CQL to HTML Code Generator Program

# Purpose:

The process of rendering CQL code into an HTML is very tedious work. In an effort to simplify the process, I have designed a program in Python that automates the process. Note that this program assumes that the CQL is written using the QDM data model.

In addition, this is not a production level generator in that it makes certain assumptions about whitespace and formatting of the source CQL. A production quality generator would need to work with arbitrary CQL, and would likely be based on an Antlr-generated CQL visitor.

# Challenges:

## Divs

A difficult aspect of the translation has to do with keeping track of divs. The divs keep the code neat and control the indentation to keep the code readable. So every time a new level of indentation is encountered, a new div must be established. Take the following code snippet as an example:

define EncounterInpatient:

["Encounter, Performed": "Encounter Inpatient"] E

where days between E."admissionDatetime" and E."dischargeDatetime" <= 120

and E."dischargeDatetime" in MeasurementPeriod

The line “define EncounterInpatient:” is placed inside a div. The following line “["Encounter, Performed": "Encounter Inpatient"] E” is also placed within a div that is nested within the “define EncounterInpatient:” div. Then the last two lines are also placed within a div that is nested within the previous two divs.

The challenge, then is how to recognize when a new div is needed and when to close a div as well. The solution to this problem is keeping track of whitespace. Let us look at the previous code snippet as it would look unformatted:

define EncounterInpatient:\n

\t["Encounter, Performed": "Encounter Inpatient"] E\n

\t\twhere days between E."admissionDatetime" and E."dischargeDatetime" <= 120

\t\tand E."dischargeDatetime" in MeasurementPeriod

By studying this snippet briefly, the pattern becomes apparent that each time a preceding tab (\t) is encountered that is greater than the previous preceding tab, we need a new div. By keeping a running tab count, it is simple to determine when a new div is needed and whether a div needs to be closed.

## CQL Patterns

The following cases must be accounted for:

***Named Expressions*** – a named expression is defined, in this program, as an identifier that immediately follows the “define” keyword, which also represents the pattern for discovery. For example, in the snippet above, EncounterInpatient is the named expression’s name (named expressions could also be strings e.g. “Encounter Inpatient” is valid). Named expressions are given an id to reference later by reference links.

***Alias*** –instances of a named expression. In the above snippet, “E” is the alias referencing “["Encounter, Performed": "Encounter Inpatient"]”. The pattern then is that alias’ are almost always a single uppercase letter that either follow a reference/property declaration or precede a dot operation (e.g. E."admissionDatetime").

***Function*** – CQL functions will be read as named expressions.

***Keyword*** – most common case. Use a predefined dictionary of reserved keywords to discover keywords as no obvious pattern presents itself.

***Variable*** – a container to hold data. The pattern that is assumed by this program is that variables are camel-case (e.g. camelCase), on their own line, and the variable name is immediately followed by a colon. For example:

endDate: X.”endResult”.”resultDate”

***Property*** - returns the value of an element of a structured value. The pattern here is that properties are always enclosed within square brackets ([ … ]) and the two arguments inside are enclosed within parenthesis and separated by a colon. For example:

“["Encounter, Performed": "Encounter Inpatient"]”

***Reference*** – references a named expression. The pattern here is exactly that of a named expressions except that the reference is never preceded by a “define” keyword.

***Literal*** - Literals allow basic values to be represented within the language. Literals are typically numeric values. However, not all numeric values are literals. Numeric values within a string, reference name, or property for instance, are not literal values. Other literal values include floating point numbers, strings enclosed within single quotes (‘ ‘), an @ symbol followed by a representation of time. The pattern, then, is that the first character is either a numeric digit, a single quote, or an @ symbol.

***Operator*** – operators include the following symbols:

: [] () {} <> . , = + - / \*

No pattern here. Use a dictionary for the discovery of operators.

***String*** – any text within parenthesis that is not within a property ([ ]) or preceded by the “define” keyword. So, any string that is not a named expression and not in a property is the pattern.

***Comment*** – 2 cases: inline and multiline. Inline pattern is anything preceded by double backslash (//). Multiline pattern is anything between a backslash-star (/\*) and a star-backslash (\*/). The text in these patterns are ignored.

# Method:

## Usage

From the command line:

$ python showCQLinHTML.py <file path for .cql file to translate> <name of named expression to translate> <name of heading for the generated snippet>

Example:

CQL file to translate (CMS126v4.cql):

library CMS126\_QDM version '4'

using QDM

valueset "Acute Respiratory Failure": '2.16.840.1.113883.3.464.1003.102.12.1018'

valueset "Chronic Obstructive Pulmonary Disease": '2.16.840.1.113883.3.464.1003.102.12.1007'

valueset "Cystic Fibrosis": '2.16.840.1.113883.3.464.1003.102.12.1002'

valueset "Emphysema": '2.16.840.1.113883.3.464.1003.102.12.1004'

valueset "Persistent Asthma": '2.16.840.1.113883.3.464.1003.102.12.1023'

valueset "Face-to-Face Interaction": '2.16.840.1.113883.3.464.1003.101.12.1048'

valueset "Home Healthcare Services": '2.16.840.1.113883.3.464.1003.101.12.1016'

valueset "Office Visit": '2.16.840.1.113883.3.464.1003.101.12.1001'

valueset "Preventive Care - Established Office Visit, 0 to 17": '2.16.840.1.113883.3.464.1003.101.12.1024'

valueset "Preventive Care Services - Established Office Visit, 18 and Up": '2.16.840.1.113883.3.464.1003.101.12.1025'

valueset "Preventive Care Services-Initial Office Visit, 18 and Up": '2.16.840.1.113883.3.464.1003.101.12.1023'

valueset "Preventive Care- Initial Office Visit, 0 to 17": '2.16.840.1.113883.3.464.1003.101.12.1022'

valueset "Preferred Asthma Therapy": '2.16.840.1.113883.3.464.1003.196.12.1212'

parameter MeasurementPeriod Interval<DateTime>

context Patient

define AcuteRespiratoryFailureDiagnoses: ["Diagnosis, Active": "Acute Respiratory Failure"]

define ChronicObstructivePulmonaryDiseaseDiagnoses: ["Diagnosis, Active": "Chronic Obstructive Pulmonary Disease"]

define CysticFibrosisDiagnoses: ["Diagnosis, Active": "Cystic Fibrosis"]

define EmphysemaDiagnoses: ["Diagnosis, Active": "Emphysema"]

define PersistentAsthmaDiagnoses: ["Diagnosis, Active": "Persistent Asthma"]

define FaceToFaceEncounter: ["Encounter, Performed": "Face-to-Face Interaction"]

define HomeHealthcareEncounter: ["Encounter, Performed": "Home Healthcare Services"]

define OfficeVisitEncounter: ["Encounter, Performed": "Office Visit"]

define EstablishedPreventiveCareForMinorEncounter: ["Encounter, Performed": "Preventive Care - Established Office Visit, 0 to 17"]

define EstablishedPreventiveCareForAdultEncounter: ["Encounter, Performed": "Preventive Care Services - Established Office Visit, 18 and Up"]

define InitialPreventiveCareForAdultEncounter: ["Encounter, Performed": "Preventive Care Services-Initial Office Visit, 18 and Up"]

define InitialPreventiveCareForMinorEncounter: ["Encounter, Performed": "Preventive Care- Initial Office Visit, 0 to 17"]

define PreferredAsthmaTherapy: ["Medication, Dispensed": "Preferred Asthma Therapy"]

define InDemographic:

AgeInYearsAt(start of MeasurementPeriod) >= 5

and AgeInYearsAt(start of MeasurementPeriod) < 64

define ActivePersistentAsthmaDiagnosis:

PersistentAsthmaDiagnoses A where Interval[A."start datetime", A."stop datetime"] overlaps MeasurementPeriod

define ValidEncounters:

OfficeVisitEncounter

union HomeHealthcareEncounter

union FaceToFaceEncounter

union EstablishedPreventiveCareForMinorEncounter

union EstablishedPreventiveCareForAdultEncounter

union InitialPreventiveCareForAdultEncounter

union InitialPreventiveCareForMinorEncounter

define EncountersDuringMeasurementPeriod:

ValidEncounters E where Interval[E."start datetime", E."stop datetime"] during MeasurementPeriod

define InitialPopulation:

InDemographic

and exists (ActivePersistentAsthmaDiagnosis)

and exists (EncountersDuringMeasurementPeriod)

define Denominator:

InitialPopulation

define ActiveDiagnoses:

ChronicObstructivePulmonaryDiseaseDiagnoses

union EmphysemaDiagnoses

union CysticFibrosisDiagnoses

union AcuteRespiratoryFailureDiagnoses

define ActiveDiagnosesDuringMeasurementPeriod:

ActiveDiagnoses D where Interval[D."start datetime", D."stop datetime"] overlaps MeasurementPeriod

define DenominatorExclusions:

exists (ActiveDiagnosesDuringMeasurementPeriod)

define Numerator:

PreferredAsthmaTherapy P where Interval[P."start datetime", P."stop datetime"] during MeasurementPeriod

define Stratification1:

AgeInYearsAt(start of MeasurementPeriod) >= 5

and AgeInYearsAt(start of MeasurementPeriod) < 11

define Stratification2:

AgeInYearsAt(start of MeasurementPeriod) >= 11

and AgeInYearsAt(start of MeasurementPeriod) < 18

define Stratification3:

AgeInYearsAt(start of MeasurementPeriod) >= 18

and AgeInYearsAt(start of MeasurementPeriod) < 50

define Stratification4:

AgeInYearsAt(start of MeasurementPeriod) >= 50

and AgeInYearsAt(start of MeasurementPeriod) < 64

If I wanted to build a snippet on the initial population, I would call the program in the following manner:

$ python showCQLinHTML.py C:\Users\Christopher\Desktop\DCG\CQL\CMS126v4.cql InitialPopulation “Initial Population”

Which would generate the following snippet (in CQL for clarity):

define InitialPopulation:

InDemographic

and exists (ActivePersistentAsthmaDiagnosis)

and exists (EncountersDuringMeasurementPeriod)

define InDemographic:

AgeInYearsAt(start of MeasurementPeriod) >= 5

and AgeInYearsAt(start of MeasurementPeriod) < 64

define ActivePersistentAsthmaDiagnosis:

PersistentAsthmaDiagnoses A where Interval[A."start datetime", A."stop datetime"] overlaps MeasurementPeriod

define EncountersDuringMeasurementPeriod:

ValidEncounters E where Interval[E."start datetime", E."stop datetime"] during MeasurementPeriod

define PersistentAsthmaDiagnoses: ["Diagnosis, Active": "Persistent Asthma"]

define ValidEncounters:

OfficeVisitEncounter

union HomeHealthcareEncounter

union FaceToFaceEncounter

union EstablishedPreventiveCareForMinorEncounter

union EstablishedPreventiveCareForAdultEncounter

union InitialPreventiveCareForAdultEncounter

union InitialPreventiveCareForMinorEncounter

define OfficeVisitEncounter: ["Encounter, Performed": "Office Visit"]

define HomeHealthcareEncounter: ["Encounter, Performed": "Home Healthcare Services"]

define FaceToFaceEncounter: ["Encounter, Performed": "Face-to-Face Interaction"]

define EstablishedPreventiveCareForMinorEncounter: ["Encounter, Performed": "Preventive Care - Established Office Visit, 0 to 17"]

define EstablishedPreventiveCareForAdultEncounter: ["Encounter, Performed": "Preventive Care Services - Established Office Visit, 18 and Up"]

define InitialPreventiveCareForAdultEncounter: ["Encounter, Performed": "Preventive Care Services-Initial Office Visit, 18 and Up"]

define InitialPreventiveCareForMinorEncounter: ["Encounter, Performed": "Preventive Care- Initial Office Visit, 0 to 17"]

Note that the actual program would generate this representation in HTML.

## Sequence of Logic:

1. Validate command line arguments
   1. Check that file exists
   2. Check that the named expression name exists
2. Read file
   1. Fill a list with dirty code (line-by-line), which means that the code has not been stripped of whitespace.
   2. The dirty code will include everything after the “context Patient” statement in the .cql file being processed.
   3. Fill a dictionary with clean code from the dirty code list with the following format

{

<lineNumber> : [<line of code>, <length of preceding whitespace that was stripped>]

…

}

Where,

lineNumber is the unique key value for the dictionary.

ine of code is a string in the first index of the dictionary’s list value holding the CQL code from the lineNumber in the file.

Length of preceding whitespace in the second index of the dictionary’s list value is used for determining when to open or close a div.

1. Build HTML header div that encapsulates the snippet to be generated.
   1. Basic format
      1. <div class="treeview hover p-l-10">

<ul class="list-unstyled">

<input type="checkbox" id=""checked>

<label for=" " class="list-header">

<strong> </strong>

</label>

<ul class="code">

<li>

<div>

* + 1. The “id” attribute in the input tag, the “for” attribute in the label tag, and the text between the <strong> </strong> tags are filled using the <name of heading for the generated snippet> argument from the command line.

1. Begin processing the CQL code for HTML translation
   1. Get the block of code that the command line argument <name of named expression to translate> or the list of references (Will explain later 4.c.iv) specifies from the clean code dictionary.
   2. Build the div that every block is encapsulated within.
      1. Format

<div class="treeview hover">

<ul class="list-unstyled">

<input type="checkbox" id=" ">

<label for=" " class="list-header">

</label>

<ul>

<li>

* + 1. The “id” attribute in the input tag, the “for” attribute in the label tag are filled using the named expression name of the code block with “Define” appended to the end.
       1. For example
          1. define InitialPopulation: would translate to
          2. <div class="treeview hover">

<ul class="list-unstyled">

<input type="checkbox" id="InitialPopulationDefine">

<label for=" InitialPopulationDefine" class="list header">

… define keyword and className tags go here

* 1. Process the CQL code from the clean code dictionary line-by-line
     1. Split the code portion from the dictionary’s value list into tokens
     2. Use the patterns outlined above to determine which case is being represented
     3. Build the HTML tag-by-tag
     4. If any references are encountered, put the reference name (also the named expression name) into a list.
        1. This list will be used to process blocks that the current block is referencing.
     5. Build the closing tags for the code block from 4.b.i
        1. </li></ul></ul></div>

1. Check the reference list to determine if more blocks need to be processed.
   1. If yes, return to step 4
   2. If no,
      1. Build the closing tags for the snippet header from 3.a
         1. </li></ul></li></ul></div>
      2. print HTML code to a file called output.html, which is available in the directory which holds the program executable
      3. Terminate program

General Notes:

Please adhere to the following style guideline:

1. Always put a line break and tab after define statement. For example

define FaceToFaceEncounter:

["Encounter, Performed": "Face-to-Face Interaction"]

Never like this

define FaceToFaceEncounter: ["Encounter, Performed": "Face-to-Face Interaction"]