cTakes Dictionary Lookup Module 2 Help

3-04-2014

# Lookup Workflow

## No Lookup Initializer

Unlike the current dictionary module, the new one has no lookup initializer that sits apart from the lookup annotator. All functionality of the current module’s Initializers resides in the new module’s Annotator. This removes some of the “pluggability” that exists for the current module, but greatly reduces complexity and overhead.

## Lookup per Registered Dictionary per Lookup Window

For each proper lookup token encountered in the lookup window, the new dictionary lookup code runs through a loop that performs lookup per each registered Dictionary, finding terms based upon the lookup token per dictionary. All Lookups are performed in the same manner, just using different term repositories (dictionaries).

## Term Match

Given the full text for all candidate terms in a dictionary, a match is attempted using that text on the full text of the lookup window.

## Single Lookup Consumer

After all terms have been appropriated from all dictionaries, a single Consumer takes the collection of terms and performs some action(s) upon them, such as storing them in the Cas. A Consumer (or delegates) can be created to differently handle terms with different Cuis, Tuis, words, entity types, schemes, etc. if that is necessary, but there is one consumer entry point, called once with the entire collection of discovered terms from all dictionaries.

Term Annotator

Annotator Descriptor XML

UIMA

UIMA Context

Initialize

Dictionary Specification XML

Dictionary Specification Parser

- Term Annotator Implementation Class

- Lookup Token POS Exclusions

- Lookup Token minimum Span Length

- Dictionary Resource URL(s)

- Dictionary Specification XML URL

- Dictionary Table(s)

- Term Consumer Implementation Class

**\***

Term Consumer

**\***

CAS

L.W.

Dict.

Get L.W.s

L.T.

Find L.T.s

Find Terms in Dictionary containing Lookup Token

Terms

- Term Table

- Rare Word Index

Match Term Text in L.W.

Matched Terms

Create Annotations

Term Annotator

Window Processor

Term

Consumer

Ann’ns

# Lookup Methods

## Lookup by Rare Word

Each dictionary entry, UMLS or other, contains the rare word used for the lookup key, a count of words (tokens) within the term, and an index of the rare word within the term. For each lookup token (word of proper part of speech and span length) in the note, the lookup code queries the dictionary for terms with a matching rare word. For each returned term, a match is attempted using the term and the appropriate token coverage in the note based upon the offset indicated by the rare word index and length of the term. This creates a forward and backward matching method instead of the forward-only matching method of the current dictionary (first word) lookup. Using a term’s rare word as an index instead of a first word decreases, on average, the number of terms for which matching must be performed.

## Text Exact Match

Because the UMLS dictionary contains rows with different combinations of lexical elements per term, using a direct string match of text in note to text of term is a valid candidate for term matching. This is different from the complex mechanism in the current (first word) lookup, and makes for simpler code and greater accuracy. This precise specification (and improved lookup speed) enables the use of an entire sentence as a lookup window rather than just a noun phrase. Usage of Sentence as a lookup window allows all possible tokens to be used for not only lookup keys, but also for term matching. For proper accuracy, custom dictionaries should also contain multiple entries for variations of term syntax. Note that term matching is attempted using the actual text in the note and also per-token cTakes-generated lexical variants of the text in the note.

## Text Overlap Match

To better approximate the current (first word) lookup annotator, one lookup method finds overlapping terms in addition to exact matching terms. This allows matches on discontiguous spans. For instance, for the text “blood, urine test” the exact match will find only one procedure: “urine test”. The overlap match will find both “urine test” and “blood test”.

# Persistence Methods (Consumer)

## All Terms Persistence

All terms discovered by the matchers can be stored in the CAS by a consumer, regardless of any property of the term. This means that for the text “lung cancer” the specific disease term “lung cancer” and broader term “cancer”. This can be useful for future searches on general concepts, e.g. searching via the CUI for “cancer” and getting all instances of “cancer” found in texts “lung cancer”, “skin cancer”, “stomach cancer”, etc.

## Most Precise Term Persistence

Matched terms can be stored only by the longest overlapping span discovered for a semantic group. This keeps, for instance, the disease “lung cancer” but not “cancer”. Using semantic groups means that both the disease “lung cancer” and the anatomical site “lung” are persisted even though the spans overlap. When using the overlap matching method, any discontiguous spans are accounted for. So, for “blood, urine test” both the discontiguous spanned term “blood test” and the contiguous spanned term “urine test” are valid.

# Dictionary Indexing

## Lookup by Rare Word

No matter what text Match algorithm is used, it must run for every Term’s text returned from the dictionary lookup of a Lookup Token. To minimize the number of Terms returned from the dictionary lookup, a Dictionary should be indexed by a Rare Word for each Term. The Rare Word would be a token of proper Lookup Token Part of Speech in the Term Text which has the minimum number of instances in the entire Dictionary. Some examples (‘-‘ denotes a non-word):

|  |  |  |  |
| --- | --- | --- | --- |
| **Term Text** | **Word Counts in Dictionary Texts** | **Rare Word in Index** | **Rare Word Count in Index** |
| Medical care vascular complications | 11 3 1 1 | vascular | 1 |
| Medical / dental care | 11 - 2 3 | dental | 1 |
| Medical / dental care treatments and procedures | 11 - 2 3 1 - 1 | treatments | 1 |
| Medical imaging | 11 2 | imaging | 1 |
| Medical imaging , ultrasound | 11 2 - 1 | ultrasound | 1 |
| Medical identification band on | 11 3 1 - | band | 1 |
| Medical identification bracelet on | 11 1 2 - | bracelet | 1 |
| Medical identification bracelet present | 11 3 2 1 | present | 1 |
| Medical procedure on the nervous system | 11 3 - - 3 3 | procedure | 1 |
| Medical procedure on the central nervous system | 11 3 - - 1 3 3 | central | 1 |
| Medical procedure on the peripheral nervous system | 11 3 - - 1 3 3 | peripheral | 1 |

For any Lookup Window and the given Dictionary, any Lookup Token “Medical” would return zero candidate terms and require no Match processing. Only when a Lookup Token exists as a Rare Word in a Term will it exist in the Dictionary index and return one or more candidate Terms upon which to attempt a Match. Sometimes this will be the first Token in a Term, but often it is not. Because a Lookup Token can be in the middle of a Term’s text, the Match must be attempted by checking words both after and before the Lookup Token in the Lookup Window text.

# Example Annotator Descriptor XML

The following Descriptor XML is in the module’s example directory and can be used to perform lookup using a Bar-Separated Value flat file as the dictionary.

<taeDescription xmlns="http://uima.apache.org/resourceSpecifier">

<frameworkImplementation>org.apache.uima.java</frameworkImplementation>

<primitive>true</primitive>

<annotatorImplementationName>

**org.apache.ctakes.dictionary.lookup2.ae.DefaultJCasTermAnnotator**

</annotatorImplementationName>

The above specifies that the default lookup annotator implementation should be used. The default lookup only uses contiguous text spans. To specify usage of the lookup annotator that allows terms on discontiguous spans, use:

<annotatorImplementationName>

**org.apache.ctakes.dictionary.lookup2.ae.OverlapJCasTermAnnotator**

</annotatorImplementationName>

In the previous Dictionary Lookup Module, the type of lookup window and the excluded parts of speech were not specified in the analysis engine descriptor, they were specified in a secondary lookup specification file. For the new module this information is in the analysis engine descriptor.

<analysisEngineMetaData>

*<!-- Simple descriptor that uses DefaultJCasTermAnnotator and Bsv Dictionary File -->*

<name>**CustomLookupAnnotatorBSV**</name>

<description>

**Dictionary Lookup Annotator descriptor for dictionaries which actually lie in a BSV file**

</description>

<version/>

<vendor/>

<configurationParameters>

*<!-- windowAnnotations and exclusionTags were originally for the LookupConsumer, but now apply to the annotator -->*

<configurationParameter>

<name>**windowAnnotations**</name>

<description>**Type of window to use for lookup**</description>

<type>String</type>

<multiValued>false</multiValued>

<mandatory>true</mandatory>

</configurationParameter>

<configurationParameter>

<name>**exclusionTags**</name>

<description>

**Parts of speech to ignore when considering lookup tokens**

</description>

<type>String</type>

<multiValued>false</multiValued>

<mandatory>false</mandatory>

</configurationParameter>

<configurationParameter>

<name>**minimumSpan**</name>

<description>

**Minimum required span length of tokens to use for lookup. Default is 3**

</description>

<type>Integer</type>

<multiValued>false</multiValued>

<mandatory>false</mandatory>

</configurationParameter>

</configurationParameters>

<configurationParameterSettings>

<nameValuePair>

<name>**windowAnnotations**</name>

<value>

*<!-- LookupWindowAnnotation is supposed to be a refined Noun Phrase -->*

<string>

**org.apache.ctakes.typesystem.type.textspan.LookupWindowAnnotation**

</string>

</value>

</nameValuePair>

<nameValuePair>

<name>**exclusionTags**</name>

<value>

<string>

**VB,VBD,VBG,VBN,VBP,VBZ,CC,CD,DT,EX,IN,LS,MD,PDT,POS,PP,PP$,PRP,PRP$,**

**RP,TO,WDT,WP,WPS,WRB**

</string>

</value>

</nameValuePair>

<nameValuePair>

<name>**minimumSpan**</name>

<value>

<integer>**3**</integer>

</value>

</nameValuePair>

</configurationParameterSettings>

The above specifies that the lookup window annotation should be a LookupWindowAnnotation, which is supposed to represent noun phrases in the text. A larger, more inclusive lookup window would be a sentence. To specify usage of a sentence as the lookup window, use:

<nameValuePair>

<name>**windowAnnotations**</name>

<value>

*<!-- In some instances LookupWindowAnnotation is missing tokens , but*

*Sentence can be used -->*

<string>**org.apache.ctakes.typesystem.type.textspan.Sentence**</string>

</value>

</nameValuePair>

If the discontiguous span annotator (OverlapJCasTermAnnotator) is used, then two additional configuration parameters are available. These parameters are optional and modify the total number of non-matching tokens that can be skipped and the number of consecutive non-matching tokens that can be skipped.

<configurationParameter>

<name>**consecutiveSkips**</name>

<description>

**Number of consecutive tokens that can be skipped when matching terms. Default is 2**

</description>

<type>Integer</type>

<multiValued>false</multiValued>

<mandatory>false</mandatory>

</configurationParameter>

<configurationParameter>

<name>**totalTokenSkips**</name>

<description>

**Total number of tokens that can be skipped when matching terms. Default is 4**

</description>

<type>Integer</type>

<multiValued>false</multiValued>

<mandatory>false</mandatory>

</configurationParameter>

…

<nameValuePair>

<name>**consecutiveSkips**</name>

<value>

<integer>**2**</integer>

</value>

</nameValuePair>

<nameValuePair>

<name>**totalTokenSkips**</name>

<value>

<integer>**4**</integer>

</value>

</nameValuePair>

What follows is exactly the same as the description used for the previous dictionary lookup module descriptor.

<typeSystemDescription>

<imports>

</imports>

</typeSystemDescription>

<typePriorities/>

<fsIndexCollection/>

<capabilities>

<capability>

<inputs>

<type allAnnotatorFeatures="true">

org.apache.ctakes.typesystem.type.syntax.BaseToken

</type>

<type allAnnotatorFeatures="true">

org.apache.ctakes.typesystem.type.textspan.LookupWindowAnnotation

</type>

</inputs>

<outputs>

<type allAnnotatorFeatures="true">

org.apache.ctakes.typesystem.type.textsem.IdentifiedAnnotation

</type>

</outputs>

<languagesSupported/>

</capability>

</capabilities>

<operationalProperties>

<modifiesCas>true</modifiesCas>

<multipleDeploymentAllowed>true</multipleDeploymentAllowed>

<outputsNewCASes>false</outputsNewCASes>

</operationalProperties>

</analysisEngineMetaData>

The resource manager configuration sets up the external resources used by the dictionary lookup module. There must be a url to a dictionary specification descriptor and a url to each dictionary resource. What follows is exactly as it was for the previous lookup module, except that the external resource dependency LookupDescriptor has been replaced by DictionaryDescriptor.

<externalResourceDependencies>

*<!—* *DictionaryDescriptor is the key for the file that contains dictionary specs-->*

<externalResourceDependency>

<key>**DictionaryDescriptor**</key>

<description/>

<interfaceName>org.apache.ctakes.core.resource.FileResource</interfaceName>

optional>false</optional>

</externalResourceDependency>

*<!-- CustomBsvDictionary is the key for the actual bsv file that contains the dictionary -->*

*<!-- This is not a filename, just a key to connect what is here with what is in the LookupDescriptor -->*

<externalResourceDependency>

<key>**CustomBsvDictionary**</key>

<description/>

<interfaceName>org.apache.ctakes.core.resource.FileResource</interfaceName>

<optional>false</optional>

</externalResourceDependency>

</externalResourceDependencies>

<resourceManagerConfiguration>

<externalResources>

<externalResource>

*<!-- The Binding is below, for DictionaryDescriptor = DictionaryDescriptorFile -->*

<name>**DictionaryDescriptorFile**</name>

<description/>

<fileResourceSpecifier>

<fileUrl>

**file:org/apache/ctakes/dictionary/lookup2/example/bsv/RareWord\_BSV.xml**

</fileUrl>

</fileResourceSpecifier>

<implementationName>

org.apache.ctakes.core.resource.FileResourceImpl

</implementationName>

</externalResource>

<externalResource>

*<!-- The Binding is below, for CustomBsvDictionary = BsvDictionaryResource -->*

<name>**BsvDictionaryResource**</name>

<description/>

<fileResourceSpecifier>

<fileUrl>

**file:org/apache/ctakes/dictionary/lookup2/example/bsv/CustomBsvDictionary.bsv**

</fileUrl>

</fileResourceSpecifier>

<implementationName>

org.apache.ctakes.core.resource.FileResourceImpl

</implementationName>

</externalResource>

</externalResources>

<externalResourceBindings>

<externalResourceBinding>

<key>**DictionaryDescriptor**</key>

<resourceName>**DictionaryDescriptorFile**</resourceName>

</externalResourceBinding>

<externalResourceBinding>

*<!-- binding key CustomBsvDictionary points up to the*

*externalResource named BsvDictionaryResource -->*

<key>**CustomBsvDictionary**</key>

<resourceName>**BsvDictionaryResource**</resourceName>

</externalResourceBinding>

</externalResourceBindings>

</resourceManagerConfiguration>

</taeDescription>

# Example Dictionary Specification Descriptor XML

The following Descriptor XML is in the module’s resource example directory and can be used to perform lookup using a Bar-Separated Value flat file as the dictionary.

<lookupSpecification>

*<!-- Defines what dictionaries will be used -->*

<rareWordDictionaries>

*<!-- typeId 0 is the cTakes constant for unknown - let the consumer take care of typing -->*

<dictionary id="**customBsv**" externalResourceKey="**CustomBsvDictionary**" caseSensitive="false" typeId="0">

<implementation>

<**rareWordBsv**/>

</implementation>

</dictionary>

</rareWordDictionaries>

*<!-- DefaultTermConsumer will persist all spans -->*

<rareWordConsumer className="**org.apache.ctakes.dictionary.lookup2.consumer.DefaultTermConsumer**">

<properties>

<property key="codingScheme" value="cTakes"/>

</properties>

</rareWordConsumer>

</lookupSpecification>

To only persist only the longest overlapping term of any given span for each semantic group, use the PrecisionTermConsumer.

*<!-- PrecisionTermConsumer will only persist only the longest overlapping span of any semantic group -->*

<rareWordConsumer className="**org.apache.ctakes.dictionary.lookup2.consumer.PrecisionTermConsumer**">

<properties>

<property key="codingScheme" value="cTakes"/>

</properties>

</rareWordConsumer>