanning from FACT\_DATA table, extracting *limit* PMIDs starting from *offset*, generating input MEDLINE format, and sending those to the Scheduler. The output SemRep result is stored in the temp directory specified in the third argument (see below). The shell script “*SemreppingFromDB.sh*” does this task automatically. This class takes three input parameters as follows:
   1. First argument is the integer specifying the *limit*, which is a number of each batch (number of MEDLINE citations that are to be taken to the Scheduler)
   2. Second argument is the starting offset from which the PMIDs whose EXIST\_SEMREP is 0 are extracted.
   3. Third argument is the temp directories where SemRep output is stored and loaded into the SemMedDB database.
   4. As SemRepping of some citations can fail in the Scheduler for various reasons (including timeout or Scheduler failure), gov.nih.nlm.skr.semrepProcess.SemrepingFromDB class needs to be executed repeatedly until the remaining citations cannot be semrepped any further.
2. Sometimes, in Step 5, Semrep results fail to be loaded into the database. In that case, use the class gov.nih.nlm.skr.semrepProcess.LoadingSemrepToDB and load the SemRep outputs into the SemMed database. There is a shell script “loadsemrep2semmeddb.sh” that does this process automatically. The input arguments are:
   1. First argument is the directory where SemRep output is located
   2. The extension of SemRep output file. In the sample, the extension is “semrep”.
   3. The third argument is the starting number of the SemRep file in the directory specified in a.
   4. The fourth argument is the ending number of the SemRep file in the directory.
3. After all the SemRep results are loaded into the database, make sure that the predications and sentences are in PREDICATION and SENTENCE table, respectively. At this stage, NORMALIZED\_SECTION\_HEADER is still null, while SECTION\_HEADER is populated from the SemRep output. So the class gov.nih.nlm.skr.PopulateNormalizedSH converts SECTION\_HEADER into normalized section header and populates that into NORMALIZED\_SECTION\_HEADER column. In this process, the file SectionHeaderNormalization.txt is used, where the first column indicates possible section header and the second column shows the matching normalized section header. A shell script populateNormalizedSH.sh does this step automatically. The input arguments are:
   1. The first argument is the file name that has the mapping from Section Header to Normalized Section header
   2. The second argument is the start location of row in SENTENCE table. Normally it is 0. But when you want to start this operation at the n-th row of SENTENCE table, you need to give the number “n”.
   3. The third argument is the limit of the select statement. As the number of predications is too big, these operation can be done in batch matter, selecting “limit” numbers of SENTENCE each time in order to reduce the memory consumption.
4. In PREDICATION table, the novelty column is still not populated and you need to populate it to SUBJ\_NOVELTY and OBJECT\_NOVELTY column. gov.nih.nlm.skr.PopulateNoveltyToPredication does this procedure. A shell script populateNovelty.sh does this step automatically.
   1. The first argument is the start location of row in PREDICATION table. Normally it is 0. But when you want to start this operation at the n-th row of PREDICATION table, you need to give the number “n”.
   2. The second argument is the limit of the select statement. As the number of predications is too big, these operation can be done in batch matter, selecting “limit” numbers of SENTENCE each time in order to reduce the memory consumption.
5. At this point, SemRep results are loaded into tables, but entity information is not. The reason that entity information is not loaded when predications are loaded is that entity information is optional and the table is huge. So if you want to load entity information, use the class gov.nih.nlm.skr.LoadSemrepEntity, whose arguments are as follows:

LoadSemrepEntity class extracts the SemRep result saved in SEMREP\_DATA column in FACT\_DATA of PREPROCESS database and loads them into the ENTITY table in the SemMed database. In doing so, it starts from the *offset,* which is the second argument and extracts *limit* number of PMIDs at one time. LoadSemrepEntity class uses the following SQL statement and extracts SemRep output:

Select \* from FACT\_DATA where EXIST\_SEMREP = 1 limit = *limit* offset = *offset*

**Appendix A. Database schema for SemMed database**

-- table CITATIONS

CREATE TABLE METAINFO (

DBVERSION varchar(10) NOT NULL,

SEMREPVERSION varchar(10),

PUBMED\_TODATE varchar(10),

COMMENT varchar(500)

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE CITATIONS (

PMID varchar(20) NOT NULL,

ISSN varchar(10),

DP varchar(50),

EDAT varchar(50),

PYEAR int(5),

PRIMARY KEY (`PMID`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1 ;

CREATE TABLE `GENERIC\_CONCEPT` (

`CONCEPT\_ID` int(10) unsigned NOT NULL auto\_increment,

`CUI` varchar(20) NOT NULL default '',

`PREFERRED\_NAME` varchar(200) character set utf8 NOT NULL default '',

PRIMARY KEY (`CONCEPT\_ID`),

KEY `PREFERRED\_NAME` (`PREFERRED\_NAME`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='Stores basic UMLS concept information';

-- table SENTENCE

CREATE TABLE `SENTENCE` (

`SENTENCE\_ID` int(10) unsigned NOT NULL auto\_increment,

`PMID` varchar(20) NOT NULL default '',

`TYPE` varchar(2) NOT NULL default '',

`NUMBER` int(10) unsigned NOT NULL default '0',

`SENT\_START\_INDEX` int(10) unsigned NOT NULL default '0',

`SENT\_END\_INDEX` int(10) unsigned NOT NULL default '0',

`SECTION\_HEADER` varchar(100),

`NORMALIZED\_SECTION\_HEADER` varchar(50),

`SENTENCE` varchar(999) character set utf8 NOT NULL default '',

PRIMARY KEY (`SENTENCE\_ID`),

FOREIGN KEY (`PMID`) REFERENCES `CITATIONS` (`PMID`) ON DELETE CASCADE ON UPDATE CASCADE,

UNIQUE KEY `SENTENCE` (`PMID`,`TYPE`,`NUMBER`),

KEY `PMID\_INDEX` USING BTREE (`PMID`),

KEY `PMID\_HASH` USING HASH (`PMID`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='Stores sentences from Medline';

CREATE TABLE `PREDICATION` (

PREDICATION\_ID int(10) unsigned NOT NULL auto\_increment,

SENTENCE\_ID int(10) unsigned NOT NULL ,

PMID varchar(20),

PREDICATE varchar(50),

SUBJECT\_CUI varchar(255),

SUBJECT\_NAME varchar(999),

SUBJECT\_SEMTYPE varchar(50),

SUBJECT\_NOVELTY tinyint(1),

OBJECT\_CUI varchar(255),

OBJECT\_NAME varchar(999),

OBJECT\_SEMTYPE varchar(50),

OBJECT\_NOVELTY tinyint(1),

FACT\_VALUE char(20),

MOD\_SCALE char(20),

MOD\_VALUE float(3,2),

PRIMARY KEY (PREDICATION\_ID),

-- Commented on March 23 2010, Dongwook since InnoDB cannot be created because of Foreign key constraint

-- In "Creating InnoDB Tables" and the title "FOREIGN KEY Constraints"...

-- Foreign keys definitions are subject to the following conditions:

-- Both tables must be InnoDB type.

-- In the referencing table, there must be an index where the foreign key columns are listed as the first columns in the same order.

-- In the referenced table, there must be an index where the referenced columns are listed as the first columns in the same order.

-- Index prefixes on foreign key columns are not supported. One consequence of this is that BLOB and TEXT columns cannot be included in a foreign key, because indexes on those columns must always include a prefix length.

FOREIGN KEY (SENTENCE\_ID) REFERENCES SENTENCE(SENTENCE\_ID) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (PMID) REFERENCES SENTENCE(PMID) ON DELETE CASCADE ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='Stores aggregate info of semantic predications';

create index pmid\_index\_btree using btree on PREDICATION (PMID);

-- table PREDICATION\_AUX

CREATE TABLE `PREDICATION\_AUX` (

`PREDICATION\_AUX\_ID` int(10) unsigned NOT NULL auto\_increment,

`PREDICATION\_ID` int(10) unsigned NOT NULL,

`SUBJECT\_TEXT` varchar(200) default '' COMMENT 'Should be NOT NULL eventually',

`SUBJECT\_DIST` int(10) unsigned default '0' COMMENT 'Should be NOT NULL eventually',

`SUBJECT\_MAXDIST` int(10) unsigned default '0' COMMENT 'Should be NOT NULL eventually',

`SUBJECT\_START\_INDEX` int(10) unsigned default '0' COMMENT 'Should be NOT NULL eventually',

`SUBJECT\_END\_INDEX` int(10) unsigned default '0' COMMENT 'Should be NOT NULL eventually',

`SUBJECT\_SCORE` int(10) unsigned default '0' COMMENT 'Should be NOT NULL eventually',

`INDICATOR\_TYPE` varchar(10) default '' COMMENT 'Should be NOT NULL eventually',

`PREDICATE\_START\_INDEX` int(10) unsigned default '0' COMMENT 'Should be NOT NULL eventually',

`PREDICATE\_END\_INDEX` int(10) unsigned default '0' COMMENT 'Should be NOT NULL eventually',

`OBJECT\_TEXT` varchar(200) default '' COMMENT 'Should be NOT NULL eventually',

`OBJECT\_DIST` int(10) unsigned default '0' COMMENT 'Should be NOT NULL eventually',

`OBJECT\_MAXDIST` int(10) unsigned default '0' COMMENT 'Should be NOT NULL eventually',

`OBJECT\_START\_INDEX` int(10) unsigned default '0' COMMENT 'Should be NOT NULL eventually',

`OBJECT\_END\_INDEX` int(10) unsigned default '0' COMMENT 'Should be NOT NULL eventually',

`OBJECT\_SCORE` int(10) unsigned default '0' COMMENT 'Should be NOT NULL eventually',

`CURR\_TIMESTAMP` timestamp NOT NULL default CURRENT\_TIMESTAMP on update CURRENT\_TIMESTAMP,

PRIMARY KEY (`PREDICATION\_AUX\_ID`),

FOREIGN KEY (`PREDICATION\_ID`) REFERENCES `PREDICATION` (`PREDICATION\_ID`) ON DELETE CASCADE ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='Stores semantic predications in sentences';

-- table COREFERENCE

CREATE TABLE `COREFERENCE` (

`COREFERENCE\_ID` int(10) unsigned NOT NULL auto\_increment,

`PMID` varchar(20) NOT NULL default '',

`ANA\_CUI` varchar(255),

`ANA\_NAME` varchar(999),

`ANA\_SEMTYPE` varchar(50),

`ANA\_TEXT` varchar(200) default '',

`ANA\_SENTENCE\_ID` int(10) unsigned NOT NULL,

`ANA\_START\_INDEX` int(10) unsigned default '0' ,

`ANA\_END\_INDEX` int(10) unsigned default '0' ,

`ANA\_SCORE` int(10) unsigned default '0' ,

`ANT\_CUI` varchar(255),

`ANT\_NAME` varchar(999),

`ANT\_SEMTYPE` varchar(50),

`ANT\_TEXT` varchar(200) default '',

`ANT\_SENTENCE\_ID` int(10) unsigned NOT NULL,

`ANT\_START\_INDEX` int(10) unsigned default '0' ,

`ANT\_END\_INDEX` int(10) unsigned default '0' ,

`ANT\_SCORE` int(10) unsigned default '0' ,

`CURR\_TIMESTAMP` timestamp NOT NULL default CURRENT\_TIMESTAMP on update CURRENT\_TIMESTAMP,

PRIMARY KEY (`COREFERENCE\_ID`),

UNIQUE KEY (`PMID`,`ANA\_START\_INDEX`,`ANA\_END\_INDEX`, `ANT\_START\_INDEX`,`ANT\_END\_INDEX`),

CONSTRAINT `PMID\_ibfk\_1` FOREIGN KEY (`PMID`) REFERENCES `SENTENCE` (`PMID`) ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT `ANA\_SENTENCE\_ID\_ibfk\_4` FOREIGN KEY (`ANA\_SENTENCE\_ID`) REFERENCES `SENTENCE` (`SENTENCE\_ID`) ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT `ANT\_SENTENCE\_ID\_ibfk\_5` FOREIGN KEY (`ANT\_SENTENCE\_ID`) REFERENCES `SENTENCE` (`SENTENCE\_ID`) ON DELETE CASCADE ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='Stores coreference';

-- table ENTITY

CREATE TABLE `ENTITY` (

`ENTITY\_ID` int(10) unsigned NOT NULL auto\_increment,

`SENTENCE\_ID` int(10) unsigned NOT NULL,

`CUI` varchar(255),

`NAME` varchar(999),

`SEMTYPE` varchar(50),

`GENE\_ID` varchar(20) NOT NULL default '',

`GENE\_NAME` varchar(20) NOT NULL default '',

`TEXT` varchar(200) default '',

`SCORE` int(10) unsigned default '0' COMMENT 'Should be NOT NULL eventually',

`START\_INDEX` int(10) unsigned default '0' COMMENT 'Should be NOT NULL eventually',

`END\_INDEX` int(10) unsigned default '0' COMMENT 'Should be NOT NULL eventually',

PRIMARY KEY (`ENTITY\_ID`),

FOREIGN KEY (`SENTENCE\_ID`) REFERENCES `SENTENCE` (`SENTENCE\_ID`) ON DELETE CASCADE ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='Stores semantic predications in sentences';