PyMedTermino

Jean-Baptiste Lamy

November 13, 2013

1 Introduction

PyMedTermino (Medical Terminologies for Python) is a Python module for easy access to the main medical terminologies in Python. The following terminologies are available:

- SNOMED CT
- ICD10
- UMLS
- VCM icons (an iconic terminology developped at Paris 13 University)

The main features of PyMedTermino are:

- A single API for accessing all terminologies
- Optimized full-text search
- Access to label, synonyms and translations
- Manage concepts and relations between concepts
- Mappings between terminologies, through UMLS or manual mapping files

For SNOMED CT and ICD10, the data are not included (because they are not freely redistribuable) but they can be downloaded for free in XML format. PyMedTermino includes scripts for exporting these data into SQLite3 databases.

For UMLS, data are not included (for the same reasons, and because they are voluminous). Thus, PyMedTermino need to be connected to a MySQL server including UMLS data, as provided by the NLM.

For VCM icons, the whole terminologies are provided as OWL ontologies and SQLite3 databases. However, the icons' image files are not included and must be downloaded separately (or you can use the VCM iconic server to generate icons dynamically): PyMedTermino only include the terminological part of VCM.

PyMedTermino has been created at the LIM&BIO reseach lab, Paris 13 University, Sorbonne Paris Cité, by Jean-Baptiste Lamy. PyMedTermino is available under the GNU LGPL licence.

Here is an example of what you can do with PyMedTermino:

```
>>> SNOMEDCT.search("tachycardia*")
[SNOMEDCT[3424008]  # Tachycardia (finding)
, SNOMEDCT[4006006]  # Fetal tachycardia affecting management of mother (disorder)
, SNOMEDCT[6456007]  # Supraventricular tachycardia (disorder)
...]

>>> SNOMEDCT[3424008].parents
[SNOMEDCT[301113001]  # Finding of heart rate (finding)
]

>>> SNOMEDCT[3424008].children
[SNOMEDCT[11092001]  # Sinus tachycardia (finding)
, SNOMEDCT[278086000]  # Baseline tachycardia (finding)
, SNOMEDCT[162992001]  # On examination - pulse rate tachycardia (finding)
...]
```

```
>>> list(SNOMEDCT[3424008].ancestors_no_double())
[SNOMEDCT[301113001]  # Finding of heart rate (finding)
, SNOMEDCT[106066004]  # Cardiac rhythm AND/OR rate finding (finding)
, SNOMEDCT[250171008]  # Clinical history and observation findings (finding)
, SNOMEDCT[404684003]  # Clinical finding (finding)
, SNOMEDCT[138875005]  # SNOMED CT Concept (SNOMED RT+CTV3)
...]

>>> SNOMEDCT[3424008].relations
set(['INVERSE_has_definitional_manifestation', 'finding_site', 'inter-
prets', 'has_interpretation', 'INVERSE_associated_with'])

>>> SNOMEDCT[3424008].finding_site
[SNOMEDCT[24964005]  # Cardiac conducting system structure (body structure)
]

>>> SNOMEDCT[3424008] >> VCM   # Maps the SNOMED CT concept to VCM icon
Concepts([
  VCM[u"current--hyper--heart_rhythm"]  #
])
```

2 Installation

1. Uncompress PyMedTermino-X.Y.tar.bz2
2. After registration with NLM, download SNOMED CT data:
   - http://www.nlm.nih.gov/research/umls/licensedcontent/snomedctfiles.html
     (download "SnomedCT_Release_INT_<date>.zip" and unzip)
   - http://www.nlm.nih.gov/research/umls/Snomed/core_subset.html
     (download "SNOMEDCT_CORE_SUBSET_<date>" and unzip)
3. After registration, download CIM10 data and its translations:
   - http://apps.who.int/classifications/apps/icd/ClassificationDownload/DLArea/Download.aspx
     (download ICD10 - "ClaML" format and unzip)
   - http://www.icd10.ch/
     (optional, for French and Deutch translations; download the XML format and unzip)
4. Edit the setup.py file and indicate the 4 paths where you have uncompressed the data, for example:

   ```
   SNOMEDCT_DIR = "/home/jiba/telechargements/base_med/SnomedCT_Release_INT_20130731"
   SNOMEDCT_CORE_FILE = "/home/jiba/telechargements/base_med/SNOMEDCT_CORE_SUBSET_201308.txt"
   ICD10_DIR = "/home/jiba/telechargements/base_med/icd10"
   CIM10_DIR = "/home/jiba/telechargements/base_med/cim10"
   ```

   Note: you can put empty strings if you don't want to instal the corresponding terminologies.
5. Compile PyMedTermino and convert the downloaded data in SQLite3 SQL databases:

   ```
   python setup.py build
   ```

   Caution : the database creation require an important disk space (~1-2 Gb).
6. Obtain root permissions. Under Linux, depending on your distribution, use one of the following commands:

   ```
   su # Mageia,...
   sudo -i # Ubuntu, Linux Mint,...
   ```
7. Instal PyMedTermino:

   ```
   python setup.py install
   ```
8. Clean the installation directory (optional, but frees an important disk space!):

   ```
   python setup.py clean
   ```

2.1 Troubleshooting

2.1.1 OperationalError: no such module: fts4

Under Windows, if you encounter this problem during install, you need to update the Sqlite3 DLL. For this, download the last version from http://www.sqlite.org/download.html, and replace the DLL in the Python27/DLLs directory by the downloaded version.

2.1.2 IOError: [Errno 22] Invalid argument

Under Windows, you get this error if the voluminous SNOMED CT files are in a shared directory. This limitation is due to the Microsoft system, thus you must put SNOMED CT files in a local directory (or use a real OS man :).

3 Loading modules and setting global parameters

```
>>> import pymedtermino
>>> pymedtermino.LANGUAGE = "en"
>>> pymedtermino.REMOVE_SUPPRESSED_CONCEPTS = 1
```

The following global parameters are available :

- pymedtermino.DATA_DIR : the directory where SQLite3 database files containing terminologies are located. Default : PyMedTermino directory.
- pymedtermino.LANGUAGE : the default language used for terms, when several translations are available. If the desired language is not available, it defaults to English. CAUTION : this parameter must be set BEFORE loading terminologies. Default : ”en” (English).
- pymedtermino.REMOVE_SUPPRESSED_CONCEPTS : if true, concepts tagged as suppressed or depreciated in terminologies are skipped. Default : 1 (true).
- pymedtermino.REMOVE_SUPPRESSED_TERMS : if true, terms (=translations) tagged as suppressed or depreciated in terminologies are skipped. Default : 1 (true).
- pymedtermino.REMOVE_SUPPRESSED_RELATIONS : if true, relations tagged as suppressed or depreciated in terminologies are skipped. Default : 1 (true).

4 SNOMED CT

4.1 Loading modules

It is now possible to import SNOMED CT in Python:

```
>>> from pymedtermino import *
>>> from pymedtermino.snomedct import *
```

4.2 Concepts

The SNOMEDCT object represents the SNOMED CT terminology. A SNOMED CT concept can be obtained from its code (in the following example, 302509004, which is the code for the heart concept) by indexing this object with curly brackets:

```
>>> concept = SNOMEDCT[302509004]
>>> concept
SNOMEDCT[302509004]  # Entire heart (body structure)
```

The has_concept() method can be used to verify if a code corresponds to a concept or not:

```
>>> SNOMEDCT.has_concept("invalid_code")
False
```

Each concept has a code and a term (= label corresponding to the preferred term) :

```
>>> concept.code
302509004

>>> concept.term
u'Entire heart (body structure)'
```

SNOMED CT also proposes synonym terms (notice the "s" on "terms") :

```
>>> concept.terms
[u'Heart', u'Entire heart', u'Entire heart (body structure)']
```

4.3 Full-text search

The search() method allows full-text search in SNOMED CT terms (including synonyms):

```
>>> SNOMEDCT.search("Cardiac structure")
[ SNOMEDCT[80891009] # Heart structure (body structure)
, SNOMEDCT[308793001] # Embryonic cardiac structure (body structure)
...]
```

Full-text search uses the FTS engine of SQLite, it is thus possible to use its functionalities. For example, for searching for all words beginning by a given prefix:

```
>>> SNOMEDCT.search("osteo*")
[ SNOMEDCT[1551001]  # Osteomyelitis of femur (disorder)
, SNOMEDCT[4598005]  # Osteomalacia (disorder)
...]
```

4.4 Is-a relations: parent and child concepts

The "parents" and "children" attributes return the list of parent and child concepts (i.e. the concepts with is-a relations):

```
>>> concept.parents
[SNOMEDCT[116004006]  # Hollow viscus (body structure)
, SNOMEDCT[80891009]  # Heart structure (body structure)
, SNOMEDCT[187639008]  # Entire thoracic viscus (body structure)
]
>>> concept.children
[SNOMEDCT[195591003]  # Entire transplanted heart (body structure)
]
```

The ancestors() and descendants() methods return all the ancestor concepts (parents, parents of parents, and so on) and the descendant concepts (children, children of children, and so on) :

```
>>> for ancestor in concept.ancestors(): print ancestor
SNOMEDCT[116004006]  # Hollow viscus (body structure)
SNOMEDCT[118760003]  # Entire viscus (body structure)
SNOMEDCT[272625005]  # Entire body organ (body structure)
[...]
```

The ancestors() and descendants() methods return Python generators; to obtain a list of ancestors or descendants, you should use the list() function:

```
>>> concept.ancestors()
<generator object ancestors at 0xb3f734c>

>>> list(concept.ancestors())
[SNOMEDCT[116004006]  # Hollow viscus (body structure)
, SNOMEDCT[118760003]  # Entire viscus (body structure)
, SNOMEDCT[272625005]  # Entire body organ (body structure)
,...]

>>> list(concept.descendants())
[SNOMEDCT[195591003]  # Entire transplanted heart (body structure)
]
```

ancestors_no_double() and descendants_no_double() methods behave identically but without duplicates. self_and_ancestors() and self_and_descendants() methods behave identically but include the concept itself in the returned concepts. self_and_ancestors_no_double() and self_and_descendants_no_double() methods combine both behaviors.

Finally, the is_a() method returns True if a concept is a descendant of another:

```
>>> concept.is_a(SNOMEDCT[272625005])
True
```

4.5 Part-of relations

"part_of" and "INVERSE_part_of" attributes provide access to subparts or superpart of the concept:

```
>>> concept.part_of
[SNOMEDCT[362010009] # Entire heart AND pericardium (body structure)
]

>>> concept.INVERSE_part_of
[SNOMEDCT[102298001] # Structure of chordae tendineae cordis (body structure)
, SNOMEDCT[181285005] # Entire heart valve (body structure)
, SNOMEDCT[181288007] # Entire tricuspid valve (body structure)
, SNOMEDCT[181293005] # Entire cardiac wall (body structure)
,...]
```

ancestor_parts() and descendant_parts() methods return a Python generator with all super- or subparts of the concept:

```
>>> list(concept.ancestor_parts())
[SNOMEDCT[362010009] # Entire heart AND pericardium (body structure)
, SNOMEDCT[362688008] # Entire middle mediastinum (body structure)
, SNOMEDCT[181217005] # Entire mediastinum (body structure)
, SNOMEDCT[302551006] # Entire thorax (body structure)
,...]

>>> list(concept.descendant_parts())
[SNOMEDCT[181285005]  # Entire heart valve (body structure)
, SNOMEDCT[192664000]  # Entire cardiac valve leaflet (body structure)
, SNOMEDCT[192747009]  # Structure of cardiac valve cusp (body structure)
,...]
```

Finally, the is_part_of() method return True if a concept is a part-of another (recursively) :

```
>>> concept.is_part_of(SNOMEDCT[91744000])
False
```

4.6 Other relations

The "relations" attribute contains the list of relations available for a given concept. Is-a relations are never included in this list, and are handled with the "parents" and "children" attributes previously seen, however part-of relations are included. Inverse relations are prefixed by "INVERSE_".

```
>>> concept = SNOMEDCT[3424008]
>>> concept
SNOMEDCT[3424008] # Tachycardia (finding)

>>> concept.relations
set([u'INVERSE_has_definitional_manifestation', u'finding_site', u'interprets', u'has_interpretation', u
```

Each relation corresponds to an attribute in the concept, which returns a list with the corresponding values:

```
>>> concept.finding_site
[SNOMEDCT[24964005] # Cardiac conducting system structure (body structure)
]
>>> concept.interprets
[SNOMEDCT[364075005]  # Heart rate (observable entity)
]
>>> concept.INVERSE_has_definitional_manifestation
[ SNOMEDCT[413342000]  # Neonatal tachycardia (disorder)
, SNOMEDCT[195069001]  # Paroxysmal atrial tachycardia (disorder)
, SNOMEDCT[195070000]  # Paroxysmal atrioventricular tachycardia (disorder)
,...]
```

4.7 Relation groups

In SNOMED CT, relations can be grouped together. The "groups" attribute returns the list of groups. It is then possible to access to the group's relation.

```
>>> SNOMEDCT[186675001]
SNOMEDCT[186675001]  # Viral pharyngoconjunctivitis (disorder)

>>> SNOMEDCT[186675001].groups
[<Group associated_morphology Inflammation (morphologic abnormality); finding_site Conjuncti-
val structure (body structure)>, <Group associated_morphology Inflammation (morphologic ab-
normality); finding_site Pharyngeal structure (body structure)>]

>>> SNOMEDCT[186675001].groups[0].relations
set([u'associated_morphology', u'finding_site'])
>>> SNOMEDCT[186675001].groups[0].finding_site
Concepts([
  SNOMEDCT[29445007]  # Conjunctival structure (body structure)
])
>>> SNOMEDCT[186675001].groups[0].associated_morphology
Concepts([
  SNOMEDCT[23583003]  # Inflammation (morphologic abnormality)
])
```

Relations that do not belong to a group are gathered into a "out-of-group" group (which is not included in the "groups" list).

```
>>> SNOMEDCT[186675001].out_of_group
<Group causative_agent Virus (organism); pathological_process Infectious process (quali-
fier value)>
```

4.8 Iterating over SNOMED CT

To obtain the terminology's first level concepts (i.e. the root concepts), use the first_levels() method:

```
>>> SNOMEDCT.first_levels()
[ SNOMEDCT[123037004] # Body structure (body structure)
, SNOMEDCT[404684003] # Clinical finding (finding)
, SNOMEDCT[308916002] # Environment or geographical location (environment / location)
,...]
```

The all_concepts() method returns a Python generator that iterates over all concepts in SNOMED CT.

```
>>> for concept in SNOMEDCT.all_concepts(): [...]
```

The all_concepts_no_double() method behaves similarly, but removes duplicates.

```
>>> for concept in SNOMEDCT.all_concepts_no_double(): [...]
```

4.9 CORE Problem List

The CORE Problem List is a subset of SNOMED CT appropriated for coding clinical information. The "is_in_core" attribute is true if a concept belongs to the CORE Problem List:

```
>>> concept.is_in_core
1
```

To iterate through all concepts in CORE Problem List:

```
>>> for core_concept in SNOMEDCT.CORE_problem_list(): [...]
```

4.10 Clinical signs associated to a concept

The associated_clinical_findings() method lists all clinical signs associated to an anatomical concept (a *body structure*) or a morphology, including their descendants or descendant parts. For example for listing all clinical findings affecting cardiac structures:

```
>>> SNOMEDCT[80891009]
SNOMEDCT[80891009]  # Heart structure (body structure)

>>> SNOMEDCT[80891009].associated_clinical_findings()
Concepts([
  SNOMEDCT[250981008]  # Abnormal aortic cusp (disorder)
, SNOMEDCT[250982001]  # Commissural fusion of aortic cusp (disorder)
, SNOMEDCT[250984000]  # Torn aortic cusp (disorder)
,...]
```

5 ICD10

5.1 Loading modules

```
>>> from pymedtermino import *
>>> from pymedtermino.icd10 import *
```

5.2 Concepts

The ICD10 object allows to access to ICD10 concepts. This object behaves similarly to the SNOMED CT terminology previously described.

```
>>> ICD10["E10"]
ICD10[u"E10"]  # Insulin-dependent diabetes mellitus

>>> ICD10["E10"].parents
[ICD10[u"E10-E14"]  # Diabetes mellitus
]

>>> list(ICD10["E10"].ancestors())
[ ICD10[u"E10-E14"]  # Diabetes mellitus
, ICD10[u"IV"]  # Endocrine, nutritional and metabolic diseases
]
```

ICD10 being monoaxial, the parents list includes at most one parent.

5.3 Translations

ICD10 is available in several languages. The get_translation() method returns the translation in a given language:

```
>>> print(ICD10["E10"].get_translation("fr"))
diabète sucré insulino-dépendant

>>> print(ICD10["E10"].get_translation("en"))
Insulin-dependent diabetes mellitus
```

The default language is defined by the pymedtermino.LANGUAGE global parameter (this parameter MUST be set BEFORE loading concepts).

5.4 Relations

ICD10 inclusions and exclusions can be accessed as relations.

```
>>> ICD10["E10"].relations
set([u'inclusion', u'exclusion', u'modifierlink'])

>>> ICD10["E10"].exclusion
[Text(ICD10[u"E10"]  # Insulin-dependent diabetes mellitus
, 'exclusion', u'diabetes mellitus (in) malnutrition-related E12.-
', 0, ICD10[u"E12"]  # Malnutrition-related diabetes mellitus
)...]
```

6 UMLS

6.1 Loading modules

```
>>> from pymedtermino import *
>>> from pymedtermino.umls import *
```

After importing modules, you need to connect to a MySQL database containing UMLS data, as following:

```
>>> connect_to_umls_db(host, user, password, database_name = "umls", encoding = "latin1")
```

host, user, password must be specified.

6.2 UMLS concepts (CUI)

In UMLS, CUI correspond to concepts: a given concept gathers equivalent terms or codes from various terminologies.
CUI can be accessed with the UMLS_CUI terminology:

```
>>> UMLS_CUI[u"C0085580"]
UMLS_CUI[u"C0085580"]  # Essential Hypertension (MDRJPN, SNOMEDCT, ICD10, BI, CCS, MDR-
POR, COSTAR, ICD10DUT, KCD5, RCD, MDRGER, AOD, MDRFRE, MDR-
CZE, SCTSPA, DMDICD10, ICPC2P, OMIM, MDRITA, MDR, MEDCIN, ICD10CM, MDR-
DUT, ICD10AM, MTH, CSP, MDRSPA, SNM, DXP, NCI, PSY, SNMI, ICD9CM, CCPSS)

>>> UMLS_CUI[u"C0085580"].term
u'Essential Hypertension'

>>> UMLS_CUI[u"C0085580"].terms
['Essential Hypertension', 'HYPERTENSION, ESSENTIAL', 'HYPERTENSION ESSENTIAL', 'Hyperten-
sion;essential', 'Essential hypertension, NOS', ...]

>>> UMLS_CUI[u"C0085580"].original_terminologies
set(['MDRJPN', 'SNOMEDCT', 'ICD10', 'BI', 'CCS', 'MDR-
POR', 'COSTAR', 'ICD10DUT', 'KCD5', 'RCD', 'MDRGER', 'AOD', 'MDRFRE', 'MDR-
CZE', 'SCTSPA', 'DMDICD10', 'ICPC2P', 'OMIM', 'MDRITA', 'MDR', 'MEDCIN', 'ICD10CM', 'MDR-
DUT', 'ICD10AM', 'MTH', 'CSP', 'MDRSPA', 'SNM', 'DXP', 'NCI', 'PSY', 'SNMI', 'ICD9CM', 'CCPSS'])
```

Relations of CUI are handled in the same way than for SNOMED CT (see section 4.6), for example:

```
>>> UMLS_CUI[u"C0085580"].relations
set(['has_finding_site', 'INVERSE_translation_of', 'SIB', 'IN-
VERSE_has_alias', 'may_be_a', None, 'RQ', 'INVERSE_mapped_from',...])

>>> UMLS_CUI[u"C0085580"].has_finding_site
[UMLS_CUI[u"C0459964"]  # Systemic arterial structure (RCD, SCTSPA, SNOMEDCT)
```

6.3 UMLS concept form source terminologies (AUI)

The UMLS_AUI terminology allows to access to UMLS atoms. A UMLS atom corresponds to a concept in a given source terminology; e.g. "type 2 diabetes in ICD10" is a different atom from "type 2 diabetes in SNOMED CT".

```
>>> UMLS_AUI[u"A0930328"]
UMLS_AUI[u"A0930328"] # Essential (primary) hypertension (ICD10)

>>> UMLS_AUI[u"A0930328"].original_terminologies
set(['ICD10'])
```

6.4 Extracting terminologies from UMLS

PyMedTermino can extract terminologies from UMLS, and use them with the source terminology codes (rather than AUI), for example to extract SNOMED CT, ICD10 and ICPC2 :

```
>>> UMLS_SNOMEDCT  = UMLS_AUI.extract_terminology("SNOMEDCT", has_int_code = 1)
>>> UMLS_ICD10     = UMLS_AUI.extract_terminology("ICD10")
>>> UMLS_ICPC2EENG = UMLS_AUI.extract_terminology("ICPC2EENG")
```

The first parameter of the UMLS_AUI.extract_terminology() function is the name of the terminology to extract (they can be found in the list of UMLS sources). The optional parameter "has_int_code = 1" indicates that the codes of the source terminology are numeric; this allows to remove quote around them.

Extracted terminologies can be used as usual:

```
>>> UMLS_ICD10["I10"]
UMLS_ICD10[u"I10"]  # Essential (primary) hypertension (ICD10)
```

It is possible to access to relations (when they exist) like previously.

6.5 Mapping between UMLS terminologies

PyMedTermino automatically defines mapping between terminologies extracted from UMLS, for example:

```
>>> UMLS_ICD10["I10"] >> UMLS_SNOMEDCT
Concepts([
  UMLS_SNOMEDCT[u"59621000"]  # Essential hypertension (SNOMEDCT)
])
```

For more information on mapping in PyMedTermino, see section 8.

7 VCM

7.1 Loading modules

```
>>> from pymedtermino import *
>>> from pymedtermino.vcm import *
```

Databases describing VCM terminologies are already included with PyMedTermino.

7.2 VCM icons

The VCM object is a terminology for accessing VCM icons, identified by their code, in French or English:

```
>>> icon = VCM["en_cours--patho--coeur"]
>>> icon = VCM["current--patho--heart"]
>>> icon =  VCM["en_cours--patho-vaisseau--coeur--traitement--medicament--rien--rien"]
```

The icon code includes up to 7 components, separated by two dashes (–):

1. The central color
2. The shape modifier(s) (separated by a single dash if there are several of them)
3. The central pictogram
4. The top-right color
5. The top-right pictogram
6. The second top-right pictogram
7. The shadow

The possible values for each component are listed in the graphical lexicon (see the VCM pictogram lexicon, or the VCM_LEXICON terminology below). Missing components in the code of the icon are replaced by "empty".

Various attributes return the icon's components:

```
>>> icon.central_color
VCM_LEXICON[496] # Red_color
>>> icon.modifiers
Concepts([
  VCM_LEXICON[536]  # Modifier_vessel
, VCM_LEXICON[504]  # Modifier_patho
])
>>> icon.central_pictogram
VCM_LEXICON[549]  # Pictogramme_heart
>>> icon.central_pictogram.text_code
```

```
heart

>>> icon.top_right_color
VCM_LEXICON[690]  # Green_color
>>> icon.top_right_pictogram
VCM_LEXICON[697]  # Drug_top_right_pictogram
>>> icon.second_top_right_pictogram
VCM_LEXICON[718]  # No_second_top_right_pictogram
>>> icon.shadow
VCM_LEXICON[722]  # No_shadow
```

The "lexs" attribute returns a set with all the components of the icon:

```
>>> icon.lexs
Concepts([
  VCM_LEXICON[536]  # Modifier_vessel
, VCM_LEXICON[549]  # Pictogramme_heart
, VCM_LEXICON[722]  # No_shadow
, VCM_LEXICON[496]  # Red_color
, VCM_LEXICON[504]  # Modifier_patho
, VCM_LEXICON[718]  # No_second_top_right_pictogram
, VCM_LEXICON[697]  # Drug_top_right_pictogram
, VCM_LEXICON[690]  # Green_color
])
```

The following attributes returns the shape modifiers of a specific category: pathological modifiers, etiology,...:

```
>>> icon.physio
>>> icon.patho
>>> icon.etiology
>>> icon.quantitative
>>> icon.process
>>> icon.transverse
```

The "consistent" attribute is True if the icon is consistent (according to the VCM ontology, as described in this article: J-B Lamy et al., Validating the semantics of a medical iconic language using ontological reasoning[1]):

```
>>> icon.consistent
True
```

7.3 Graphical lexicon

The VCM_LEXICON terminology describes the lexicon of the VCM graphical primitives: pictograms, colors and shapes. Each primitive is identified by an arbitrary numeric code, for example for the heart pictogram:

```
>>> heart = VCM_LEXICON[549]
>>> heart
VCM_LEXICON[549] # Pictogramme_heart
```

Each concept of the lexicon also has a textual code (easier to memorize, and available in French and English), and a category:

```
>>> heart.text_code
u'heart'
>>> heart.text_codes
[u'heart', u'coeur']
>>> heart.category
2
```

The categories correspond to the various parts of the VCM icons:

0 Central color

1 Shape modifier

[1] J-B Lamy et al., Validating the semantics of a medical iconic language using ontological reasoning, Journal of Biomedical Informatics 2013, 46(1):56-67
http://www.sciencedirect.com/science/article/pii/S153204641200130X

2 Central pictogram

3 Top-right color

4 Top-right pictogram

5 Second top-right pictogram

6 Shadow

You can also use the category and the textual code to obtain a lexicon concept:

```
>>> VCM_LEXICON[2, "heart"]
VCM_LEXICON[549] # Pictogramme_heart
```

Relations are handled as usual in (see the section about SNOMED CT: parents, children, is_a(), ancestors(), descendants(),...). In addition the graphical_is_a relation indicates the other graphical primitive that are reused by th the lexicon concept. For example the heart rhythm pictogram reuse the heart pictogram:

```
>>> heart_rhythm = VCM_LEXICON[2, "heart_rhythm"]
>>> heart_rhythm.graphical_is_a
[VCM_LEXICON[549]  # Pictogramme_heart
]
```

The "graphical_children" and "graphical_parents" attributes return the list of lexicon concepts that re-use or are reused by the concept.

7.3.1 Creating a VCM icon from lexicon concepts

A set of lexicon concepts can be assembled into a VCM icon:

```
>>> Concepts([VCM_LEXICON[549], VCM_LEXICON[496], VCM_LEXICON[504]]) >> VCM
Concepts([
  VCM[u"current--patho--heart"]  #
])
```

7.4 Medical concepts

VCM_CONCEPT is a terminology that represents the medical concepts described by VCM. Each medical concept is defined by an arbitrary numeric code, for example for the heart:

```
>>> heart = VCM_CONCEPT[266]
>>> heart
VCM_CONCEPT[266] # Cardiac_structure
```

Relations are handled as usual in PyMedTermino (see the section about SNOMED CT: parents, children, is_a(), ancestors(), descendants(), relations...).

VCM_CONCEPT_MONOAXIAL is a terminology identical to VCM_CONCEPT, but monoaxial. The concepts are thus the same, but with at maximum a single parent for each concept. This terminology is mostly used in intern for mapping from VCM_CONCEPT (multiaxial) to VCM_LEXICON (monoaxial).

8 Mappings

A mapping allows to transcode one or more concepts from a source terminology to a destination terminology. PyMedTermino uses the >> operator for mapping, in the following way:

```
concept(s) >> DESTINATION_TERMINOLOGY
```

where concept(s) can be a concept of the source terminology, or a set of concepts (see section 9). The >> operator returns a set of concepts in the destination terminology. The >> operators can thus be chained:

```
concept(s) >> INTERMEDIARY_TERMINOLOGY >> DESTINATION_TERMINOLOGY
```

PyMedTermino includes several mappings, described in the following subsections.

8.1 UMLS mappings

8.1.1 UMLS_CUI <=> UMLS_AUI

PyMedTermino can map CUI to AUI, and vice versa:

```
>>> UMLS_CUI[u"C0085580"] >> UMLS_AUI
Concepts([
  UMLS_AUI[u"A16015049"]  # Hypertension primitive (MDRFRE)
, UMLS_AUI[u"A11101884"]  # Hypertension essentielle, non précisée (MDRFRE)
, UMLS_AUI[u"A11089284"]  # Hypertension essentielle non précisée (MDRFRE)
...])
```

8.1.2 Terminology extracted from UMLS <=> CUI or AUI

PyMedTermino can map concepts of terminology extracted from UMLS to CUI or AUI, and vice versa:

```
>>> UMLS_ICD10["I10"] >> UMLS_CUI
Concepts([
  UMLS_CUI[u"C0085580"]  # Essential Hypertension (MDRJPN, SNOMEDCT, ICD10, BI, CCS, MDR-
POR, COSTAR, ICD10DUT, KCD5, RCD, MDRGER, AOD, MDRFRE, MDR-
CZE, SCTSPA, DMDICD10, ICPC2P, OMIM, MDRITA, MDR, MEDCIN, ICD10CM, MDR-
DUT, ICD10AM, MTH, CSP, MDRSPA, SNM, DXP, NCI, PSY, SNMI, ICD9CM, CCPSS)
])
```

8.1.3 Terminology extracted from UMLS <=> source terminology

PyMedTermino can map concepts of terminology extracted from UMLS to the source terminology, and vice versa:

```
>>> ICD10["I10"] >> UMLS_ICD10
Concepts([
  UMLS_ICD10[u"I10"]  # Essential (primary) hypertension (ICD10)
])
```

8.1.4 Terminology extracted from UMLS <=> another terminology extracted from UMLS

PyMedTermino automatically create mapping between the terminologies extracted from UMLS with UMLS_AUI.extract_terminology():

```
>>> UMLS_ICD10["I10"] >> UMLS_SNOMEDCT
Concepts([
  UMLS_SNOMEDCT[u"59621000"]  # Essential hypertension (SNOMEDCT)
])
```

8.2 SNOMEDCT <=> VCM

This mapping maps SNOMED CT concepts to (or from) VCM icons. It has been built automatically from the SNOMEDCT <=> VCM_CONCEPT and VCM_CONCEPT <=> VCM_LEXICON mappings (as described in this article: J-B Lamy et al., A Semi-automatic Semantic Method for Mapping SNOMED CT Concepts to VCM Icons [2]).

```
>>> from pymedtermino.snomedct_2_vcm import *

>>> SNOMEDCT[3424008]
SNOMEDCT[3424008]  # Tachycardia (finding)

>>> SNOMEDCT[3424008] >> VCM
Concepts([
  VCM[u"current--hyper--heart_rhythm"]  #
])
```

[2] J-B Lamy et al., A Semi-automatic Semantic Method for Mapping SNOMED CT Concepts to VCM Icons, Studies in health technology and informatics 2013, 192:42-6
http://ebooks.iospress.nl/publication/33954

8.3 VCM_LEXICON => VCM

A set of VCM lexicon element (pictogram, color,...) can be assembled into a VCM icon:

```
>>> Concepts([VCM_LEXICON[549], VCM_LEXICON[496], VCM_LEXICON[504]]) >> VCM
Concepts([
  VCM[u"current--patho--heart"]  #
])
```

8.4 VCM_CONCEPT <=> VCM_LEXICON

This mapping maps VCM medical concepts to (or from) VCM lexicon elements. It has been built manually, and is part of the VCM ontology.

```
>>> VCM_CONCEPT[266] >> VCM_LEXICON
Concepts([
  VCM_LEXICON[549]  # Pictogramme_heart
])
>>> VCM_LEXICON[549] >> VCM_CONCEPT
Concepts([
  VCM_CONCEPT[266]  # Cardiac_structure
, VCM_CONCEPT[102]  # Cardiac_function
])
```

8.5 SNOMEDCT <=> VCM_CONCEPT

This mapping maps SNOMED CT concepts (mostly body structures and morphologies) to (or from) VCM medical concepts. It has been built manually.

```
>>> SNOMEDCT[302509004]
SNOMEDCT[302509004] # Entire heart (body structure)

>>> SNOMEDCT[302509004] >> VCM_CONCEPT
Concepts([
  VCM_CONCEPT[266] # Cardiac_structure
, VCM_CONCEPT[239] # Thorax_region
])
```

8.6 Examples

By chaining several mapping, it is possible to map an ICD10 concept to SNOMED CT via UMLS:

```
>>> ICD10["I10"] >> UMLS_ICD10 >> UMLS_SNOMEDCT >> SNOMEDCT
Concepts([
  SNOMEDCT[59621000]  # Essential hypertension (disorder)
])
```

If you want to use this method as a default mapping from ICD10 to SNOMED CT, you can register this mapping as following:

```
>>> (ICD10 >> UMLS_ICD10 >> UMLS_SNOMEDCT >> SNOMEDCT).register()

>>> ICD10["I10"] >> SNOMEDCT
Concepts([
  SNOMEDCT[59621000]  # Essential hypertension (disorder)
])
```

9 Set of concepts

The Concepts() class represents set of concepts. Such a set can contain each concept only once, and it inherits from Python's set() the methods for computing intersection, union, difference, ..., of two sets. It also proposes the following methods:

- Concepts.find(parent_concept) : returns the first concept of the set that is a descendant of concept_parent (including concept_parent itself).
- Concepts.extract(parent_concept) : returns all concepts of the set that are descendant of concept_parent (including concept_parent itself).

- Concepts.subtract(parent_concept) : returns a new set after removing all concepts that are descendant of parent_concept (including concept_parent itself).
- Concepts.subtract_update(parent_concept) : same as subtract(), but modify the set "in place".
- Concepts.imply(concepts2) : returns true if all concepts in the set are descendants of at least one of the concept in the set concepts2.
- Concepts.keep_most_specific() : keeps only the most specific concepts, i.e. remove all concepts that are more general that another concept in the set.
- Concepts.keep_most_generic() : keeps only the most general concepts, i.e. remove all concepts that are more specific that another concept in the set.
- Concepts.lowest_common_ancestors() : returns the lowest common ancestors.
- Concepts.all_subsets() : returns all the subsets included in the set.

10 Using PyMedTermino without Python

PyMedTermino can also be used without Python, simply for converting SNOMED CT and ICD10 XML data into SQL database. The SQLite3 databases created can then be interrogated with most programming language, however you won't have access to high level functions proposed by PyMedTermino (such as the ancestors() and descendants() functions).

The definition of the tables of the databases can be found in the scripts/import_sonmedct.py and scripts/import_icd10.py files.