Updating HeD Artifacts from DSTU 2 to 3

# Overview

This document provides a detailed description of the changes made to the Clinical Decision Support Knowledge Artifact Specification as part of the DSTU Update 3. Broadly, the changes focused on two aspects:

1. Alignment with Structured Data Capture’s form definition structures
2. Replacement of HeD expression logic representation with Expression Logical Model (ELM) representation from the Clinical Quality Language.

The changes made in connection with each of these areas will be detailed, followed by a step-by-step process for updating an artifact from DSTU 2 to DSTU 3.

# Changes Made to Align with SDC

The changes described here state what was changed. For a description of the alignment process and requirements traceability, refer to the included spreadsheet, SDC\_HeD\_Mapping.xlsx.

## base.xsd

Added complexType FormattedText

Added complexType FormattedNumber

Changed type of KnowledgeResource.title to FormattedText

Changed type of KnowledgeResource.description to FormattedText

## action.xsd

Changed type of ActionGroup.title to FormattedText

Changed type of ActionGroup.description to FormattedText

Added new element called ActionBase.number

Added new attribute ActionBase.order

## behaviortype.xsd

Added CardinalityBehaviorType

## behavior.xsd

Added CardinalityBehavior

## catalogitem.xsd

Renamed DocumentationItem.displayText to prompt

Changed type of prompt to FormattedText

Changed type of description to FormattedText

Renamed ListConstraint to EnumerationConstraint to clarify meaning

Added a new subtype of RangeConstraint called LookupConstraint

Added a new subtype of RangeConstraint called MaskConstraint

Added new element called textAfterResponse in DocumentationItem

Added new element called scopedIdentifier to ItemDefinition

Moved list item from an inner type to an independent complex type called EnumerationItem

Added new element called identifier to EnumerationItem

Added new element called number to EnumerationItem

Added new element called additionalInstructions to EnumerationItem

Added new element called valueMeaning to EnumerationItem

Added new attribute called fillIn to EnumerationItem

Added new element called additionalInstructions to DocumentationItem

# Changes Made to Replace HeD Expression Logic with ELM

## action.xsd

CreateAction.actionSentence - changed type from Expression to elm:Expression

UpdateAction.actionSentence - changed type from Expression to elm:Expression

RemoveAction.actionSentence - changed type from Expression to elm:Expression

FireEventAction - Removed pending further definition

CollectInformationAction.initialValue - changed type from Expression to elm:Expression

## actor.xsd

Actor.actor - changed type from Expression to Party

## catalogitem.xsd

ExpressionConstraint.constraint - changed type from Expression to elm:Expression

LookupConstraint.items - changed type from Expression to elm:Expression

ValueSetConstraint.valueSet - changed type from ValueSet to elm:ValueSetRef

EnumerationItem.value - changed type from Expression to elm:Expression

## clinicalexpression.xsd

Removed this schema (replaced by clinicalexpression.xsd from ELM)

## condition.xsd

Condition.logic - changed type from Expression to elm:Expression

## expression.xsd

Removed this schema (replaced by expression.xsd from ELM)

## knowledgedocument.xsd

KnowledgeDocument.externalData.codesystem - added elm:CodeSystemDef

KnowledgeDocument.externalData.valueset - added elm:ValueSetDef

KnowledgeDocument.externalData.parameter - changed type from ParameterDef to elm:ParameterDef

KnowledgeDocument.externalData.def - changed type from ExpressionDef to elm:ExpressionDef

KnowledgeDocument.expressions.def - changed type ExpressionDef to elm:ExpressionDef

KnowledgeDocument.externalData.trigger - added Trigger

KnowledgeDocument.triggers - removed (moved to the externalData section to consolidate with other external data)

## literalexpression.xsd

Removed, not represented in ELM

## trigger.xsd

Trigger.expression - removed (replaced by a new def element)

Trigger.def - added elm:ExpressionDef

# Upgrading an Artifact from DSTU 2 to DSTU 3

Almost all of the changes required to bring an artifact from DSTU2 to DSTU3 are due to the change to the use of ELM, rather than the original HeD Schema for the representation of logic. The following sections detail the required changes.

## Namespaces

Since the ELM schema is defined in a different namespace, ELM elements referenced within HeD artifacts must now be namespace qualified. To do this, first introduce the required namespaces:

xmlns:elm="urn:hl7-org:elm:r1"

xmlns:t="urn:hl7-org:elm-types:r1"

xmlns:a="urn:hl7-org:cql-annotations:r1"

If appropriate for the context of the artifact, give schemaLocation information for each namespace:

urn:hl7-org:elm:r1 ../schema/elm/clinicalexpression.xsd

urn:hl7-org:elm-types:r1 ../schema/elm/types.xsd

urn:hl7-org:cql-annotations:r1 ../schema/elm/cqlannotations.xsd

And finally, add the appropriate local namespace prefix to all expression elements within the artifact:

<def name="PatientAge">

<elm:expression xsi:type="elm:Property" path="age">

<elm:source xsi:type="elm:ExpressionRef" name="Patient" />

</elm:expression>

</def>

Because the HeD expression schema served as a conceptual basis for ELM, most expression types have direct representation in ELM and adding the namespace qualifier is all that is required. However, some types have minor name changes, some types have conceptual changes that must be accounted for, and some types have no direct counterpart and must be expressed using a combination of ELM expressions. These changes will be discussed in detail in the sections that follow.

## ClinicalRequest -> Retrieve

The ClinicalRequest type in HeD maps to the Retrieve type in ELM, with the following exceptions:

* The *cardinality* attribute is not specified in ELM. An ELM retrieve will always return a list. If the HeD cardinality was *Multiple*, it can be removed. If the HeD cardinality was *Single*, the Retrieve should be wrapped in a *SingletonFrom* expression to extract the singleton result.
* The *isInitial* attribute is not used in ELM. The information it conveyed can be determined by static analysis and so can be removed. This functionality may be reintroduced at a later time to enable dynamic control over payload sizes, but this aspect is considered an implementation detail by the normative specification.
* The *useValueSets* attribute is not used in ELM. As with *isInitial*, this information can be determined by static analysis, and is considered an implementation detail.

The following example illustrates the upgrade of a simple singleton clinical request:

<def name="Patient">

<expression xsi:type="ClinicalRequest" cardinality="Single"

dataType="vmr:EvaluatedPerson" isInitial="true" />

</def>

In DSTU3:

<def name="Patient">

<elm:expression xsi:type="elm:SingletonFrom">

<elm:operand xsi:type="elm:Retrieve"

dataType="vmr:EvaluatedPerson"/>

</elm:expression>

</def>

Note the use of a *SingletonFrom* to replace the *cardinality=”Single”* attribute in the DSTU2 version.

## XXXLiteral expressions

The various *XXXLiteral* types are not present in ELM, so these must be replaced by their ELM equivalents, as specified in the following table:

|  |  |
| --- | --- |
| HeD Schema | ELM Equivalent |
| BooleanLiteral | Literal w/ valueType=”t:Boolean” |
| IntegerLiteral | Literal w/ valueType=”t:Integer” |
| IntegerIntervalLiteral | Interval w/ appropriate attributes and literals |
| PhysicalQuantityLiteral | Quantity w/ appropriate attributes |
| PhysicalQuantityIntervalLiteral | Interval w/ appropriate attributes and literals |
| QuantityIntervalLiteral | Interval w/ appropriate attributes and literals |
| RealLiteral | Literal w/ valueType=”t:Decimal” |
| RealIntervalLiteral | Interval w/ appropriate attributes and literals |
| TimestampLiteral | DateTime w/ appropriate elements |
| TimestampIntervalLiteral | Interval w/ appropriate attributes and literals |
| StringLiteral | Literal w/ valueType=”t:String” |
| UrlLiteral | Literal w/ valueType=”t:String” |

For *CodeLiteral* expressions, the ELM *Code* type is used:

* *codeSystem* and *codeSystemName* attributes are used to create a *CodeSystemDef* element in the artifact:
  + <codesystem name="CPT-4" id="2.16.840.1.113883.6.12"/>
* The *code* attribute maps directly, the *displayName* attribute maps to *display*
* The *system* element of the ELM *Code* is then a *CodeSystemRef* with the *name* attribute set to the name defined for the *CodeSystemDef*.

For all other literal expressions, an *Instance* selector with appropriate *InstanceElement* elements for each attribute/element of the value must be constructed.

## Expression descriptions

*Expression.description* elements have been replaced in ELM with the more general notion of *Annotations*. For a simple description, use an *Annotation* with a narrative element:

<description>Asthma diagnosis code entered during this or any prior encounter AND Persistent asthma code entered during this or any prior encounter</description>

Becomes:

<elm:annotation xsi:type="a:Annotation"><a:s>Asthma diagnosis code entered during this or any prior encounter AND Persistent asthma code entered during this or any prior encounter</a:s></elm:annotation>

## Nary expressions (And/Or/Union/Intersect)

Nary expressions such as *And*, *Or*, *Union*, and *Intersect* are modeled as binary expressions in ELM and must be expanded to multiple expressions.

## ValueSet expressions

*ValueSet* expressions are not supported in ELM (by design) and must be replaced by an equivalent *ValueSetDef/ValueSetRef* combination. The value set *id* and *version* attributes are specified in the *ValueSetDef* as part of the external data for the artifact. Choose a description name and use that local name in the *ValueSetRef* in the actual expression:

<codes xsi:type="ValueSet" id="12345" authority="zynx.com/valuesets" version="1" />

Becomes:

<valueset name="Inhaled Corticosteroids Medications" id="12345" version="1"/>

In the external data, and:

<elm:codes xsi:type="elm:ValueSetRef" name="Inhaled Corticosteroids Medications"/>

In the expression. The *authority* attribute is not used in ELM and is considered an implementation detail.

## DateAdd/DateDiff/DatePart

The *DateAdd* in HeD is represented uses date/time arithmetic in ELM using an *Add* of the DateTime argument and a *Quantity* with the appropriate granularity:

<begin xsi:type="DateAdd">

<date xsi:type="Today" />

<granularity xsi:type="Literal" valueType="DateGranularity"

value="Month" />

<numberOfPeriods xsi:type="IntegerLiteral" value="-12" />

</begin>

becomes:

<elm:low xsi:type="elm:Add">

<elm:operand xsi:type="elm:Today"/>

<elm:operand xsi:type="elm:Quantity" unit="a" value="-12"/>

</elm:low>

Note that if the quantity operand is not a literal (e.g. a parameter reference) an *Instance* of type t:Quantity must be constructed.

*DateDiff* in HeD is represented as *DifferenceBetween*, or *DurationBetween*, depending on whether boundary counting or duration counting is intended. Note that ELM did not distinguish between the two, so an appropriate decision must be made on a case-by-case basis. Typically, age calculations would use a *DurationBetween*, or better yet, the actual *Age* or *AgeAt* operations in ELM.

## IsEmpty/IsNotEmpty

*IsNotEmpty* in HeD is represented using an *Exists* operator in ELM. *IsNotEmpty* is represented as the logical negation (*Not*) of an *Exists*.

## ObjectExpression

An *ObjectExpression* in HeD is represented as an *Instance* selector in ELM.

## ObjectRedefine

An *ObjectRedefine* in HeD is represented as a singleton *Query*/*Return* combination in ELM:

<actionSentence xsi:type="ObjectRedefine">

<source xsi:type="ExpressionRef" name="BaseInsulinOrder" />

<property name="dose[1].doseQuantity" xsi:type="PropertyExpression">

<value xsi:type="PhysicalQuantityIntervalLiteral"

lowClosed="true" highClosed="true">

<low value="12" unit="Unit" />

<high value="12" unit="Unit" />

</value>

</property>

<property name="dose[1].frequency">...</property>

<property name="prnReason">...</property>

</actionSentence>

becomes:

<actionSentence xsi:type="elm:Query">

<elm:source alias="O">

<elm:expression xsi:type="elm:ExpressionRef" name="BaseInsulinOrder"/>

</elm:source>

<elm:return>

<elm:expression xsi:type="elm:Instance"

classType="vmr:SubstanceAdministrationProposal">

<elm:element name="substance">

<elm:value xsi:type="elm:Property"

path="substance" scope="O"/>

</elm:element>

<elm:element name="dose">

<elm:value xsi:type="elm:Property"

path="dose" scope="O"/>

</elm:element>

<elm:element name="dose[1].doseQuantity">

<elm:value xsi:type="elm:Interval" lowClosed="true" highClosed="true">

<elm:low xsi:type="elm:Quantity" value="2" unit="Unit" />

<elm:high xsi:type="elm:Quantity" value="2" unit="Unit" />

</elm:value>

</elm:element>

<elm:element name="dose[1].frequency">...</elm:element>

<elm:element name="prnReason">..</elm:element>

</elm:expression>

</elm:return>

</actionSentence>

Note that unlike the *ObjectRedefine*, the *Query* return element must specify all attributes, rather than only those that are being evaluated.

An alternative representation not involving the *Query* element in ELM would be to promote the source to a *List* using a list selector, perform a *ForEach* and then extract the result with a *SingletonFrom*.