

# HOWTO: Build and Deploy the UMLS references terminologies \*DRAFT\*

## Introduction/Background

The VPR has a built in terminology run-time engine dubbed "TermEng" that utilizes standard reference terminologies such as [LOINC](#), [SNOMED CT](#), [RxNorm](#), etc.

TermEng is:	TermEng is NOT:
<ul style="list-style-type: none"><li>• A lightweight interface/wrapper/bridge to one or more terminology servers/services, called data sources.</li><li>• A simple java object model, used within the VPR, for getting term attributes and checking relationships in an efficient manner</li><li>• Intended to support a very high query volume via a high performance caching layer.</li></ul>	<ol style="list-style-type: none"><li>1. A terminology server (like Apelon, 3M HDD, UMLS UTS, etc.)</li><li>2. Intended to support content authoring.</li><li>3. Intended to replace any authoritative data sources</li></ol>

Because TermEng can interface with multiple terminology services, a plugin must be developed (called an TermDataSource) to interface with each system. We have experimented with a few different types of TermDataSources but the one that is currently the most complete/useful is one that reads from a pre-compiled, [UMLS Metathesaurus](#) (a machine-readable vocabulary knowledge source, that is useful in a variety of settings) stored in the popular [H2 database](#) file format, called the H2TermDataSource, and is the datasource that is currently configured into the VPR platform and the most widely used.

H2TermDataSource combined with UMLS Metathesarus	
Advantages	Disadvantages
<ol style="list-style-type: none"><li>1. for development environments, it can be cumbersome to require a lot of extra 3rd party systems (like Apelon)</li><li>2. for production environments, relying on an external server (like Apelon) creates a single point of failure for the whole system</li><li>3. having simple file-based databases like H2, make it simple to swap out old content with new content</li><li>4. The UMLS metathesarus already does the work of merging the structures of all the source vocabularies we don't have to develop separate import mechanisms for each vocabulary.</li></ol>	<ol style="list-style-type: none"><li>1. Downloading and compiling UMLS is time consuming (even though it only needs to be done once)</li><li>2. Due to UMLS licencing, we are unsure if we are permitted to re-distribute a pre-compiled UMLS database to simplify the deployment process which would save a lot of time (and the need for this whole guide!)</li><li>3. It does require deploying a couple hundred extra megabytes of data to each VPR instance</li></ol>

Other TermDataSource Ideas/Proposals/Projects		
Name	Status	Description

VETSTermDataSource	Experimental	<p>This was one of the first explorations, it used web services to read from the VETS terminology server developed by STS.</p> <p>It does not work anymore, but we can resurrect it when/if needed.</p>
UMLSRRFDataSource	In Progress	<p>Realizing how complex it is to build a UMLS database using the mechanisms described below, this is an attempt at simplifying the process greatly by reading directly from the UMLS .RRF files.</p> <p>This eliminates the need for a temporary relational database, its a little difficult to work with the .RRF files w/o having tons of memory, so it needs more work.</p>
GOFDataSource	Experimental	<p>This was a first attempt at generated a Cache globals export file (.GOF) for some experimental work at incorporating some UMLS content into JDS (specifically doing a native JDS subsumption index)</p> <p>This is being merged into UMLSRRFDataSource</p>
MongoTermDataSource	Experimental	<p>In the early days we were using Mongo as an alternative JDS CDR. This was an experiment at pushing terminology content into MongoDB to do subsumption queries. It probably doesn't work anymore.</p>
ApelonDataSource	Proposal	<p>At some point, we should try hooking Apelon and 3M HDD into this as they are more stable data sources, but similarly, we will need to pre-compile their content into faster datasources so that the entire HMP platform does not fail if one of these data sources fail.</p>
HDDDataSource	Proposal	

VistADataSource	Proposal	It might be interesting to pull some terminology (master files for orders, meds, etc.) from local VistA accounts. We already have some of it in other places, but it could be interesting to merge it into TermEng as well.
CTSDataSoruce	Proposal	<a href="#">HL7's CTS</a> is a standard for querying terminology services. Actually the VETSTermDataSoruce was loosely based on CTS. Maybe we could formalize the VETS source into a CTS source for consuming anything published that way. It might also be a common way to query Aphelon or 3M HDD.
SPARQLDataSource	Proposal	As _____ mentioned, RDF/SPARQL are emerging standards for publishing and consuming terminology services. We might be able to skip the intermediary (UMLS, Aphelon, HDD, etc) and go straight to the source (like IHSTDO) by crawling RDF/SPARQL web-services.

## Overview of how to generate a pre-compiled UMLS database:

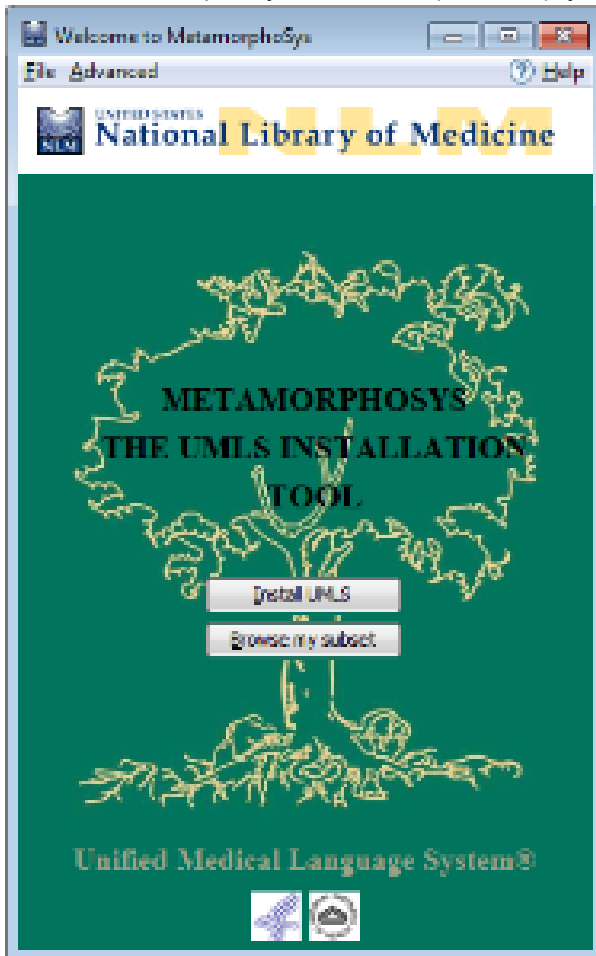
- Prerequisites/Warnings/Ramblings:
  - Be a java developer, with some decent SQL knowledge, because this isn't going to be easy..... sorry 😞
  - Have a lot of time to wait for the various steps to complete (some take hours or more)
  - Accept my apologies in advance because its going to be a long day for you....
  - All of my engineering morals/principals told me not to document this completely, but to finish up the UMLSRRFDataSource (mentioned above) which makes this 10x easier and 10x faster**
    - Unfortunately, we don't have time at the moment to finish it (management decision), so this will have to do for now.
  - I also think we can ultimately distribute the finished .zip files (how does Aphelon/3M do it?) but that decision needs to be vetted by someone else
- Download the [UMLS Knowledge Source Files](#)
  - You will need to create an account first if you don't already have one.
  - This guide is being written based on UMLS version 2012AB
  - Time Required: The files total 3+ GB, so it may take a while
- Use the [UMLS MetamorphoSys](#) installation wizard to subset UMLS to the specific vocabularies of interest
  - We currently mainly use LOINC, SNOMED CT and drug database (NDF,NDF-RT,RxNorm)
  - CPT and others will be used down the road, and might have [licencing issues](#)
  - Time Required: ~30-60m
- Load the UMLS subset into a temporary PostgreSQL database
  - This is an unfortunate intermediate step that will hopefully not be necessary in the near future.
  - Time Required: 1+ hours
-

5. Generate condensed JSON documents from the PostgreSQL database which are stored into an H2 database
  - a. Uses a custom java CLI program
  - b. The H2 database files are usually .zipped to conserve space
  - c. Time Required: 1-2 hours
6. Deploy the files to the VPR (/data) directory

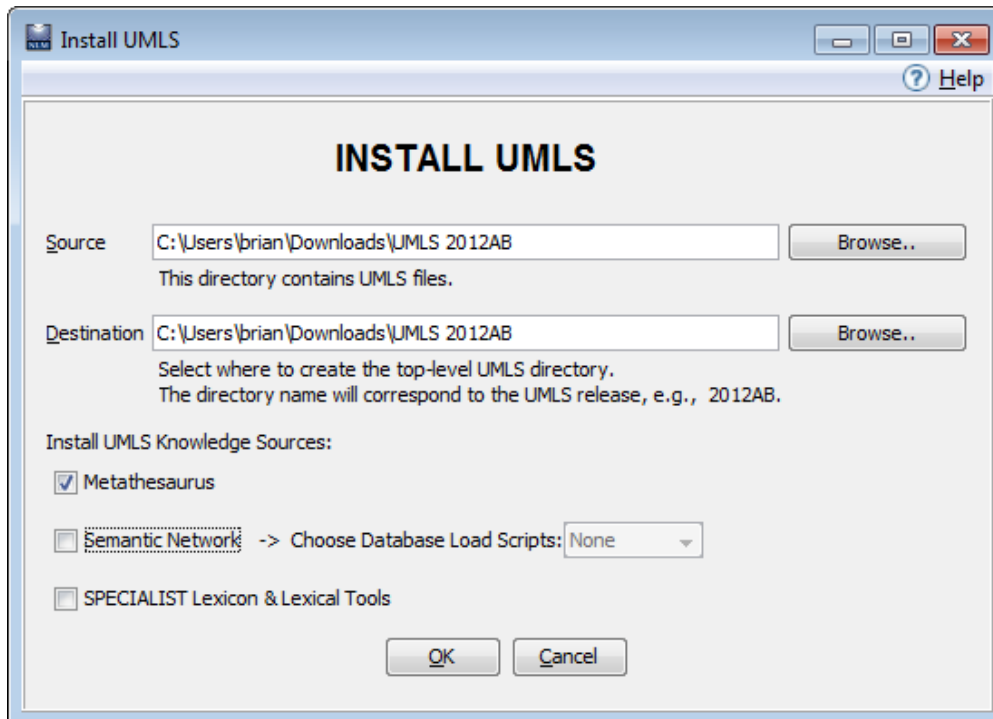
## Detailed Steps

### Download and Subset UMLS

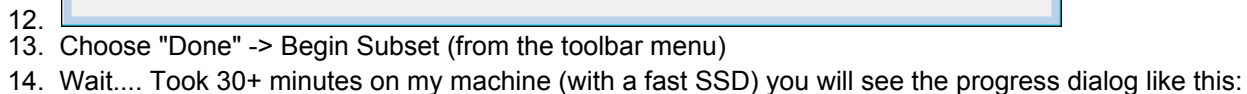
1. After downloading the files, move them all into a single directory (eg UMLS2012AB)
2. Extract mmsys.zip into the current directory
3. Launch MetamorphoSys via run.bat (windows), you should see something like this:



- 4.
5. Click Install UMLS, you only need to install Metathesarus:

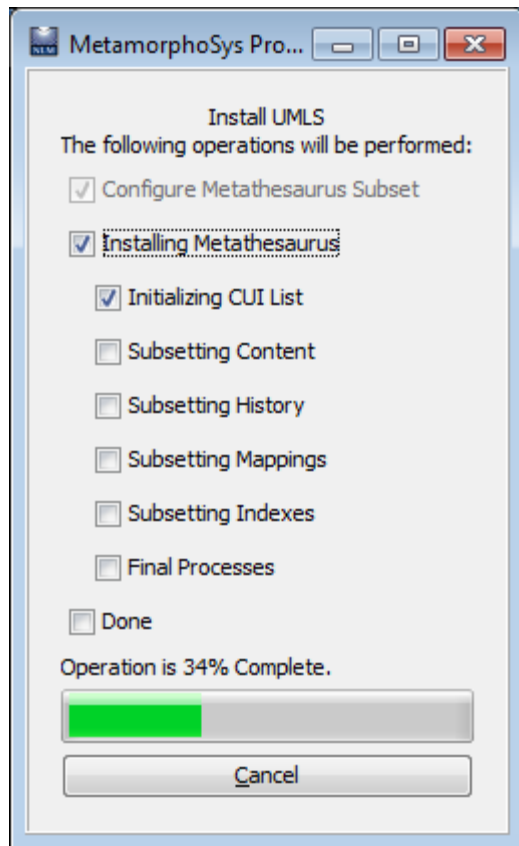


- 6.
7. Select New Configuration, accept the licence agreement, choose any of the default subsets
8. The defaults should be fine for the most part, but under the "Output Options" tab:
  - a. deselect "build and sort browser index files (/indexes)" (this will save a lot of time)
9. Under the source list tab, CTRL-CLICK to ensure the follow are selected:
  - a. International Classification of Diseases, Ninth Revision, Clinic Modification 21013 (ICD9CM\_2013)
  - b. LOINC, 240 (LNC240)
  - c. National Drug File, 2012\_09\_04 (NDFRT\_2012\_09\_04)
  - d. RxNorm Vocabulary, 12AA\_120904F
  - e. SNOMED Clinical Terms, 2012\_07\_31
  - f. Veterans Health Administration National Drug File, 2012\_07\_27
10. When choosing some of these sources, it will ask you if you want to include related sources, say yes
  - a. However, there may be licencing issues
  - b. For SNOMED CT it asks to include US extensions and Spanish language terms.
    - i. I only selected US extensions because Spanish is a large subset.
11. It should look something like this:

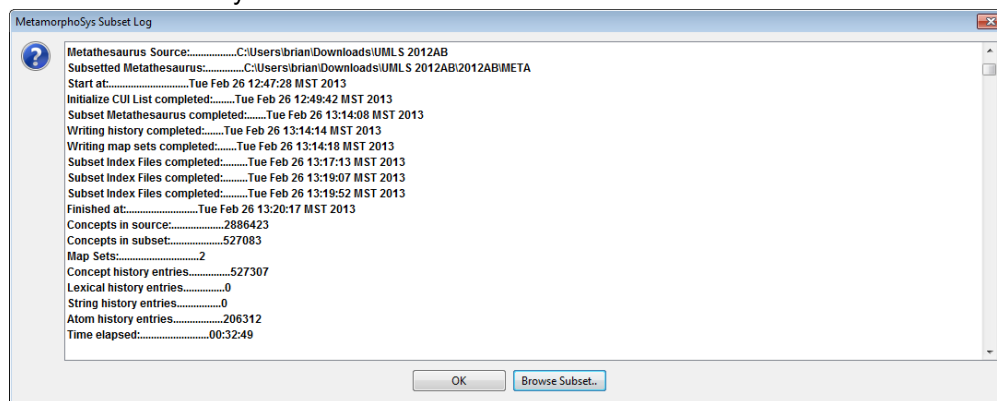


13. Choose "Done" -> Begin Subset (from the toolbar menu)

14. Wait.... Took 30+ minutes on my machine (with a fast SSD) you will see the progress dialog like this:



- 15.
16. When complete, you will get a status report, and you should have a ton of data (nearly 7+ gigs) in the destination directory



17.

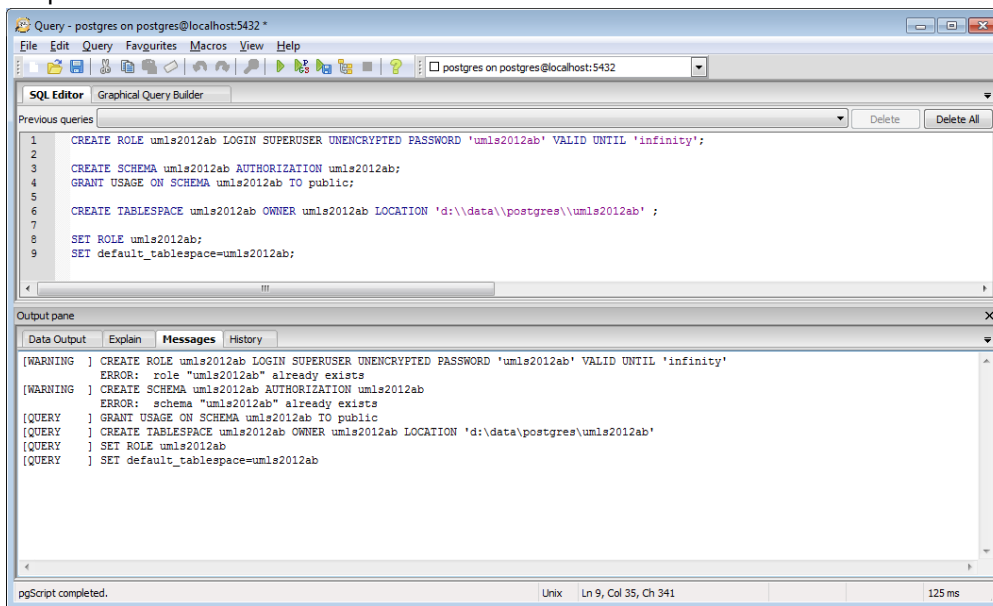
## Load the UMLS subset into temporary relational database

1. To load the UMLS subset, there is a java loader program in the /hmp/hmp-main/umls-import-tools/ project.
  - a. The UMLSLoader.bat script is the launching point.
2. I installed PostgreSQL 9.1 initially to explore how UMLS was structured and what data elements were needed by the VPR.
  - a. Other RDBMS's might work (Cache Relational, mySQL, etc.) but would likely need some hacking of the loader program.

3. In my database, I have many different UMLS versions and schemas, but the following script should set a new database up enough to get things ready for the loader program
- this script creates a new schema, new user (with the same password) and a new tablespace (since its a lot of data, I needed to put it on my other hard disk)
  - you need to create the target directory first (mine was d:\data\postgres\UMLS2012AB)
  - execute this as a pg script:

```
CREATE ROLE umls2012ab LOGIN UNENCRYPTED PASSWORD 'umls2012ab' VALID UNTIL
'infinity';
CREATE SCHEMA umls2012ab AUTHORIZATION umls2012ab;
GRANT USAGE ON SCHEMA umls2012ab TO public;
CREATE TABLESPACE umls2012ab OWNER umls2012ab LOCATION
E'd:\\data\\postgres\\umls2012ab' ;
SET ROLE umls2012ab;
SET default_tablespace=umls2012ab;
```

- d. Output looks like this:



- e.

4. Now you are ready to load UMLS into your database. Modify the /hmp/hmp-main/umls-import-tools/UMLSLoader.bat script with your new schema name, username and password.
- also ensure that its pointing to the correct RRF file locations (which were generated by the UMLS MetaMorphosys subsetting process)
  - UPDATE:** Remove the file reference to MRHIER.rrf from the script, it is not needed by later steps and will save you 20+ minutes of loading....



5. Run the UMLSLoader.bat, Wait.... Took over an hour for me....
  - a. The loader program will generate a schema, load all the specified files, generate indexes and analyze/optimize the tables.
  - b. The output will look something like this:

```

SQL=COMMENT ON COLUMN umls2012ab.MRCONSO.CODE IS 'Unique Identifier or code for
string in source'
SQL=ALTER TABLE umls2012ab.MRCONSO ADD COLUMN STR varchar(3000) NOT NULL
SQL=COMMENT ON COLUMN umls2012ab.MRCONSO.STR IS 'String'
SQL=ALTER TABLE umls2012ab.MRCONSO ADD COLUMN SRL integer NOT NULL
SQL=COMMENT ON COLUMN umls2012ab.MRCONSO.SRL IS 'Source Restriction Level'
SQL=ALTER TABLE umls2012ab.MRCONSO ADD COLUMN SUPPRESS char(1) NOT NULL
SQL=COMMENT ON COLUMN umls2012ab.MRCONSO.SUPPRESS IS 'Suppressible flag'
SQL=ALTER TABLE umls2012ab.MRCONSO ADD COLUMN CUF integer
SQL=COMMENT ON COLUMN umls2012ab.MRCONSO.CUF IS 'Content view flag'
Loading MRCONSO.RRF. (2984622 of 2984622 records complete) 100%
Done.
SQL=CREATE INDEX X_MRCONSO_CUI ON umls2012ab.MRCONSO (CUI) TABLESPACE umls2012ab
SQL=CREATE INDEX X_MRCONSO_AUI ON umls2012ab.MRCONSO (AUI) TABLESPACE umls2012ab
SQL=CREATE INDEX X_MRCONSO_idx1 ON umls2012ab.MRCONSO (code, sab) TABLESPACE uml
s2012ab
SQL=VACUUM ANALYZE umls2012ab.MRCONSO
SQL=DROP TABLE IF EXISTS umls2012ab.MRREL
SQL=CREATE TABLE umls2012ab.MRREL (<) TABLESPACE umls2012ab
SQL=ALTER TABLE umls2012ab.MRREL OWNER TO umls2012ab
SQL=COMMENT ON TABLE umls2012ab.MRREL IS 'Related Concepts'
SQL=ALTER TABLE umls2012ab.MRREL ADD COLUMN CUI1 char(8) NOT NULL
SQL=COMMENT ON COLUMN umls2012ab.MRREL.CUI1 IS 'Unique identifier for first conc
ept'
SQL=ALTER TABLE umls2012ab.MRREL ADD COLUMN AUI1 varchar(9) NOT NULL
SQL=COMMENT ON COLUMN umls2012ab.MRREL.AUI1 IS 'Unique identifier for first atom
'
SQL=ALTER TABLE umls2012ab.MRREL ADD COLUMN STYPE1 varchar(50) NOT NULL
SQL=COMMENT ON COLUMN umls2012ab.MRREL.STYPE1 IS 'The name of the column in MRCON
SO.RRF that contains the first identifier to which the relationship is attached
'
SQL=ALTER TABLE umls2012ab.MRREL ADD COLUMN REL varchar(4) NOT NULL
SQL=COMMENT ON COLUMN umls2012ab.MRREL.REL IS 'Relationship label'
SQL=ALTER TABLE umls2012ab.MRREL ADD COLUMN CUI2 char(8) NOT NULL
SQL=COMMENT ON COLUMN umls2012ab.MRREL.CUI2 IS 'Unique identifier for second con
cept'
SQL=ALTER TABLE umls2012ab.MRREL ADD COLUMN AUI2 varchar(9) NOT NULL
SQL=COMMENT ON COLUMN umls2012ab.MRREL.AUI2 IS 'Unique identifier for second ato
m'
SQL=ALTER TABLE umls2012ab.MRREL ADD COLUMN STYPE2 varchar(50) NOT NULL
SQL=COMMENT ON COLUMN umls2012ab.MRREL.STYPE2 IS 'The name of the column in MRCON
SO.RRF that contains the second identifier to which the relationship is attache
d'
SQL=ALTER TABLE umls2012ab.MRREL ADD COLUMN REL1 varchar(100)
SQL=COMMENT ON COLUMN umls2012ab.MRREL.REL1 IS 'Additional relationship label'
SQL=ALTER TABLE umls2012ab.MRREL ADD COLUMN RUI varchar(10) NOT NULL
SQL=COMMENT ON COLUMN umls2012ab.MRREL.RUI IS 'Unique identifier for relationshi
p'
SQL=ALTER TABLE umls2012ab.MRREL ADD COLUMN SRUI varchar(50)
SQL=COMMENT ON COLUMN umls2012ab.MRREL.SRUI IS 'Source attributed relationship i
dentifier'
SQL=ALTER TABLE umls2012ab.MRREL ADD COLUMN SAB varchar(20) NOT NULL
SQL=COMMENT ON COLUMN umls2012ab.MRREL.SAB IS 'Source abbreviation'
SQL=ALTER TABLE umls2012ab.MRREL ADD COLUMN SL varchar(20) NOT NULL
SQL=COMMENT ON COLUMN umls2012ab.MRREL.SL IS 'Source of relationship labels'
SQL=ALTER TABLE umls2012ab.MRREL ADD COLUMN RG varchar(10)
SQL=COMMENT ON COLUMN umls2012ab.MRREL.RG IS 'Relationship group'
SQL=ALTER TABLE umls2012ab.MRREL ADD COLUMN DIR varchar(1)
SQL=COMMENT ON COLUMN umls2012ab.MRREL.DIR IS 'Source asserted directionality fl
ag'
SQL=ALTER TABLE umls2012ab.MRREL ADD COLUMN SUPPRESS char(1) NOT NULL
SQL=COMMENT ON COLUMN umls2012ab.MRREL.SUPPRESS IS 'Suppressible flag'
SQL=ALTER TABLE umls2012ab.MRREL ADD COLUMN CUF integer
SQL=COMMENT ON COLUMN umls2012ab.MRREL.CUF IS 'Content view flag'
Loading MRREL.RRF. (3028000 of 9389454 records complete) 32%

```

- c.
6. Create one extra index:
  - a. This index is needed for the next step but does not get generated automatically:
  - b.
 

```

CREATE INDEX x_mrel_idx2
ON umls2012ab.mrrel
USING btree
(aui2 COLLATE pg_catalog."default" )
TABLESPACE umls2012ab;

```
  - c. It will take a few minutes to generate the index

7. Verify:

- a. I have a lot of extra schema's, but this is what my PostgreSQL database looks like after its done loading:

The screenshot shows the pgAdmin III interface. The left pane displays the 'Object browser' tree for 'PostgreSQL 9.1 (localhost:5432)'. The tree structure includes:

- Servers (1)
  - PostgreSQL 9.1 (localhost:5432)
    - Databases (1)
      - postgres
        - Catalogs (2)
        - Extensions (2)
        - Schemas (5)
          - public
          - umls2011ab
          - umls2011abfull
          - umls2012aa
          - umls2012ab
        - Collations (0)
        - Domains (0)
        - FTS Configurations (0)
        - FTS Dictionaries (0)
        - FTS Parsers (0)
        - FTS Templates (0)
        - Functions (0)
        - Sequences (0)
        - Tables (5)
          - mrconso
          - mrhier
          - mrrank
          - mrrel
          - mrsat
        - Trigger Functions (0)
        - Views (0)
      - Slony Replication (0)
    - Tablespaces (5)
      - pg\_default
      - pg\_global
      - umls2011ab
      - umls2011abfull
      - umls2012aa
    - Group Roles (0)
    - Login Roles (4)
      - postgres

The right pane shows the 'Statistics' tab for the selected 'Tables (5)'. It displays a table of statistics:

Statistic	Value
Sequential Scans	17
Sequential Tuples Read	8953866
Index Scans	0
Index Tuples Fetched	0
Tuples Inserted	2984622
Tuples Updated	0
Tuples Deleted	0
Tuples HOT Updated	0
Live Tuples	2984526
Dead Tuples	0
Heap Blocks Read	366186
Heap Blocks Hit	3212480
Index Blocks Read	36879
Index Blocks Hit	0
Toast Blocks Read	0
Toast Blocks Hit	0
Toast Index Blocks Read	0
Toast Index Blocks Hit	0
Last Vacuum	2013-02-26 16:07:50.547-07
Last Autovacuum	
Last Analyze	2013-02-26 16:07:52.741-07
Last Autoanalyze	2013-02-26 16:05:00.192-07
Vacuum counter	1
Autovacuum counter	0
Analyze counter	1
Autoanalyze counter	2
Table Size	507 MB
Toast Table Size	8192 bytes
Indexes Size	288 MB

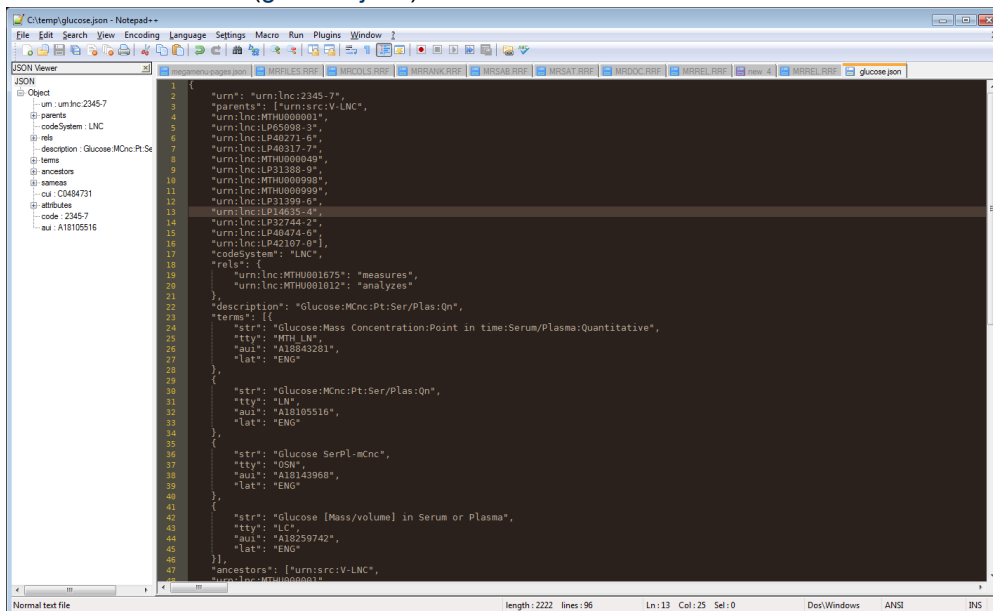
At the bottom of the window, a status bar indicates 'Refreshing tables... Done.' and '0.23 secs'.

b.

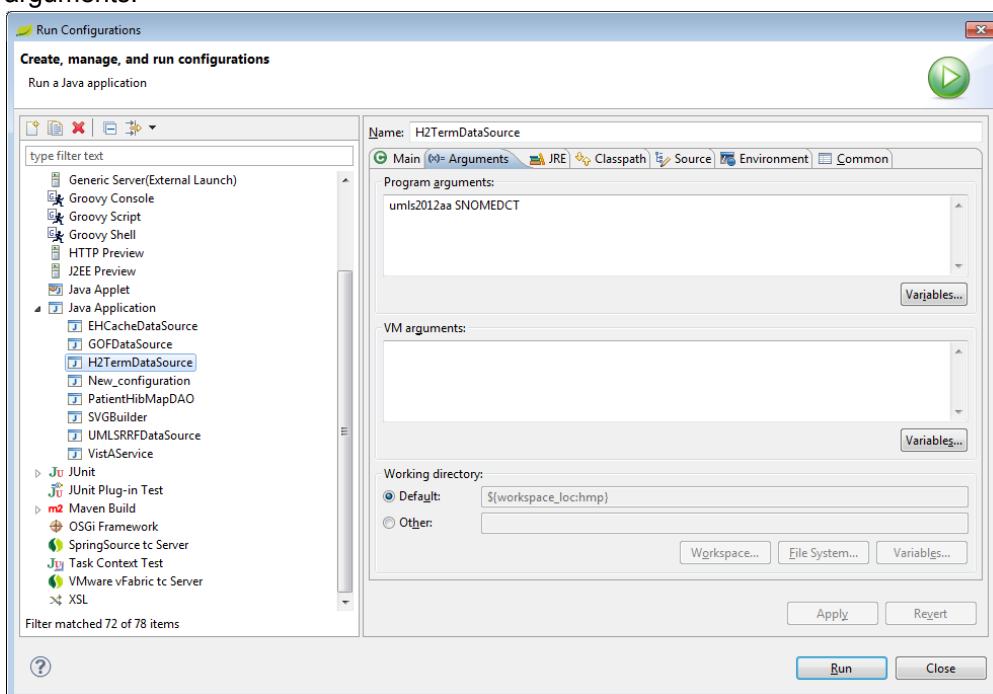
8. You no longer need the UMLS files or downloads, you may delete them if you want.

**Generate JSON extracts and store them into H2 DB files**

1. Now that we have all the UMLS data in a queryable form, we need to convert each concept into a small-ish JSON document that looks something like this:
  - a. Which looks like this ([glucose.json](#))



- b.
2. Open `gov.va.cpe.vpr.termeng.H2TermDataSource.java` in Eclipse (or your java IDE of choice)
  3. This class has a `main(String[] args)` method to launch the process, so right click on the class and select run as -> Java Application.
    - a. It will probably throw an error the first time, we need to edit the run configuration to add 2 program arguments:



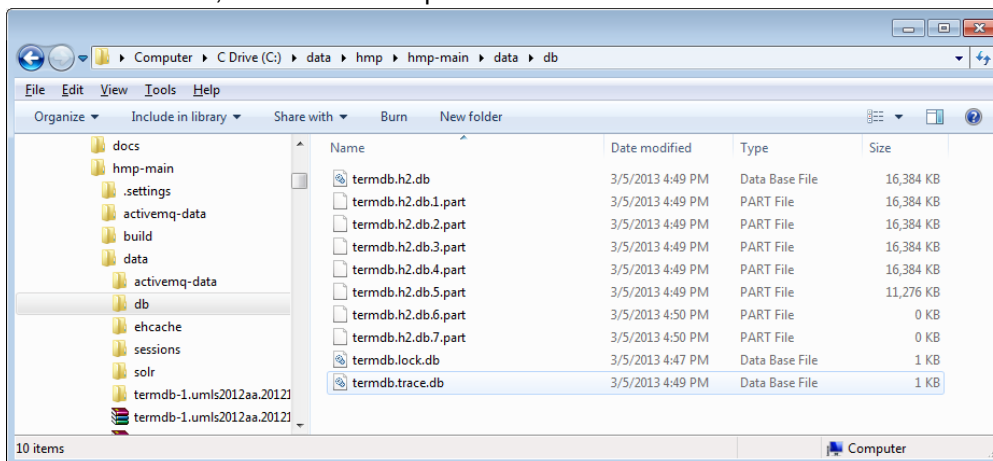
- b.
- c. some of the configuration is still built into the source code, so if you need to change the postgres port etc, you may need to change the code in the `main()` method a bit.
- d. The first argument says which postgresQL schema to use: **umls2012ab**
- e. the second argument says which source abbreviations to read: **SNOMEDCT**

4. We are actually going to run this program 3 times to generate 3 different vocabulary database files:
  - a. **H2TermDataSource umls2012aa SNOMEDCT** to generate **SNOMEDCT.zip**
    - i. This contains just the SNOMEDCT source as its the largest vocabulary
    - ii. Took: about 1.5 hours for me
  - b. **H2TermDataSource umls2012aa LNC** to generate **LOINC.zip**
    - i. This only contains LOINC codes
    - ii. Took: about 10 minutes for me
  - c. **H2TermDataSource umls2012aa VANDF NDFRT RXNORM** to generate **DRUGS.zip**
    - i. This includes the 3 main drug systems that we use in HMP: NDF, NDF-RT and RxNorm
    - ii. Took: about 20 minutes for me
  - d. Eventually, we will build a **PROBLEMS.ZIP** as well which will include ICD-9,10 and CPT codes
    - i. we don't actually have anything that uses it yet, and
    - ii. There are licencing issues with CPT, so
    - iii. Skip this for now....
  - e. These can take a while, you might want to start with LOINC as its the smallest
5. Here is what the output looks like while its running:

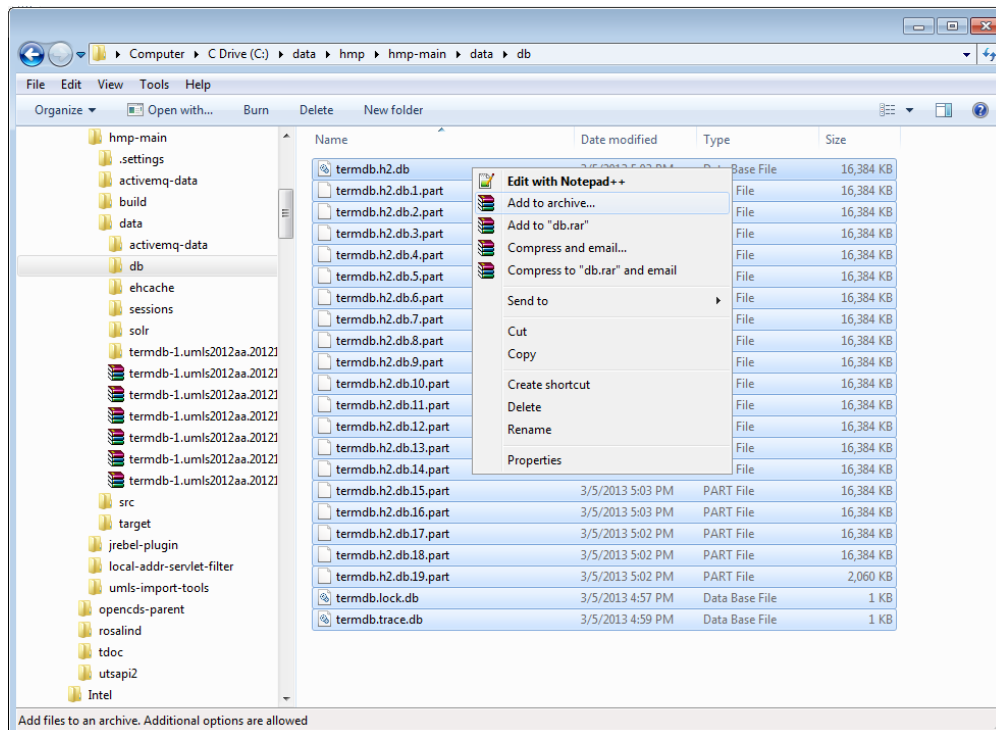
```

H2TermDataSource [Java Application] C:\data\apps\java\jdk1.6.0_29\bin\javaw.exe (Mar 5, 2013 4:57:13 PM)
Loading: LNC (count: 399622)
1.00% complete. (4000/399622 terms, 1000 concepts)
1.99% complete. (7972/399622 terms, 2000 concepts)
3.00% complete. (11972/399622 terms, 3000 concepts)
4.00% complete. (15972/399622 terms, 4000 concepts)
5.00% complete. (19972/399622 terms, 5000 concepts)
6.00% complete. (23972/399622 terms, 6000 concepts)
7.00% complete. (27962/399622 terms, 7000 concepts)
8.00% complete. (31962/399622 terms, 8000 concepts)
  
```

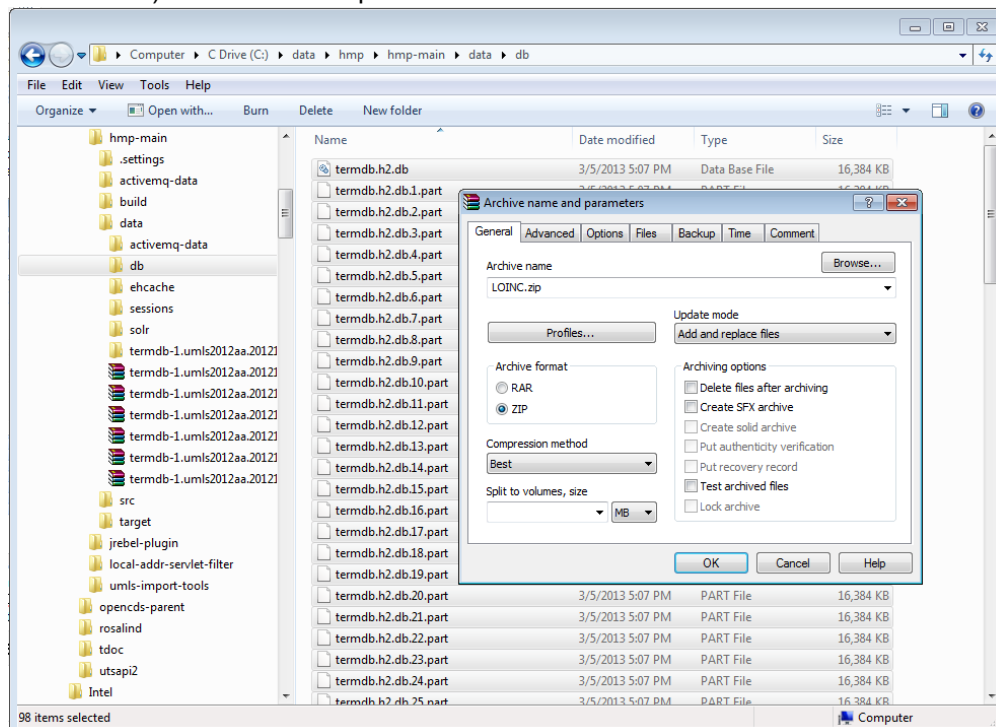
- a.
6. However, **DO NOT** run these 3 back to back, you need to move/compress the files in between:
  - a. Open `hmp/hmp-main/data/db/`
  - b. it will look like this, but with tons of .part files



- c.
- d. These files represent the entire database of LOINC,SNOMED,etc concepts.
- e. Now that its been generated its read-only so we can compress these large files into one smaller .zip file
  - i. Add all the files in the DB directory (termdb.\*) to a .zip file

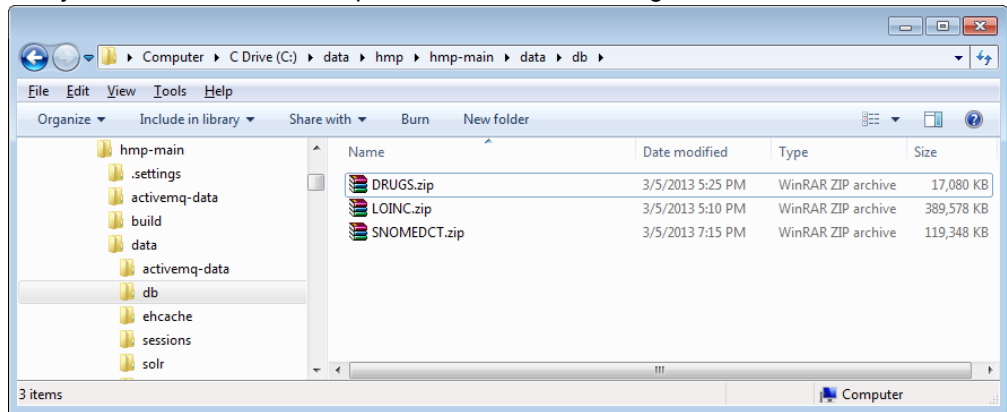


- ii.
- iii. I use WinRAR b/c I like it, and I think there are limitations to the built in windows stuff, but feel free to .zip with your tool of choice.
- iv. I changed the archive name to reflect the vocabulary (ex: LOINC.ZIP, SNOMEDCT.zip and DRUGS.ZIP) and set the compression to max



- v.
- vi. now, delete all the termdb.\* files (but not the .zip you just created) and run the next database build

vii. In the end, you should have the 3 .zip files that look something like this:



1.

7. After you create all 3 databases, we no longer need the postgresSQL database, you may remove it if you want.

## Incorporate the files into the HMP spring/maven configuration

1. Now that you have generated the 3 database files, we need to plug them into TermEng before you can start up HMP
2. We use Spring XML configuration to do this
3. Search for frame-config.xml (this is the spring resource.xml file that contains all the knowledge subset of spring beans)
  - a. its located in hmp/src/main/webapp/WEB-INF/spring/
4. find the termEng bean, it currently looks something like this:
  - a. 

```
<!-- Terminology Config -->
<bean id="termEng" class="gov.va.cpe.vpr.termeng.TermEng"
factory-method="createInstance">
<constructor-arg index="0">
<array>
<bean class="gov.va.cpe.vpr.termeng.H2TermDataSource" c:jdbcurl="jdbc:h2:split:zip:${termdb.dir}/termdb-${termdb.version}-drugdb.zip!/termdb"/>
<bean class="gov.va.cpe.vpr.termeng.H2TermDataSource" c:jdbcurl="jdbc:h2:split:zip:${termdb.dir}/termdb-${termdb.version}-loincdb.zip!/termdb"/>
<bean class="gov.va.cpe.vpr.termeng.H2TermDataSource" c:jdbcurl="jdbc:h2:split:zip:${termdb.dir}/termdb-${termdb.version}-sctdb.zip!/termdb"/>
</array>
</constructor-arg>
</bean>
```

5. Change it to point to the 3 files we just created:

a. `${termdb.dir}` points to `hmp/hmp-main/data/`

b. `<!-- Terminology Config -->`

```
<bean id="termEng" class="gov.va.cpe.vpr.termeng.TermEng"
```

```
factory-method="createInstance">
```

```
<constructor-arg index="0">
```

```
<array>
```

```
<bean class="gov.va.cpe.vpr.termeng.H2TermDataSource" c:jdbcurl="jdbc:h2:split:zip:\${termdb.dir}/db/DRUGS.zip!/termdb"/>
```

```
<bean class="gov.va.cpe.vpr.termeng.H2TermDataSource" c:jdbcurl="jdbc:h2:split:zip:\${termdb.dir}/db/LOINC.zip!/termdb"/>
```

```
<bean class="gov.va.cpe.vpr.termeng.H2TermDataSource" c:jdbcurl="jdbc:h2:split:zip:\${termdb.dir}/db/SNOMEDCT.zip!/termdb"/>
```

```
</array>
```

```
</constructor-arg>
```

```
</bean>
```

6. At the VA, I then uploaded these 3 .zip files (with different names) to our [nexus repository](#) and then used maven to create an artifact reference

- a. this is nice because:
  - i. maven automatically downloads the files and puts them in the correct spot the first time you start up the system
  - ii. I only ever had to generate these files once and upload them (thus the reason these instructions are so complex)
  - iii. we can version the database files like other libraries
- b. since not everyone has access to our internal nexus repository, you will probably need to remove those references since:
  - i. you manually put the files in the data/ directory
  - ii. the artifacts are not listed as optional, so maven probably is throwing errors
- c. To remove the references,

- i. edit hmp/hmp-main/pom.xml
- ii. find this portion of XML and REMOVE it (or comment it out)
- iii. 

```
<!-- This ensures that the required vocabulary databases are downloaded
from nexus -->
<execution>
<id>copy-termdb-databases</id>
<phase>process-resources</phase>
<goals>
<goal>copy</goal>
</goals>
<configuration>
<artifactItems>
<artifactItem>
<groupId>gov.va.hmp</groupId>
<artifactId>termdb</artifactId>
<version>${termdb.version}</version>
<classifier>drugdb</classifier>
<type>zip</type>
<overWrite>false</overWrite>
</artifactItem>
<artifactItem>
<groupId>gov.va.hmp</groupId>
<artifactId>termdb</artifactId>
<version>${termdb.version}</version>
<classifier>loincdb</classifier>
<type>zip</type>
<overWrite>false</overWrite>
</artifactItem>
<artifactItem>
<groupId>gov.va.hmp</groupId>
<artifactId>termdb</artifactId>
<version>${termdb.version}</version>
<classifier>sctdb</classifier>
<type>zip</type>
<overWrite>false</overWrite>
</artifactItem>
</artifactItems>
<outputDirectory>${basedir}/data</outputDirectory>
<overWriteReleases>false</overWriteReleases>
<overWriteSnapshots>true</overWriteSnapshots>
</configuration>
</execution>
```



