The Use of HL7 Specifications in Quality Measurement & Reporting

Stan Rankins
Integration Architect

Healthcare Intelligence

Objectives

- Discuss current standards involved with electronic quality measurement and reporting
- Break down QRDA I & III at a very high-level
- Explore Relationships Between QRDA and HQMF
- Discuss issues around the current standards
- Discuss work being done by S&I Clinical Quality Framework around eCQMs and reporting and new standards

Current Standards Framework

Foundational HL7 v3 Standards

- Among others:
  - Reference Information Model (RIM)
  - Data Types Specifications
    - ISO-Harmonized Data Types, Release 1
    - Abstract Data Type Specifications
      - Data Types Release 1 (R1) – Used by CDA-based documents like QRDA
      - Data Types Release 2 (R2) – Used by HQMF-based documents
  - Vocabulary

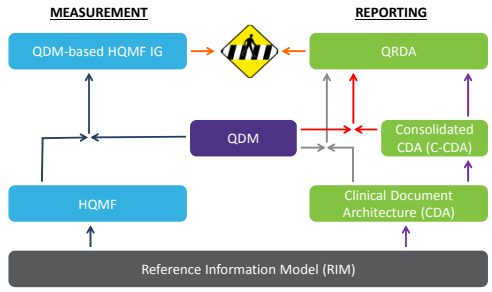
Quality Measure Standards

- Quality Data Model (QDM)
- HL7 Health Quality Measures Format (HQMF)
- HL7 QDM-based HQMF Implementation Guide (IG)

Quality Reporting Standards Framework

- HL7 Clinical Document Architecture (CDA)
- Consolidated CDA (C-CDA)
- Quality Reporting Document Architecture (QRDA)
  - Category I
  - Category III

eCQM and Reporting Standards Diagram



Healthcare Intelligence

7

CMS QRDA Implementation Guide (IG)

- CMS QRDA IG
  - In its 3rd annual release, with each release corresponding to a particular program year
  - During the first year, there were multiple CMS QRDA IGs, siloed by category and whether they were for the Eligible Professional or Eligible Hospital Community (PY 2014)
  - Beginning with PY 2015, the CMS QRDA IGs were consolidated
  - Work continues on the PY 2017 CMS QRDA IG – publication *TBD*
    - The Eligible Hospital portion went through public comment 3/30 – 4/18
    - The Eligible Professional portion was split out from the EH section due to a timing issue with regulations. It went through public comment from 4/11 – 5/2

Healthcare Intelligence

8

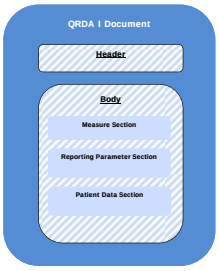
QRDA I

Healthcare Intelligence

9

QRDA I Outline

- Header:
  - Metadata about the file including some patient and provider info
- Body:
  - *Measure section:* Measures for which data is being provided
  - *Reporting Parameters section:* Reporting period information
  - *Patient Data section:* Encounters, Diagnoses, etc.



Healthcare Intelligence

10

QRDA I Example

- Human Readable: CAC_Multiple_Sample.html
- QRDA I XML: CAC_Multiple_Sample.xml

Healthcare Intelligence

11

QRDA I Relationship to HQMF

- According to QRDA I, a QRDA I file would be generated based on the patient being in the Initial Population for one of the measures being reported in the file
- Data Elements provided in the QRDA I file based on the data elements needed by each of the measures being reported in the file
- Data elements conveyed through templates mapped to the respective templated elements in the respective HQMF files

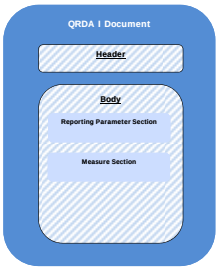
Healthcare Intelligence

12

QRDA III

QRDA III Outline

- Header:
  - Metadata about the file including some provider info
- Body:
  - *Reporting Parameters section:* Reporting period information
  - *Measure section:*
    - *Performance Rate(s)/CV if applicable*
    - *Measure Pop (MP) Counts*
    - *Per MP, Stratifier and SDE count(s)*



QRDA III Example

- Human Readable: Comprehensive Primary Care (CPC) Initiative Sample QRDA-III Report.htm
- QRDA III XML: Comprehensive Primary Care (CPC) Initiative Sample QRDA-III Report.xml

Measure Score

- Set by the respective measure - codified in the HQMF file
- Per HQMF, determines what measure populations may be provided in a measure and how those measure populations are used
- Used to determine what measure populations should be provided in a QRDA III file for a measure and if a performance rate(s) or measure observation(s) should be provided

Note: Measure Score is not provided in a QRDA I or QRDA III file for each measure.

Universally Unique Identifiers (UUIDs)

- In QRDA III, UUIDs are used to indicate measures and related items
  - Measure
    - CMS 136 v4: "40280381-4555-e1c1-0145-b957717c3738"
  - Measure Populations
    - CMS 125 v3 Denominator: "37E2BB57-67E4-4C21-A1C2-77857A05B908"
  - Stratifiers
    - CMS 126 v3 Stratum 4: "40280381-3d61-56a7-013e-66493e984444"

Note: These UUIDs match the UUIDs in the related HQMF files, thereby allowing comparison of apples to apples and oranges to oranges.

Problems With Current Framework

Problems with Current QDM Model

- Flat
  - Example:
    - Current QDM has 3 different attributes to express different facets about the level of care within an encounter:
      - Facility Location
      - Facility Location Arrival Datetime
      - Facility Location Departure Datetime
- Inconsistent
- Does not always map well to EHRs and how data is collected
- Different than the model used for Clinical Decision Support (CDS)

Current Measure Language Problems

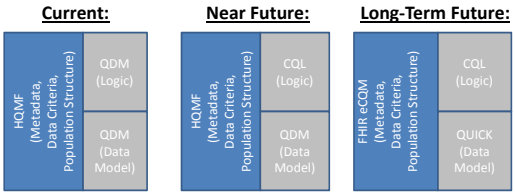
- The current QDM expression language is:
  - Use-case oriented and therefore not very extensible
  - Requires mapping to HQMF
    - Rigid
    - Weighty (e.g., CMS 117 v5 rel. on 04/06/16 has approx. 5,923 lines of XML)
    - Includes a model tightly coupled with its internal expression language
    - Contains different rules than QDM, so certain conventions were adopted in the QDM-based HQMF IG
    - Very Complex and Difficult to understand
  - Not the same language being used for Clinical Decision Support (CDS)

Standards & Interoperability Clinical Quality Framework Initiative

Scope of the Initiative

- To identify, define, and harmonize electronic standards that promote integration between CDS and eCQM in the areas of:
  - Metadata: Identify common metadata across the two domains and harmonize the identification and representation of that metadata.
  - Quality Information Data Model: Develop a quality information data model that supports the requirements of eCQM and CDS.
  - Logical Expression Language: Develop a common expression language that can be used to define both CDS and eCQM logical expressions
- To refactor existing CDS and eCQM standards to utilize the harmonized standards
- To the extent possible, pilot the standards

HQMF Framework



Future Logical Expression Language

- Clinical Quality Language
- CQL Defined

  A high-level, domain-specific language, focused on clinical quality, which can be turned into a syntax-independent, canonical representation of logic for clinical decision support

- Promises
  - Easier to write and understand than HQMF
  - Flexible
  - Plug and play models – e.g., QDM, QUICK
  - No requirement for expression language mapping to HQMF

Don't Worry – Be happy

- HQMF is **not** going away…at least, not now
- HQMF metadata, data criteria, population criteria and measure observation sections will still be used

HQMF Structure:

```
<qualityMeasureDocument>
  <!-- Metadata -->
  <!-- Data Criteria -->
  <!-- Population Criteria -->
  <!-- Measure Observations (if needed) -->
</qualityMeasureDocument>
```

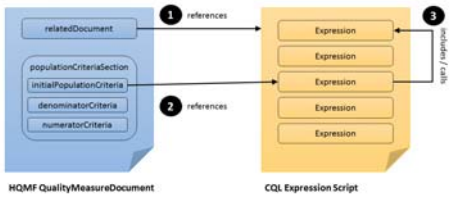

Reasons for Keeping HQMF

- Provides context for the measure through use of metadata
- Gives overall structure for a measure
- Offers cohesiveness by explanation of how populations relate to each other
- Supplies a way to transition to the new expression language
- Asserts the individual data points that need to be reported to CMS

HQMF – High-level Changes

- Current:
  - HQMF Contains Metadata
  - QDM Datatypes and Expression Language Mapped to HQMF Templates and Expression Language
  - 1 file: HQMF
- Future:
  - HQMF Contains Metadata
  - HQMF & CQL Have Symbiotic Relationship:
    - Data points provided in HQMF to help with scoop and filter approach
    - Populations point to a .cql file and related definitions
  - 3 files: HQMF, CQL & ELM

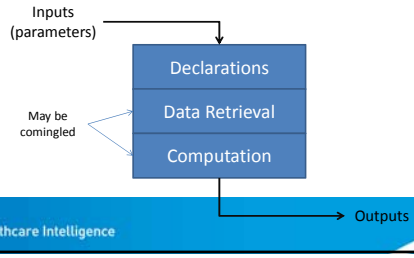
Picture Worth A Thousand Words



CQL Expressions

CQL File Structure

- Each library is a readable, plain text file – e.g., CMS 146
- May include other libraries by reference
- Logic in each file is potentially reusable by other libraries



CQL Declarations

- **library** declaration
  - Defines the name and optional version of the library
- **using** declaration
  - Defines the data model(s) in use in file
- **include** declaration
  - Defines other libraries (CQL files) referenced
- **context** declaration
  - Defines the context for subsequent statements(e.g. Patient or Population)
  - Anchors references in the file
- **parameter** declaration
  - Defines available "inputs"
- **valueset** declaration
  - Defines user-friendly labels for value sets within the library

Declaration Examples

```
// Library definition
library CMS146 version '2'

// Data model reference
using QDM version '4.2'

// Include Common library
include ChlamydiaScreening_Common  version '2'  as Common

// Define additional value sets for use within this artifact
valueset "Acute Pharyngitis": '2.16.840.1.113883.3.464.1003.102.12.1011'

// Establish patient context
context Patient
```


Retrieve

- Retrieves information from the data layer
  - Respects current context (Patient, Population)
- Specified in terms of Data Model
  - '[' namedTypeSpecifier ']'
    - ["Encounter, Performed"]
    - ["Procedure, Order"]
    - ["Device, Applied"]
- Optionally filter by "primary" Code
  - ["Encounter, Inpatient": "Inpatient"]
- Optionally filter by specific Code
  - ["Encounter, Performed": severity in "Severities"]

Data Types

- Simple Types
  - Boolean: *True* or *False*
  - String: *'female'*
  - Number: *16*
  - Date/Time: *@2015-05-01*
- Clinical Types
  - Quantities: *3 months*, *6 'g/mL' // UCUM is sometimes used but not always*
  - Value Sets: *"Female Administrative Sex"*
- Structured Types
  - Model Classes: *["Encounter, Performed"]*
  - Tuples: *Tuple { causeOfDeath: PE.cause , dateofDeath: PE."start datetime" }*

Data Types

- List Types
  - { 1, 2, 3, 4, 5 }
- Interval Types
  - Interval(today – 1 years, today]
  - Interval(@2013-01-01, date from(now))
  - Interval[D."onset datetime", P."incision datetime"]
  - Interval[*start of MeasurementPeriod*, Last(M."start datetime"))

Simple Expressions

- Logic
  - `A and B`
  - `A and not (B or C)`
- Comparison
  - `A >= B`
  - `A <> B`
- Arithmetic
  - `A + B`
  - `A + (B * C)`

Timing/Interval Operations

- Full set from QDM
  - *starts before start, starts same day as*
- Timing Phrases
  - *starts 3 days before start*
  - *starts 3 days or less before start*
  - *Ends within 3 days of start*
- Interval operators
  - *meets, overlaps, during*
- Boundary access
  - *start of MeasurementPeriod*
- Membership
  - *X in interval[4, 6]*

Interval Operators

Operator/Inverse	Diagram	Interpretation
X same as Y Y same as X	X Y	start of X = start of Y and end of X = end of Y
X before Y Y after X	X Y	end of X < start of Y
X meets before Y Y meets after X	X Y	successor of end of X = start of Y
X overlaps before Y Y overlaps after X	X Y	start of X <= start of Y and start of Y <= end of X
X begins Y Y begun by X	X Y	start of X = start of Y and end of X <= end of Y
X included in (during) Y Y includes X	X Y	start of X >= start of Y and end of X <= end of Y
X ends Y Y ended by X	X Y	start of X >= start of Y and end of X = end of Y

Date/Time Manipulation

- Date Construction
  - *@2014-01-01T12:00:00-06:00*
  - *Date(2014, 1, 1, 12, 0, 0, -6)*
  - *convert '2014-01-01T12:00:00-06:00' to DateTime*
- Date Arithmetic
  - *today + 3 months - 2 days*
  - *months between start of X and end of X*
  - *difference in days between X and Y*
  - *duration in months of X*
- Date/Time extraction
  - *date of D // returns the date without the time*
  - *time of D // returns the time without the date*
- Component extraction
  - *month of D // returns the number of whole units*

List Operations

- Selector – builds a list
  - *{ 1, 2, 3, 4, 5 }*
- Membership – determine if an element is in a list
  - *X in { 1, 2, 3, 4, 5 }*
  - *{ 1, 2, 3, 4, 5 } contains X*
- Comparison
  - *L = { 4, 5, 6 } // true if L has the same elements*
  - *L includes { 4, 5, 6 } // true if L includes each element*
  - *{ 4, 5, 6 } included in L // inverse of includes*
- Indexer/Position
  - *{ 4, 5, 6 }[1] // 1-based, evaluates to 4*
- Count
  - *Count({ 4, 5, 6 }) // evaluates to 3*
- First/Last
  - *First({ 4, 5, 6 }) // evaluates to 4*

Queries

- The "query" construct is used to perform various operations, including filtering, shaping, sorting, and relating results.
- Simplest query involves only a single source:

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

The alias "E" allows the source to be referenced anywhere within the query.

Filtering

- A "where" clause returns only those elements that satisfy the condition:

["Encounter, Performed": "Inpatient"] E
where E."length of stay" >= 120 days

Shaping

- The "return" clause allows the shape of the result to be described:

```
// Select elements to be returned
["Medication, Active": "Insulin"] M
    return Tuple { dose: M.dose, frequency: M.frequency }

// Compute value for returned elements
["Procedure, Performed": "Laparoscopy"] P
    return Tuple { result: P.result, duration: duration in days of Interval[P."start
datetime", P."stop datetime"]}

// Extract values
["Medication, Administered": "Warfarin"] M
    return duration of months in Interval[M."start datetime", M."stop datetime"]
```


Sorting

- "sort by" allows the results of a query to be ordered
- Sorting is evaluated after any "return"

```
// basic sort by
["Encounter, Performed": "Outpatient"] E
    sort by E."facility location arrival datetime"

//sort by with return
["Medication, Order": "Insulin"] M
    return Tuple { startDate: date from (M."start datetime"), dose: M.dose, refills: M.refills }
    sort by M.startDate desc
```


Filtering by Relationships

- Use the "with" keyword to introduce a filtering relationship:

```
// use of with
["Encounter, Performed": "Inpatient"] E
    with ["Diagnosis": "Acute Pharyngitis"] D
    such that Interval[D."start datetime", D."stop datetime") in Interval[E."start
    datetime", E."stop datetime")
```

NOTE: This operation is known as a semi-join in database languages.

Multi-source Queries

- Queries can specify multiple sources to be combined:

```
// multi-source query
define "Encounters with overlapping Warfarin and Parenteral Therapies" =
    from Encounters E, // another query called Encounters
        "Warfarin Therapy" W, // another query called Warfarin Therapy
        "Parenteral Therapy" P // another query called Parenteral Therapy
    where W."start datetime" in Interval[E."admission datetime", E."discharge
    datetime"]
    and P."start datetime" in Interval[E."admission datetime", E."discharge datetime"]
    and duration in days of (Interval[W."start datetime", W."stop datetime") intersect
    Interval[P."start datetime", P."stop datetime")) >= 5
```


Set Operations

- CQL supports standard set operations:
  - Union, intersection, and difference (except)

```
// union
["Encounter, Performed": "Outpatient"] EO
    union ["Encounter, Performed": "Inpatient"] EI

//intersect
["Encounter, Performed": "Inpatient"] EI
    where EI."length of stay" <=120 days
    intersect ["Encounter, Performed": "Inpatient"]
    ends during MeasurementPeriod

// difference (i.e., except)
["Encounter, Performed": "Provider Interaction"]
    except ["Encounter, Performed": "Face-to-Face"]
```


Conditional Expressions

- CQL supports an "if" expression, as well as a SQL-style "case" expression:

```
// conditional expression
If X > Y then X else Y

// selected case expression
case X
  when 1 then 12
  when 2 then 14
  else 15
end

// general case expression
case when X > Y then X when Y > X then Y else 0 end
```

Define Statements

- "define" statements can be used to break expressions into smaller chunks:

```
// encounters definition
define Encounters:
    ["Encounter, Performed": "Inpatient"] EI
    where Interval[EI."admission datetime", EI."discharge datetime"] during MeasurementPeriod

// encounters with dates definition
define "Encounters With Dates":
    Encounters E return Tuple { encounter: E, effectiveDate: Interval(E."admission datetime",
    E."discharge datetime") }

// dates definition
define Dates:
    distinct  "Encounters With Dates" X return X.effectiveDate
```


Defining Functions

- CQL allows functions to be defined:

```
// function CumulativeDuration
define function CumulativeDuration(intervals: List<Interval<DateTime>>){
    return Sum((collapse intervals) X return duration in days of X)
}

// use function
Define Encounters: ["Encounter, Performed": "Inpatient"

define CMD:
    CumulativeDuration(Encounters E return Interval(E."admission datetime", E."discharge
    datetime"))
```


Using Libraries

- Each library defines a name and optional version
- Elements referenced from a library must be qualified
- Library names must be unique within a repository
- Version number is optional for library definition
  - If none given, references cannot use a version
- Version is optional for include definition
  - If specified, that version must be used
  - If none given, the "most recent" version is used

- Questions?

Standards Resources

- Quality Data Model, v4.2
- Health Quality Measures Format (HQMF)
- QDM-based HQMF Implementation Guide (IG), R1.3
- Clinical Document Architecture (CDA)
- Consolidated CDA
- Quality Reporting Document Architecture (QRDA) Category I
- Quality Reporting Document Architecture (QRDA) Category III
- CMS QRDA IG (PY 2016)

Standards Resources

- Clinical Quality Language (CQL)
- CQL-based HQMF IG
- Quality Improvement Core (QICore) IG
- Quality Information and Clinical Knowledge (QUICK)
- S&I Clinical Quality Framework