Cooking with CQL Qs&As – Session 41

Thursday, January 16, 2020

Using Fast Healthcare Interoperability Resources (FHIR)

Q: Based on EXM130 – consider the Procedure.performed element:
http://hl7.org/fhir/STU3/procedure-definitions.html#Procedure.performed_x_
http://hl7.org/fhir/procedure-definitions.html#Procedure.performed_x_

In the Fast Healthcare Interoperability Resources (FHIR) Standard for Trial Use (STU) 3, it is defined as a choice of dateTime|Period and in R4, it is defined as a choice of dateTime|Period|string|Age|Range where the Quality Improvement (QI) Core constrains out the "string" type.

For STU3, the standard "Normalize Interval" function works:

```
define "Total Colectomy Performed":
      [Procedure: "Total Colectomy"] Colectomy
            where Colectomy.status = 'completed'
                  and Global."Normalize Interval"(Colectomy.performed) starts on or
      before end of "Measurement Period"
```

But for R4, we need to expand the "Normalize Interval" function to allow for the new types:

```
define function "Normalize Interval"(choice Choice<FHIR.dateTime, FHIR.Period, FHIR.Timing,
FHIR.instant, FHIR.string, FHIR.Age, FHIR.Range>):
   case
     when choice is FHIR.dateTime then
          Interval[FHIRHelpers.ToDateTime(choice as FHIR.dateTime),
   FHIRHelpers.ToDateTime(choice as FHIR.dateTime)]
     when choice is FHIR.Period then
                 FHIRHelpers.ToInterval(choice as FHIR.Period)
     when choice is FHIR.instant then
                        Interval[FHIRHelpers.ToDateTime(choice as FHIR.instant),
   FHIRHelpers.ToDateTime(choice as FHIR.instant)]
     when choice is FHIR.Age then
                   Interval[FHIRHelpers.ToDate(Patient.birthDate) +
   FHIRHelpers.ToQuantity(choice as FHIR.Age),
                          FHIRHelpers.ToDate(Patient.birthDate) +
   FHIRHelpers.ToQuantity(choice as FHIR.Age) + 1 year)
     when choice is FHIR.Range then
```

```
                  Interval[FHIRHelpers.ToDate(Patient.birthDate) +
    FHIRHelpers.ToQuantity((choice as FHIR.Range).low),
                         FHIRHelpers.ToDate(Patient.birthDate) +
    FHIRHelpers.ToQuantity((choice as FHIR.Range).high) + 1 year)
      when choice is FHIR.Timing then
                  Message(null as Interval<DateTime>, true, '1', 'Error', 'Cannot compute
    a single interval from a Timing type')
        when choice is FHIR.string then
          Message(null as Interval<DateTime>, true, '1', 'Error', 'Cannot compute an
    interval from a String value')
                else null as Interval<DateTime>
          end
```

It's a great idea to have a "Normalize Interval" function in the global library, but will you have one function, "Normalize Interval" function, that contains the flexible arguments in the parameter or will you have multiple ones depending on the FHIR resource? How many choice data type elements will the resource have?

A: We could define "Normalize Interval" versions that had only the choices for what we wanted but when you start doing that, the differences between the sets of types defined in the different elements becomes unwieldy. By defining one version of "Normalize Interval" that has all the possible choice types we encounter in timing elements, we can define one function that can handle all of those and then as long as the types that the elements can be are a subset of these, then the Clinical Quality Language (CQL) can work with it.

Q: Based on the EXM130 expression below, can you explain the "Message" function and each component of it?

```
 define function "Normalize Interval"(choice Choice<FHIR.dateTime, FHIR.Period, FHIR.Timing,
 FHIR.instant, FHIR.string, FHIR.Age, FHIR.Range>):
    case
      when choice is FHIR.dateTime then
           Interval[FHIRHelpers.ToDateTime(choice as FHIR.dateTime),
    FHIRHelpers.ToDateTime(choice as FHIR.dateTime)]
      when choice is FHIR.Period then
                FHIRHelpers.ToInterval(choice as FHIR.Period)
      when choice is FHIR.instant then
                       Interval[FHIRHelpers.ToDateTime(choice as FHIR.instant),
    FHIRHelpers.ToDateTime(choice as FHIR.instant)]
      when choice is FHIR.Age then
                  Interval[FHIRHelpers.ToDate(Patient.birthDate) +
    FHIRHelpers.ToQuantity(choice as FHIR.Age),
                              FHIRHelpers.ToDate(Patient.birthDate) +
    FHIRHelpers.ToQuantity(choice as FHIR.Age) + 1 year)
      when choice is FHIR.Range then
                       Interval[FHIRHelpers.ToDate(Patient.birthDate) +
    FHIRHelpers.ToQuantity((choice as FHIR.Range).low),
                              FHIRHelpers.ToDate(Patient.birthDate) +
    FHIRHelpers.ToQuantity((choice as FHIR.Range).high) + 1 year)
      when choice is FHIR.Timing then
                       Message(null as Interval<DateTime>, true, '1', 'Error', 'Cannot compute
    a single interval from a Timing type')
        when choice is FHIR.string then
          Message(null as Interval<DateTime>, true, '1', 'Error', 'Cannot compute an
    interval from a String value')
                     else null as Interval<DateTime>
            end
```

A: Clinical Quality Language (CQL) is a functional language, meaning everything in CQL is going to return a value. Even in the case where we're throwing an 'Error' or returning a 'Warning,' it will still return a value. The first argument to the Message function is the value it will return as an Interval. You can also add conditions to log the Message. Below are the components of this line broken down:

Message(null as Interval<DateTime>, true, '1', 'Error', 'Cannot compute a single interval from a Timing type')

- null as Interval<DateTime> = the first argument is the value you want to return
- true = the condition, you can conditionally return or log the message.
- 1 = the error code.
- Error = the message you want to send back - this can be 'Error,' 'Warning,' or 'Information.' If it's an 'Error,' it also stops processing. If it's a 'Warning,' it's a message that comes back as part of the evaluation. Since it doesn't stop processing, it needs to send back a result.
- Cannot compute a single interval from a Timing type = the message you actually want to send out.

Q: Based on the EXM130 expression below, why does the message only happen to the timing, instance, and string? Does it not apply for all resources?

```
  define function "Normalize Interval"(choice Choice<FHIR.dateTime, FHIR.Period, FHIR.Timing,
  FHIR.instant, FHIR.string, FHIR.Age, FHIR.Range>):
    case
      when choice is FHIR.dateTime then
          Interval[FHIRHelpers.ToDateTime(choice as FHIR.dateTime),
    FHIRHelpers.ToDateTime(choice as FHIR.dateTime)]
      when choice is FHIR.Period then
                    FHIRHelpers.ToInterval(choice as FHIR.Period)
      when choice is FHIR.instant then
                               Interval[FHIRHelpers.ToDateTime(choice as FHIR.instant),
    FHIRHelpers.ToDateTime(choice as FHIR.instant)]
      when choice is FHIR.Age then
                      Interval[FHIRHelpers.ToDate(Patient.birthDate) +
    FHIRHelpers.ToQuantity(choice as FHIR.Age),
                               FHIRHelpers.ToDate(Patient.birthDate) +
    FHIRHelpers.ToQuantity(choice as FHIR.Age) + 1 year)
      when choice is FHIR.Range then
                      Interval[FHIRHelpers.ToDate(Patient.birthDate) +
    FHIRHelpers.ToQuantity((choice as FHIR.Range).low),
                               FHIRHelpers.ToDate(Patient.birthDate) +
    FHIRHelpers.ToQuantity((choice as FHIR.Range).high) + 1 year)
      when choice is FHIR.Timing then
                      Message(null as Interval<DateTime>, true, '1', 'Error', 'Cannot compute
    a single interval from a Timing type')
        when choice is FHIR.string then
          Message(null as Interval<DateTime>, true, '1', 'Error', 'Cannot compute an
    interval from a String value')
                    else null as Interval<DateTime>
          end
```

A: In the case of a timing element as highlighted above, a FHIR.string means you have a string that the user entered as the value for that time. For example, in this condition the resource string could have a value of "last spring." With Fast Healthcare Interoperability Resources (FHIR) in the condition resource, it is possible to communicate that the user has actually entered the timing as just a natural language string and that's what the string choice type means. Regarding FHIR.Instant as highlighted above, it is okay since it is just a DateTime that has to be to the millisecond. Regarding timing, in R4 it is an observation which was expanded to support not only DateTime and Period, but also Timing and Instant.

Timing is a very complex object that lets you build up very complex schedules, therefore, it is very flexible. However, if there is a use case where there is a system that wants to use the timing type of the effective element of an observation it means that the data received cannot be processed, which will result in an error message.

Q: In the EXM130 expression below, would it be better to move the FHIR.Timing and the FHIR.string to the bottom of the list if one were to prioritize the list?

```
define function "Normalize Interval"(choice Choice<FHIR.dateTime, FHIR.Period, FHIR.Timing,
FHIR.instant, FHIR.string, FHIR.Age, FHIR.Range>):
```

```
case
  when choice is FHIR.dateTime then
       Interval[FHIRHelpers.ToDateTime(choice as FHIR.dateTime),
FHIRHelpers.ToDateTime(choice as FHIR.dateTime)]
  when choice is FHIR.Period then
              FHIRHelpers.ToInterval(choice as FHIR.Period)
  when choice is FHIR.instant then
                     Interval[FHIRHelpers.ToDateTime(choice as FHIR.instant),
FHIRHelpers.ToDateTime(choice as FHIR.instant)]
  when choice is FHIR.Age then
                Interval[FHIRHelpers.ToDate(Patient.birthDate) +
FHIRHelpers.ToQuantity(choice as FHIR.Age),
                       FHIRHelpers.ToDate(Patient.birthDate) +
FHIRHelpers.ToQuantity(choice as FHIR.Age) + 1 year)
  when choice is FHIR.Range then
                Interval[FHIRHelpers.ToDate(Patient.birthDate) +
FHIRHelpers.ToQuantity((choice as FHIR.Range).low),
                       FHIRHelpers.ToDate(Patient.birthDate) +
FHIRHelpers.ToQuantity((choice as FHIR.Range).high) + 1 year)
  when choice is FHIR.Timing then
                Message(null as Interval<DateTime>, true, '1', 'Error', 'Cannot compute
a single interval from a Timing type')
    when choice is FHIR.string then
      Message(null as Interval<DateTime>, true, '1', 'Error', 'Cannot compute an
interval from a String value')
              else null as Interval<DateTime>
       end
```

A: The case expression in Clinical Quality Language (CQL) doesn't carry any implication of ordered processing and in any given instance choices are exclusive. Therefore, you will only have one set of the types chosen at a given time. For example, you would not have a FHIR.dateTime and a FHIR.Timing in the same element. However, implementations may walk through each of the instances from top to bottom so it would be reasonable to move the FHIR.Timing and the FHIR.string to the bottom to allow for the more common instances to be hit first, but it doesn't change the actual semantics of the function.

Q: The measure EXM135 is an example of ConditionOnsetAbatementPrevalencePeriod where we determine the Onset/Abatement and Prevalence Period for a condition. We define the Prevalence Period:

```
define function "Prevalence Period"(condition Condition):
```

```
Interval[start of "Normalize Onset"(condition.onset), end of "Normalize Abatement"(condition))
```

This makes sense for Pregnancy, but could you also use the procedure ‘Delivery’ to say that it ended?

A: Yes, this is correct. For all other conditions you can follow the example but for pregnancy you can also use ‘Delivery’.

Q: For the measure EXM135, is there a reason you exclude the timepoint of the abatement time by using parentheses “)” instead of brackets “]”?

A: Yes, there are two reasons:

1. This is when they said the condition was over so it really shouldn’t be included during the time that it is happening. It is an exclusive boundary.
2. **More importantly, if this element is not present and we use the inclusive boundary, then it would mean that this condition goes until the end of time**. This is the opposite of what we are trying to say. Using the inclusive boundary means that we don’t know when that interval ends.

So, excluding the abatement time assumes that the condition abated some time before the recorded date. If this is false, then it means the condition goes on until the end of time.