|  |
| --- |
| Intermountain Healthcare |
| CEM Best Practice Style Guide |
| A guide for authoring Clinical Element Models in CDL |

**Versioning**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Description | Author | Date |
| 1 | Initial | Teresa Conway | October, 2011 |
| 2 | Major revision | Tom Oniki | Aug 2012 |
| 2.1 | Incorporated edits | Tom Oniki | Aug 2012 |

Contents

[**Versioning** 1](#_Toc333001751)

[Contents 2](#_Toc333001752)

[Introduction 7](#_Toc333001753)

[Purpose of this Document 7](#_Toc333001754)

[Clinical Element Models 7](#_Toc333001755)

[Abstract Instance Model 7](#_Toc333001756)

[Abstract Constraint Model 10](#_Toc333001757)

[The Use of Terminology in CEMs 12](#_Toc333001758)

[Code Systems and CEMs 12](#_Toc333001759)

[References to Codes in CEMs 13](#_Toc333001760)

[Value Sets in CEMs 13](#_Toc333001761)

[The Constraint Definition Language (CDL) 14](#_Toc333001762)

[Authoring CEMs in CDL 14](#_Toc333001763)

[Authoring Tools 14](#_Toc333001764)

[Modes of Authoring 15](#_Toc333001765)

[Authoring “From Scratch” 15](#_Toc333001766)

[Authoring using a Template 15](#_Toc333001767)

[Authoring using another CEM 15](#_Toc333001768)

[CDL Data Types 15](#_Toc333001769)

[CD/CO 15](#_Toc333001770)

[II 17](#_Toc333001771)

[INT 17](#_Toc333001772)

[PQ 17](#_Toc333001773)

[REAL 18](#_Toc333001774)

[ST 18](#_Toc333001775)

[TS 18](#_Toc333001776)

[CDL Elements in a CEM 19](#_Toc333001777)

[General CDL rules 19](#_Toc333001778)

[Copyright statement 20](#_Toc333001779)

[Import statements 20](#_Toc333001780)

[Header 21](#_Toc333001781)

[Model Name 24](#_Toc333001782)

[key 24](#_Toc333001783)

[data 24](#_Toc333001784)

[Qualifier/Item/Modifier/Attribution references 25](#_Toc333001785)

[Constraint Statements 27](#_Toc333001786)

[Conventions 32](#_Toc333001787)

[Naming of Models 32](#_Toc333001788)

[References to Terminology 33](#_Toc333001789)

[Ordering of Elements 34](#_Toc333001790)

[Modifications to Models 34](#_Toc333001791)

[Creating Terminology for Use in CEMs 35](#_Toc333001792)

[Key Codes and Type Codes 35](#_Toc333001793)

[Value Sets 35](#_Toc333001794)

[Further Constraint of Value Sets 35](#_Toc333001795)

[Inheritance in CEMs and CDL 37](#_Toc333001796)

[Inheritance via Reference classes 37](#_Toc333001797)

[Inheritance via Extends and Restricts Statements 40](#_Toc333001798)

[CEM structures 43](#_Toc333001799)

[Statements 43](#_Toc333001800)

[Statements Differentiated by Value Choice 48](#_Toc333001801)

[Statements Differentiated by Clinical Usage and Semantics 48](#_Toc333001802)

[Procedures 57](#_Toc333001803)

[Components 59](#_Toc333001804)

[Components Used as Qualifiers 60](#_Toc333001805)

[Components Used as Items 61](#_Toc333001806)

[Guideline: Creating a New Component CEM Versus Reusing an Existing CEM 61](#_Toc333001807)

[Guideline: Modeling as a component or a statement 62](#_Toc333001808)

[Modifiers 63](#_Toc333001809)

[Attributions 64](#_Toc333001810)

[Attribution Structure 64](#_Toc333001811)

[Associations 68](#_Toc333001812)

[Panels 72](#_Toc333001813)

[CDL support for propagation 76](#_Toc333001814)

[Semantic Links 77](#_Toc333001815)

[Qualifiers vs. Panels vs. Semantic Links 78](#_Toc333001816)

[Annotations (Comments) 80](#_Toc333001817)

[Examples of Commonly Used CEM structures 81](#_Toc333001818)

[Components 81](#_Toc333001819)

[Aggregate Qualifier 81](#_Toc333001820)

[Body Location/Body Location Pre-coordinated Qualifiers 82](#_Toc333001821)

[Body Position Qualifier 84](#_Toc333001822)

[Route, RouteMethodDevice and MethodDevice Qualifiers 84](#_Toc333001823)

[AssociatedCondition Qualifier 84](#_Toc333001824)

[Periodicity Qualifier 85](#_Toc333001825)

[Course Qualifier 85](#_Toc333001826)

[Severity Qualifier 85](#_Toc333001827)

[Alleviating Factor Qualifier 85](#_Toc333001828)

[Exacerbating Factor Qualifier 85](#_Toc333001829)

[Date of Onset Qualifier 86](#_Toc333001830)

[Date of Resolution Qualifier 86](#_Toc333001831)

[Associated Signs and Symptoms Qualifier 86](#_Toc333001832)

[Abnormal Interpretation Qualifier 86](#_Toc333001833)

[Delta Flag Qualifier 87](#_Toc333001834)

[ReferenceRangeNar Qualifier 87](#_Toc333001835)

[ReferenceRange<data type> Qualifier 87](#_Toc333001836)

[Attributions 88](#_Toc333001837)

[Observed 88](#_Toc333001838)

[Performed 88](#_Toc333001839)

[ReportReceived 89](#_Toc333001840)

[SpecimenCollected 90](#_Toc333001841)

[SpecimenReceivedByLab 90](#_Toc333001842)

[Resulted 90](#_Toc333001843)

[Corrected 90](#_Toc333001844)

[Verified 90](#_Toc333001845)

[Ordered 90](#_Toc333001846)

[Discontinued 90](#_Toc333001847)

[CounterSigned 90](#_Toc333001848)

[PutOnHold 90](#_Toc333001849)

[TakeOffHold 91](#_Toc333001850)

[Other Attributions 91](#_Toc333001851)

[Modifiers 93](#_Toc333001852)

[Subject Modifier 93](#_Toc333001853)

[Negation Indicator Modifier 93](#_Toc333001854)

[Uncertainty Modifier 94](#_Toc333001855)

[Authoring issues 94](#_Toc333001856)

[Creating a New Component CEM Versus Reusing an Existing CEM 94](#_Toc333001857)

[Modeling a Statement as an Evaluation Versus an Assertion 94](#_Toc333001858)

[Creating a new statement CEM versus reusing a CEM 94](#_Toc333001859)

[Use of “normal” in a value set 95](#_Toc333001860)

[Representing that data is absent 95](#_Toc333001861)

[Use of nullFlavor vs adding “unknown” or “other” to a value set 95](#_Toc333001862)

[Use of negation vs “not present” in a value set 95](#_Toc333001863)

[Modeling as a component or a statement 95](#_Toc333001864)

[New Subtypes vs. Generic Subtypes 95](#_Toc333001865)

[Large Value Sets in Component Models 96](#_Toc333001866)

[Advantages and disadvantages of each approach 96](#_Toc333001867)

[Isosemantic models 96](#_Toc333001868)

[Practicality of “one model” for each data element 97](#_Toc333001869)

[Derivations/summaries and calculations 97](#_Toc333001870)

[“Context” or “Usage” models 97](#_Toc333001871)

[“UI” models 97](#_Toc333001872)

[Precoordination versus postcoordination 98](#_Toc333001873)

[Denormalization 98](#_Toc333001874)

[What to put in the model versus what to leave in terminology or knowledge 99](#_Toc333001875)

[Semantic links vs “pointer” attributes 99](#_Toc333001876)

[Qualifiers vs Panels vs semantic links 99](#_Toc333001877)

[Generic vs specific models 99](#_Toc333001878)

[When to create separate models vs use the same model 99](#_Toc333001879)

[Use case differences in models 99](#_Toc333001880)

[Modeling the implied procedure 99](#_Toc333001881)

[Separating the key code from the value set head concept – question vs answer 99](#_Toc333001882)

[Appendix A – Model and Terminology Changes and Backward Compatibility 101](#_Toc333001883)

[Appendix B – Rationale for Assertion pattern 106](#_Toc333001884)

[Appendix C – Attribution Examples 107](#_Toc333001885)

[Observed Attribution Examples 107](#_Toc333001886)

[Performed Attribution Examples 116](#_Toc333001887)

[ReportedReceived Attribution Example 119](#_Toc333001888)

[SpecimenCollected/SpecimenReceivedByLab/Resulted Attribution Example 120](#_Toc333001889)

[Corrected Attribution Examples 124](#_Toc333001890)

[Verified Attribution Examples 127](#_Toc333001891)

[Ordered Attribution Example 129](#_Toc333001892)

[Discontinued Attribution Example 131](#_Toc333001893)

[Documented Attribution Example 133](#_Toc333001894)

[Countersigned Attribution Example 135](#_Toc333001895)

[PutOnHold Attribution Example 138](#_Toc333001896)

[TakeOffHold Attribution Example 140](#_Toc333001897)

# Introduction

## Purpose of this Document

The purpose of this document is to give modelers the information and background needed to author Clinical Element Models (CEMs) in the Constraint Definition Language (CDL).

The document assumes the reader has some level of familiarity with CDL. It does not provide an extensive description of the need for Detailed Clinical Models (DCMs) and CEMs. It is assumed that readers are familiar with those principles.

It does not provide extensive technical detail about CDL. CDL is described in detail in the “Qualibria Constraint Definition Language (CDL) Language Guide”.

This document also does not discuss implementation of CEMs in a runtime system.

## Clinical Element Models

Intermountain Healthcare devised the Clinical Element Model to specify, at a granular and computable fashion, the structure and semantics of each data element that will be stored in an EMR. The Clinical Element Model is a two-level strategy consisting of an Abstract Instance Model, which defines a generic structure of data, and an Abstract Constraint Model, which defines how to constrain the Abstract Instance Model to represent specific data elements. Details are found in the Clinical Element Model Reference Document. [NEED REF]

The Constraint Definition Language (CDL) is a GE Healthcare-developed syntax for expressing constraint models conformant with the Abstract Constraint Model. Constraint models expressed in CDL are informally known as “Clinical Element Models” (CEMs). CDL is described in the “Qualibria Constraint Definition Language (CDL) Language Guide”. [NEED REF]

The Language Guide is the definitive CDL reference. In contrast, the purpose of this document is to provide model authors with guidelines for using CDL to consistently author CEMs. This document assumes the reader has or is developing a working knowledge of CDL. Examples are in CDL, and descriptions of how to accomplish modeling tasks are from the perspective of CDL.

### Abstract Instance Model

The CEM Abstract Instance Model is shown in Figure 1.

#### Type and Key

Figure . The CEM Abstract Instance Model

The type is a controlled terminology code acting as an identifier for the model. For example, the type “Heart Rate Measurement” is the identifier for the CEM model used for capturing heart rate measurements. The type is a code, a “designation”[[1]](#footnote-1) of which appears as the name of the model in a CDL file. (See “The use of Terminology in CEMs” for further explanation.)

The key is also a controlled terminology code. It represents the real world concept that is the focus of the model. For example, in the model representing a heart rate measurement, the key is a code meaning “heart rate measurement”, i.e., the clinical observation of a heart rate. (Contrast this with the “type”, which is a code representing a CEM model for heart rate measurements.) It maps to a code in a standard terminology like SNOMED CT or LOINC.

#### Value Choice

The “value” of a model is represented by the “value choice”. If the model has a single value, as in the case of heart rate measurements, for example, the value choice consists of a single “data” component. If the model’s value is composed of multiple components, as in the case of a patient, for example, whose “value” is the collection of name, address, birthdate, etc., the value choice consists of multiple “items”, each of which is itself defined as a CEM.

A model can have either “data” or one or more “items”, not both. The only exception to this rule is annotations (i.e., comments). (See the Comments section.)

A CEM can be thought of essentially as a name/value pair, or a question/answer pair. In this view, the key is the “name” or “question” and the value choice (data or items) is the “value” or “answer”.

For example, in the heart rate measurement model, the key, or “name”, is “Heart Rate Measurement”. (The question would be, “what is the patient’s heart rate measurement?) In this example, “data” (the value choice) holds the value of the heart rate measurement (the answer to the question, “what is the patient’s heart rate measurement?”).

The terminal nodes within any CEM (the value choice down at the furthest level of nesting) are “data” elements. In other words, every CEM and every element within a CEM eventually ends at a “data” element. Each data element is a CEM data type. The CEM data types are based on the HL7 v3 R2/ISO 21090 data types. They are described in the CDL reference guide.

#### Qualifiers

Qualifiers (represented by “quals” in the diagram) provide more information about the value choice, without appreciably changing the semantics of the value. Some examples are the body location from which a heart rate measurement was taken, or the severity of a disease.

Qualifiers are themselves separately-defined CEMs.

#### Modifiers

Modifiers (represented by “mods” in the diagram) appreciably change the meaning of the value choice. Only a few examples have been identified. They are:

* Subject, used to indicate that an instance of a model pertains to a subject other than the patient of record.
* Negation: Negation is used to negate the assertion being made by the key and value.
* Uncertainty: This modifier is used to express levels of uncertainty.
* RiskFor: This modifier is used to indicate that the patient is at risk for a given disease or disorder, as opposed to actually having the disease or disorder.
* Goal: This modifier is used to indicate that the stated event or observation is a goal for the patient, as opposed to a documentation of an actual occurrence or observation.[[2]](#footnote-2)

Modifiers are themselves separately-defined CEMs.

#### Attributions

Attributions (represented by “attrs” in the diagram) provide the “who”, “when”, “where”, and “why” somehow related to the model instance. For example, the Observed Attribution represents who made the observation that resulted in the model instance as well as when, where, and why the observation was made. The Documented Attribution provides information related to the documentation of the model instance’s information. The SpecimenCollected Attribution provides information related to the collection of the specimen whose analysis resulted in the model instance’s information. All Attributions share the same core structure.

Attributions are themselves separately-defined CEMs.

### Abstract Constraint Model

The Abstract Constraint Model outlines the aspects of the Abstract Instance Model that may be constrained in order to define the structure and semantics of a particular clinical element. The most commonly used constraints are listed below, using Figure 2 as an illustration[[3]](#footnote-3).

* Constrain the type of a CEM to be a specific code.
  + In Figure 2, the line

model HeartRateMeas is statement {

constrains the type to be “HeartRateMeas”, i.e., the model whose name is “HeartRateMeas”.

* Constrain the key of a CEM to be a specific code or value set.
  + In Figure 2, the line

key code(HeartRate\_KEY\_ECID);

constrains the key to be the code represented by “HeartRate\_KEY\_ECID”.

* Constrain the value choice of a CEM to either “data” or “items”, as appropriate for a specific data element.
  + In Figure 2, in the line

data PQ value.min(0) value.max(300)

“data ” constrains the value choice to be “data” (instead of a list of items).

* Constrain the “data” of a CEM to a specific data type as appropriate for a specific data element.
  + As stated above, in Figure 2, in the line

data PQ value.min(0) value.max(300)

“data PQ” constrains the data type to be PQ (Physical Quantity).

* Constrain the universe of all items, qualifiers, modifiers, and attributions to just the set of qualifiers, modifiers, and attributions applicable to a specific clinical element.
  + Additionally, the cardinalities of the various items, qualifiers, modifiers, and attributions may be constrained.

Figure . Constraints in a CEM

**model** HeartRateMeas **is** **statement** {

/\* constrains “type” to be “HeartRateMeas” \*/

/\* constrains “reference class” to be “statement” \*/

**key** **code**(HeartRate\_KEY\_ECID);

/\* constrains “key” to be the code “HeartRate\_KEY\_ECID” \*/

**data** PQ value.min(0) value.max(300)

/\* constrains the “value choice” to be “data” \*/

/\* constrains the data type of “data” to be “PQ” \*/

/\* constrains the valid range of data to be 0-300 \*/

**qualifier** MethodDevice methodDevice **card**(0..1);

**qualifier** BodyLocation bodyLocation **card**(0..1);

**qualifier** BodyPosition bodyPosition **card**(0..1);

/\* constrains the valid set of qualifiers to be {“MethodDevice”, “BodyLocation”, “BodyPosition”} \*/

/\* constrains the cardinalities of the qualifiers \*/

**attribution** Observed observed **card**(0..1);

/\* constrains the valid set of attributions to be {Observed} \*/

**constraint** methodDevice.CD.**code**.**domain** (HeartRateMetDev\_VALUESET\_ECID);

/\* constrains the valid set of coded values for “methodDevice” to be

those defined by the value set named HeartRateMethodDevice\_VALUESET\_ECID, which is a subset of the value set defined in the MethodDevice model \*/

}

* + In Figure 2, the following lines constrain the universe of qualifiers down to the set of MethodDevice, BodyLocation, and BodyPosition to define the valid qualifiers (and their cardinalities) for a HeartRateMeas instance:

qualifier MethodDevice methodDevice card(0..1);

qualifier BodyLocation bodyLocation card(0..1);

qualifier BodyPosition bodyPosition card(0..1);

The line

attribution Observed observed card(0..1);

constrains the universe of attributions down to just Observed to define the valid attributions for a HeartRateMeas instance. It also constrains the cardinality of Observed.

* Constrain the allowable values for a coded attribute[[4]](#footnote-4) in a CEM to a specific code or set of codes contained in a “value set”.
  + Additionaly, when a CEM is used in another model, the containing model may constrain the contained CEM’s values beyond what was declared in the contained CEM’s definition.
* Constrain the unit of measure for a PQ[[5]](#footnote-5) (Physical Quantity) data type.
  + A “normal” unit specifies the unit of measure to which instances of the model should be normalized to upon storage.
  + A “domain” of units specifies a value set representing the acceptable units of measure that may be received or displayed for an instance of the model.
* Constrain the physiologic range of a “data” component in a CEM.
  + In Figure 2, in the line

data PQ value.min(0) value.max(300)

“value.min” and “value.max” constrain the valid range of data[[6]](#footnote-6).

These elements will be discussed in greater detail in the CDL Elements of a CEM section.

## The Use of Terminology in CEMs

### Code Systems and CEMs

Coded, controlled terminology is used throughout CEMs wherever possible and practical. In the ECIS implementation of CEMs (where ECIS is the Enterprise Clinical Information System which Intermountain Healthcare and GE Healthcare jointly developed), references to codes in CEMs are assumed to refer to codes in the “ECIS” code system. Codes (concept identifiers, or just “concepts”) within the ECIS code system are known as “ECIDs” (ECIS Concept Identifiers).

The intent is to create ECIS code system concepts based on standards wherever possible, with mappings to those standards so the standards can be used in exchange as needed. The intent is not to propose the ECIS code system as a replacement for any existing standards.

The codes and value sets of the ECIS code system are created as needed by the CEMs; i.e., the sole purpose of the ECIS code system is to provide the coded terminology needed by the CEMs[[7]](#footnote-7). A type code or key code is created when a CEM model calls for it. A value set is created and maintained when a CEM needs a list of allowable values for its “data”.

The concepts and value sets of the ECIS code system are maintained in a terminology server, as opposed to being “hard-coded” codes or enumerations in the CEM files. This separation of the models from the terminology[[8]](#footnote-8) allows for flexibility and reuse. This separation also allows a different (non-Qualibria) implementation of CEMs to bind models to code systems other than the ECIS code system.

### References to Codes in CEMs

A concept may have multiple designations (text representations). A CEM refers to a concept by one of its designations[[9]](#footnote-9). The convention is to make the designations that appear in CEMs end in “\_ECID”. For example, in the heart rate measurement model of Figure 2, the string, “HeartRate\_KEY\_ECID” references a coded concept that means, “heart rate measurement” and, at least in the ECIS implementation, is the designation of a concept in the ECIS code system. This ECID maps to the LOINC code for “heart rate measurement”. In another implementation, even though the designation ends in “ECID”, this reference in the model might be bound directly to the LOINC code.

The exception to this convention of using “\_ECID” to refer to codes is the name of a CEM. The name of a CEM is also a code in the ECIS code system, but the reference to that code in the CEM leaves off the “\_ECID” ending. As an example, Figure 3 shows the first line of the Heart Rate Measurement CEM (HeartRateMeas). In this example, “HeartRateMeas” refers to the type code. “HeartRateMeas” is a designation of that code, which serves as the identifier of the model. These type codes will not be found in any other non-ECIS code system, since they are the identifiers of CEM models.

These references to terminology are discussed further in the References to Terminology section within the Conventions section.

### Value Sets in CEMs

Value sets warrant special consideration. A value set is the set of codes that constitutes the valid coded values for a coded attribute of a CEM. For example, the Administrative Marital Status value set, containing codes for “married”, “single”, “divorced”, etc., defines the valid values for the AdministrativeMaritalStatus component CEM[[10]](#footnote-10). A value set (the set itself, separate from its members) is represented by a coded concept in the ECIS code system. That concept, as is true for all concepts, may be represented by multiple designations in the terminology server which houses the ECIS code system. One designation of the value set concept will be a string ending in “\_VALUESET\_ECID”. This is the designation used in a CEM to reference the value set. The AdministrativeMaritalStatus example is shown in Figure 4.

## The Constraint Definition Language (CDL)

Figure . Example of a type reference

**model** HeartRateMeas **is** **statement** {

. . .

Figure . Example of a value set reference.

**model** AdministrativeMaritalStatus **is** **component** {

**key** **code**(AdministrativeMaritalStatus\_KEY\_ECID);

**data** CD **code**.**card**(1) **code**.**domain**(MaritalStatus\_VALUESET\_ECID);

}

The Constraint Definition Language (CDL) is an Implementation Technology Specification of the CEM model, i.e., CDL is a grammar for expressing constraints according to the Abstract Constraint Model. Many other formalisms (Microsoft Word documents, UML/OCL, XML-based languages, OWL, etc.) are conceivable. CDL was developed by GE Healthcare to take advantage of tooling available for developing parsers and compilers of context-free grammars.

This document explains features of CDL that are important in common CEM authoring. It is not intended to be a complete description of the language. For more information about CDL, the reader is invited to consult the Qualibria Constraint Definition Language (CDL) Language Guide. As of this writing, the version dated October 22, 2010 is the most recent version.

# Authoring CEMs in CDL

## Authoring Tools

At present, CDL CEMs are authored using a CDL IDE tool, an eclipse-based plugin that connects with the CEM repository for check-in/check-out and supports certain CDL syntax helps. The tool was developed by GE Healthcare and is not at present available to the public. A strategy for enabling broader public authoring of CDL models is being developed.

CDL is a text based language, so an option, although certainly not ideal, is to author CDL using a text editor.

## Modes of Authoring

There are three possible modes of authoring a new CEM: 1) “from scratch”, 2) using a template, or 3) copying from another CEM.

### Authoring “From Scratch”

“From scratch” refers to starting a CEM without basing the CEM on an existing CEM or template.

### Authoring using a Template

CDL Templates provide “patterns” for common kinds of CEM models. Templates exist for:

* Measurements
* Evaluations
* Clinical Assertions
* Impressions

For a discussion of these kinds of CEMs, see the Statements section of this document.

The model author identifies the kind of model she needs to create. She then opens the appropriate template, saves it as a new CEM (giving it an appropriate name), and modifies the template appropriately given her modeling requirements.

### Authoring using another CEM

Perhaps the most common authoring mode is to author using another CEM. In this mode, the author finds an existing CEM similar to the new one he will be creating and saves it under another file name. He then changes the name of the model and makes any other appropriate modifications.

## CDL Data Types

The CEM strategy defines logical data types to be used as the “data” component of a CEM. The CEM data types are based on the HL7 v3/ISO data types . [NEED REF] A full explanation of the data types as implemented in CDL is presented in the Qualibria Constraint Definition Language (CDL) Language Guide. The most commonly used data types are briefly described below.

### CD/CO

The CD and CO data types are used when a CEM’s “data” needs to be drawn from a controlled set of codes. Both the CD and CO data type have the following components:

* **code**: Code is a string assumed to be a unique identifier in a code system.[[11]](#footnote-11)
* **codeSystem**: CodeSystem is a code[[12]](#footnote-12) representing the code system from which the CD.code (or CO.code) is drawn. This component is suppressed in the ECIS implementation of the language, since only the ECIS code system is used for patient data storage.
* **codeSystemVersion**: CodeSystemVersion is a string representing the version of the codeSystem. This component is supressed in the ECIS data types. This component is suppressed in the ECIS implementation of the language, since the ECIS code system is not versioned and changes to individual concepts are assumed to be backward compatible.
* **originalText**: OriginalText is a string that represents CD.code (or CO.code) and is what a user actually saw or selected. It is not to be confused with a “display text” that is a default string which the code system associates with the code. OriginalText is “instance data”; it is the string that a user saw or selected when the instance of data was stored. In contrast, maintainers of a code system associate a “display text” with each concept, but this is not “instance data”. It is static knowledge that does not change depending on the instance. Such a display text is not captured in the data type. It is assumed that the system can “look up” such a string via terminology services.
* **codingRationale**: CodingRationale is a code[[13]](#footnote-13) that indicates the origin of the CD.code (or CO.code) component (e.g., whether it was the code originally sent/submitted, a code translated from the original, a code set by the receiving system, etc.).
* **translation**: Translation captures the data that was actually sent/submitted by a sender/application since a terminology mapping may have been performed to arrive at the code and codeSystem components. Its structure[[14]](#footnote-14) mimics that of the CD (or CO) data type itself, except it contains no translation (to prevent recursion).

The CO data type has additional semantics. It is used when there is an implied ordering associated with the codes in a value set. It is assumed that the ordering is represented in terminology (in relationships or properties). For example, suppose codes “1+” and “2+” are both valid values for a CEM (i.e., members of the value set referenced by the CEM). It would be expected that a terminology property or relationship would be associated with the “1+” and “2+” codes in the terminology server (e.g., a relationship relating “1+” and “2+” with an “IS-LESS-THAN” relationship). The fact that the CEM’s data type is “CO” (as opposed to just “CD”) is an indication to an application that it may query the underlying terminology to retrieve ordering information[[15]](#footnote-15).

The CO data type also enables mathematical operations to be performed on its instances. It does this via the “value” component, which the CO contains in addition to the components described above.

* **value**: Value is a number[[16]](#footnote-16) that enables mathematical operations to be performed on CO instances. For example, if instances of a CEM element of type CO are retrieved, and their “CO.code” values are “P”, “Q”, and “R”, and their “CO.value” values are “1”, “3”, and “2”, an application would be able to calculate the average of the values ((1+3+2)/3). Two points are worth noting about CO.value:
  + The numeric value associated with a code is actually static knowledge, not instance-specific data. In the example above, “1” is always associated with the code “P” – different instances of “P” for different patients at different points in time all have the same numeric value. Hence, it is expected that in the terminology server, each code that can be used in a CO would have a numeric property associated with it. Further, this “value” component actually is a convenience denormalization – it could just as well have been omitted from the data type, requiring an application to “look up” the numeric value associated with “P” in the terminology server instead of storing it with every instance[[17]](#footnote-17).
  + Given the above, the expectation is that storing a CO.value is performed by calling the terminology server and looking up the numeric value associated with the given code[[18]](#footnote-18). The CO.value is not set through user input.
  + CO.value need not be an ordinal number; its type is “double”.  The CO’s ordering information and its numeric property may be the same, but need not be.

### II

The II (Instance Identifier) data type is used to capture an “external” identifier, i.e., an identifier assigned and managed by an organization external to the EMR and exposed for business purposes, as opposed to an “internal” identifier assigned by the EMR and not generally exposed to applications and services. A Patient EMPI or medical record number, a laboratory accession number, an insurance account number, a social security number, and an order management system order identifier are all examples of external identifiers.

* **extension**: The extension is the identifier (a string) assigned by the external system.
* **type**: Type is a code[[19]](#footnote-19) representing the type of identifier (“medical record number”, “accession number”, etc.
* **correlationId**: CorrelationId is a string representing the assigning authority of the identifier.

### INT

NEEDS COMPLETION

### PQ

The Physical Quantity (PQ) data type is used for numeric data to which a unit of measure applies. The PQ has the following components:

* **codingRationale**: CodingRationale is a code[[20]](#footnote-20) that indicates the origin of the PQ.unit component (e.g., whether it was the code originally sent/submitted, a code translated from the original, a code set by the receiving system, etc.).
* **value**: Value is the numeric[[21]](#footnote-21) value of the PQ.
* **storagePrecision**: StoragePrecision[[22]](#footnote-22) captures the number of significant floating point digits in the stored instance. In some implementations, this precision would be lost if it were not explicitly captured.
* **operator**: Operator is a code[[23]](#footnote-23) which represents “>”, “<”, “>=”, “<=”, or “=”[[24]](#footnote-24). In conjunction with PQ.value, it is used to represent, for example, “> 250”, “<= 125”, etc.
* **unit**: Unit is a code[[25]](#footnote-25) representing the unit of measure associated with PQ.value.
* **unitOriginalText**: UnitOriginalText is a string representing PQ.unit and is what a user actually saw or selected for the unit of measure.
* **originalText**: OriginalText is a string representing the entire PQ (value and unit) and is what a user actually saw or selected.
* **translation**: Translation captures the data that was actually sent/submitted by a sender/application, since unit conversion may have been performed to arrive at PQ.value and PQ.unit. Its structure[[26]](#footnote-26) mimics that of the PQ data type itself, except it contains no translation (to prevent recursion).

### REAL

NEEDS COMPLETION

### ST

The ST data type is used for text (string) data. Its components are described below:

* **value**: Value is the text that needs to be captured.
* **language**: Language is a code[[27]](#footnote-27) that represents the written language pertaining to ST.value.
* **translation**: Translation captures the data that was actually sent/submitted by a sender/application, since language translation may have been performed to arrive at ST.value. Its structure[[28]](#footnote-28) mimics that of the ST data type itself, except it contains no translation (to prevent recursion).

### TS

TS is the timestamp data type. It is used for both dates and dates/times.

* **value**: Value is the date/time of the timestamp[[29]](#footnote-29). The timestamp may be captured in a format such as YYYY-MM-DDTHH:mm:ss. If the value specifies just a date (without specifying a time), the time portion is left off, e.g., “2012-05-12”. If the value is only precise to the hour (not the minute and second), the minute and second are left off, e.g., “2012-05-12T07”.
* **storagePrecision**: StoragePrecision captures the number of significant digits of the calendar representation. NEEDS COMPLETION
* **timezone**: Timezone is a code that captures the local timezone to which the TS.value instance pertains. This is important since in most implementations TS.value has been normalized to a specified timezone. Explicitly storing the local timezone of the instance allows conversion back to the local timezone for display or interpretation. For example, if a TS.value captures “201205121800” (UTC), a timezone code of “CDT” allows one to know that the local time was 1pm. This is important for display and for certain interpretations of the event (e.g., knowing that the event occurred at midday may be important). If an implementation stores an offset along with TS.value, the local time of day is now explicit, but the timezone is still possibly ambiguous. For example, “201205121800-0500” (meaning UTC – 5 hrs), may refer to the Eastern Standard Time (EST) or Central Daylight Savings Time (CDT).
* **operator**: Operator is a code[[30]](#footnote-30) which represents “>”, “<”, “>=”, “<=”, or “=”[[31]](#footnote-31). In conjunction with TS.value, it is used to represent, for example, “> 2012-03-16” (i.e., “later than” 2012-03-16), “<= 2010-01-01” (i.e., “earlier than” 2010-01-01), etc.
* **originalText**: OriginalText is a string representing what a user actually saw or selected, since in some cases, TS.value will be a reformatted and a timezone-converted version of what the user actually saw or selected.

## CDL Elements in a CEM

The elements of a CEM were presented in the Clinical Element Models section. Their expression in CDL will be examined in more depth in this section.

“Line”, “element”, and “statement” will be used interchangeably in this document to mean a line of CDL. “Statement” will only be used in this context when it will not be confused with the “statement” reference class or a CEM constraining the “statement” reference class.

### General CDL rules

* Each CDL file defines a single CEM[[32]](#footnote-32).
* Comment blocks in CDL files begin with “/\*” or “/\*\*” and end with “\*/”. Individual lines within the comment block begin with “\*”.
* Regardless of model type, the copyright statement, import statements (if the model references any other models), the header, and the model name appear. Whether other elements appear in the CEM depends on the type of model. These rules are described in the various sections of the CEM structures section.
* The actual model content appears after the header.
  + The first line of the content is the model name.
  + Following the model name, the body of the model is enclosed in opening and closing braces (“{” and “}”).
  + The content between the braces defines the logical structure of the model.
* Each import statement and each statement within the body of the CEM ends in a semicolon.

A schematic of a CEM model is shown in Figure 5.

### Copyright statement

The copyright statement is a comment in CDL and appears at the beginning of each CDL file, as shown below.

/\*

\* Copyright Â© 2009-2011 General Electric Company

\* All Rights Reserved

\*/

### Import statements

A CEM may reference another CEM when:

* it extends/restricts another CEM
* it uses another CEM as a qualifier, item, modifier, or attribution
* (in the case of associations) it specifies another CEM as a source or target

When a CEM references another CEM, it must “import” the other CEM using an import statement, as shown here:

**import** SomeModel;

where “SomeModel” is the name of a CEM. Import statements are listed at the beginning of a CDL file, after the copyright statement. Each import statement ends in a semicolon.

Copyright statement (commented)

Import statement(s)

Header (commented)

Model name {

Body

}

### Header

The header is a series of comments containing “metadata” pertaining to the CEM. An example is shown in Figure 6. The elements of the header are:

* Description:
  + Model Description: The description starts with the name of the model, and then on the next line contains a description of the model. Model descriptions should be as complete as possible. Listing of sample values helps reader understanding.
  + Qualifier/item/modifier/attribution description: Qualifiers items, modifiers, and attributions are described in their own files, and hence do not need to be re-described in a model that references them. Sometimes, however, the current model’s usage of another model as a qualifier, item, modifier, or attribution warrants further contextual explanation of the qualifier, item, modifier, or attribution. For example, the “BodyLocation” model is described as “a particular place in the body”. But when it is used as a qualifier in the HeartRateMeas model, its contextual meaning becomes “the particular place in the body from which a heart rate is measured”. When it is used as a qualifier in the PainAssert model, its contextual meaning becomes, “the particular place in the body at which pain is felt”. In such cases, the contextual meaning of the qualifier, item, modifier, or attribution should be described in the referencing model.
* Author: The author of the model is indicated in the header with a “@author” tag, followed by the author’s name, in a First Initial – Last Name concatenation. Only one author instance is expected.

Figure . Schematic of a CEM.

* Creation Date: The date of creation of the CEM is indicated in the header with a “@createdate” tag, followed by a date in MM/DD/YYYY format. Only one createdate instance is expected.
* Version: The version of the model is indicated in the header with a “@version” tag, followed by a version number. The versioning system in CEMs is currently only loosely defined, encouraging modelers to increment the number each time a change is made. Only one version instance is expected.
* Status: The status of the model is indicated in the header with a “@status” tag, followed by a status in all capital letters. The state machine of CEMs is currently only loosely defined; ACTIVE was used for models that were to be released to the Qualibria product; PROPOSED appears in all others. Only one status instance is expected.
* Usage: The usage of the model is indicated in the header with a “@usage” tag, followed by a usage in all capital letters. Until further notice, use “STORAGE” for all CEMs. Only one usage instance is expected.
* Context: The context of the model, i.e., the product or program that it is expected will use the model, is indicated in the header with a “@context” tag, followed by a context in all capital letters. Multiple instances of context may appear, each with its own “@context” line.
* Note, Issue, and Change: Notes are any pieces of information pertinent to development of the model. Issues capture information regarding problems or things that still need to be addressed or changed. Changes capture modifications made to the model. Multiple instances of all three may appear. All three have the same format, namely:
  + A tag (“@note”, “@issue”, or “@change”)
  + (following an indentation) a “@date” tag, followed by the date of the note/issue/change in MM/DD/YYYY format
  + (following an indentation) a “@modeler” tag, followed by the name of the person initiating the note/issue/change, in a First Initial – Last Name concatenation
  + (following an indentation) a “@value” tag, followed by text describing the note/issue/change

Figure . Header example.

/\*\*

\* COREDiseaseDisorder

\* This model captures a statement that a disease or disorder or

\* finding is present in a patient.

\* @author CTilley

\* @createdate 12/16/2011

\* @version 1.0

\* @context SHARP

\* @status PROPOSED

\* @note

\* @date 12/16/11

\* @modeler CTilley

\* @value The use case that motivated this model was natural language

\* processing of clinical notes.

\* @issue

\* @date 12/6/2011

\* @modeler TOniki

\* @value Should these core model constraints use the same key

\* codes as the parent?

\* @change

\* @date 12/16/2011

\* @modeler CTilley

\* @value added qualifier severity.

\* @change

\* @date 01/25/2012

\* @modeler TConway

\* @value changed Key code to DiseaseDisorder\_KEY\_ECID.

\*/

### Model Name

The first line after the header states the name of the model and its reference class:

**model** SomeModel **is** **SomeReferenceClass**

where “SomeModel” is the name of the model (i.e., a designation which represents the model’s type code), and SomeReferenceClass is the name of a valid reference class. (See the section.)

Optionally, the line may begin with “partial” if this model is to be used as the parent of another model. (See the section.)

Optionally, the line may end with an “extends” or “restricts” statement. (See the section.)

### key

The key statement, if present, is the first line of the CEM body. In the CEM Abstract Instance Model, “key” is defined as a CD data type; consequently, it has a “code” component. (See the section.) Hence, the keyword “key” can be thought of as representing a CD.code, i.e., the “code” component of the CD data type instance which represents the key.

That “code” may be constrained to a single code (i.e., for an instance to be valid, it must carry the specified code) or may be constrained to draw from a specified value set, or “domain” (i.e., for an instance to be valid, it may carry any member of a specified set of codes).

The first option – constraint to a single code -- is represented as follows:

**key** **code**(SomeConcept\_KEY\_ECID);

(A space between “code” and the opening parenthesis is optional.) This statement is shorthand for saying, “constrain the CD.code of the ‘key’ to be a single *code*, represented by the designation ‘SomeConcept\_KEY\_ECID’”.

The second option – constraint to a value set – is represented as follows:

**key** **domain**(SomeSet\_KEY\_VALUESET\_ECID);

(A space between “domain” and the opening parenthesis is optional.) This statement is shorthand for saying, “constrain the CD.code of the ‘key’ to be drawn from the *domain* (or valueset) represented by the designation ‘SomeSet\_KEY\_VALUESET\_ECID’.

### data

The data statement constrains the data type of the CEM to be a particular data type. Its format is:

**data** <data type>;

where <data type> specifies a valid data type, as described in the Data Types section of this document and in the Qualibria Constraint Definition Language (CDL) Language Guide.

Optionally, the data statement can also constrain data type-specific properties. By far the most common constraints are constraints of a CD.code to a code or value set (domain) and constraint of the CD.code’s cardinality. Similar to the key statement, the value set constraint takes the following format:

**data** CD **code**.**domain**(SomeSet\_VALUESET\_ECID);

(A space between “domain” and the opening parenthesis is optional.) This statement “binds” the CD.code (or “CD code”) of the “data” element to a particular domain (value set[[33]](#footnote-33)). A valid instance of data must have a CD.code that is a member of the value set represented by SomeSet\_VALUESET\_ECID.

Less frequently, a data’s CD.code is constrained to a single code:

**data** CD **code**.**code**(SomeCode\_ECID);

Constraining the CD.code’s cardinality is performed as follows:

**data** CD **code**.**card**(0..1) **code**.**domain**(SomeSet\_VALUESET\_ECID);

(A space between “card” and the opening parenthesis is optional.) This says that the cardinality of the data’s CD.code (or “CD code”) is 0..1. In other words, the CD.code in a valid instance is optional and may or may not be present. Specifying an optional cardinality permits the situation in which the desired value could not be represented by any code in the specified value set. In such instances, the CD.code may be left empty and it is expected that CD.originalText will capture text that represents the desired value[[34]](#footnote-34). This situation is known as “Coded With Exceptions” (CWE) in HL7.

In contrast, if the modeler determines that in a given situation an instance must absolutely draw from a specified value set, and not allow text instead of the CD.code, then the cardinality is set to “1”. This condition is the default – if no cardinality is expressed for CD.code, it is assumed required. This situation is known as “Coded No Exceptions” (CNE) in HL7.

### Qualifier/Item/Modifier/Attribution references

The body also specifies valid qualifiers, items, modifiers, and attributions for the CEM. It does so by referring to other CEMs, whose definitions are kept in separate CDL files. All these references have the same format:

**qualifier** SomeCEM someLabel **card**(<cardinality>)

**item** SomeCEM someLabel **card**(<cardinality>)

**modifier** SomeCEM someLabel **card**(<cardinality>)

**attribution** SomeCEM someLabel **card**(<cardinality>)

where:

* “SomeCEM” is a model name as found in the CEM’s own definition
* “someLabel” is any string of the author’s choosing, by which this element may be referenced within the CEM. By convention the label name is the same as the CEM name, except its first letter is not capitalized.
* “card” expresses the valid cardinality governing instances of this element within an instance of the CEM. The common values[[35]](#footnote-35) of <cardinality> are “1”, “0”[[36]](#footnote-36), “0..1”, “0..M”, and “1..M”. Any other range of two integers (e.g., “1-2”) is allowed, but is uncommon. The most common cardinalities are “0..1” and “0..M”, since in a logical model few elements are actually required[[37]](#footnote-37).

Examples are shown in Figure 7.

**qualifier** BodyLaterality bodyLaterality **card**(0..1);

**qualifier** DeltaFlag deltaFlag **card**(0..1);

**modifier** Subject subject **card**(0..1);

**modifier** Uncertainty uncertainty **card**(0..1);

**attribution** Observed observed **card**(0..1);

**attribution** ReportedReceived reportedReceived **card**(0..1);

**item** OrderItem orderItem **card**(1-M);

**item** PersonName personName **card**(1-M);

### Constraint Statements

A constraint statement[[38]](#footnote-38) has the form:

**constraint** <object to be constrained> <constraint value>

The object to be constrained is a constrainable “property” (called a “verb” in the Qualibria Constraint Definition Language (CDL) Language Guide) of a data type. The most common properties constrained using constraint statements are described below.

#### CD.code.code/CO.code.code

This type of constraint constrains a “CD.code” or a “CO.code” to a specific “code”. Its format is

**constraint** CD.**code**.**code**(SomeCode\_ECID);

An example is shown in Figure 8. In this example, the model is extending another (parent) model. It constrains the “data” (of type CD) of the parent to a specific code, with the statement

**constraint** CD.**code**.**code**(CerebrovascularDisease\_ECID);

Note that this constraint is used to constrain *another* model’s “data”; it might validly be used to constrain a model’s *own* “data” element, but the “data” constraint described earlier, of the form

**model** CerebrovascularDiseaseAssert **is** **statement** **extends** ClinicalAssert {

**constraint** CD.**code**.**code**(CerebrovascularDisease\_ECID);

**qualifier** AlleviatingFactor alleviatingFactor **card**(0-M);

**qualifier** ExacerbatingFactor exacerbatingFactor **card**(0-M);

**qualifier** Periodicity periodicity **card**(0-M);

. . .

}

**data** CD **code**.**code**(SomeCode\_ECID);

Figure . An example of a code constraint.

by convention is used instead.

#### CD.code.domain/CO.code.domain

This type of constraint further constrains a “CD.code” or a “CO.code” that was first constrained to a value set in another model. Its format is:

**constraint** <SomeObject.>CD.**code**.**domain**(SomeSet\_VALUESET\_ECID);

or simply

**constraint** CD.**code**.**domain**(SomeSet\_VALUESET\_ECID);

Two examples are shown in Figure 9. In this example, as in the one above, the model is extending another (parent) model. It further constrains the “data” (of type CD) of the parent to a more constrained value set, with the statement

**constraint** CD.**code**.**domain**(Headache\_VALUESET\_ECID);

Just as above, this constraint is used to constrain *another* model’s “data”; it might validly be used to constrain a model’s *own* “data” element, but the “data” constraint described earlier, of the form

**data** CD **code**.**domain** (SomeSet\_ECID);

by convention is used instead.

In Figure 9, the statement

**constraint** bodyLocation.CD.**code**.**domain**(HeadBodyLocation\_VALUESET\_ECID);

**model** HeadacheAssert **is** **statement** **extends** ClinicalAssert {

**constraint** CD.**code**.**domain**(Headache\_VALUESET\_ECID);

**qualifier** PainCharacter painCharacter **card**(0-M);

**qualifier** Duration duration **card**(0..1);

. . .

**constraint** bodyLocation.CD.**code**.**domain** (HeadBodyLocation\_VALUESET\_ECID);

}

is another example of a domain constraint. This statement further constrains the CD.code of the element known by the “bodyLocation” label in the parent model to the specified value set.

Figure . An example of a domain constraint.

Figure 10 illustrates further constraint of a value set first constrained in the qualifier’s own CEM definition. The statements

**constraint** bodyLocation.CD.**code**.**domain** (HeartChamber\_VALUESET\_ECID);

**constraint** deltaFlag.CD.**code**.**domain** (DeltaFlagNumericNom\_VALUESET\_ECID);

constrain the value sets that were first constrained in the BodyLocation and DeltaFlag CEMs to narrower value sets appropriate for use in the EjectionFractionMeas model.

#### PQ.unit.domain

This type of constraint specifies the set of units of measure that a system should expect to receive or display for an instance of the current model. Its format is:

**constraint** PQ.unit.**domain** (GeneralFractionUnits\_VALUESET\_ECID);

Alternatively, if the constraint is on a PQ defined in another model, the format is:

**constraint** <object>.PQ.unit.**domain** (GeneralFractionUnits\_VALUESET\_ECID);

An example is shown in Figure 10. In this example, the statement

**constraint** PQ.unit.**domain** (SomeSet\_VALUESET\_ECID);

**model** EjectionFractionMeas **is** **statement** {

**key** **code**(EjectionFraction\_KEY\_ECID);

**data** PQ;

. . .

**qualifier** BodyLocation bodyLocation **card**(0..1);

**qualifier** DeltaFlag deltaFlag **card**(0..1);

. . .

**modifier** Subject subject **card**(0..1);

**attribution** Observed observed **card**(0..1);

**attribution** ReportedReceived reportedReceived **card**(0..1);

**attribution** Verified verified **card**(0..1);

**constraint** bodyLocation.CD.**code**.**domain** (HeartChamber\_VALUESET\_ECID);

**constraint** deltaFlag.CD.**code**.**domain** (DeltaFlagNumericNom\_VALUESET\_ECID);

**constraint** PQ.unit.**domain** (GeneralFractionUnits\_VALUESET\_ECID);

**constraint** PQ.unit.**normal** (Percent\_ECID);

}

says that for an instance of this model, a system should accept any unit of measure in the “general fraction” value set. This also means that any unit of measure in that value set is suitable for display of instances of this model[[39]](#footnote-39). However, it is expected that before storage, if the submitted unit of measure is not the “normal” unit (see next section), a unit conversion will be performed.

Figure . Another example of a domain constraint and an example of PQ constraints.

All models whose “data” element is of data type PQ are expected to contain one of these constraint statements.

#### PQ.unit.normal

This type of constraint constrains a PQ.unit to a normalized unit of measure. This is the unit of measure that will always be stored for an instance of this PQ. If another unit is submitted, the expectation is that a unit conversion will be performed to the normal unit. Its format is:

**constraint** PQ.unit.**normal** (SomeCode\_ECID);

Alternatively, if the constraint is on a PQ defined in another model, the format is:

**constraint** <object>.PQ.unit.**normal** (SomeCode\_ECID);

Figure 8 illustrates this type of constraint with the following statement:

**constraint** PQ.unit.**normal** (Percent\_ECID);

This means that an instance of this model will be normalized to a unit of measure of “percent”.

All models whose “data” element is of data type PQ are expected to contain one of these constraint statements, unless the model will be extended/restricted, in which case the extending/restricting model is expected to declare the normal unit.

#### <any element>.card

This type of constraint constrains the cardinality of an element. Its format is:

**constraint** <object>.**card** (<cardinality value>);

Examples are shown in Figure 11. The statement

**constraint** order.sigText.**card** (0);

for instance further constrains the cardinality of the “order.sigText” element originally declared in the parent model (OrderContainer). In this particular case, the constraint in essence “deletes” the item from the valid set of items for the model. (This is a common use case for the cardinality constraint – to “constrain out” elements in the parent model when a child model is made.)

Note that the object being constrained can be specified by a nested path, as in the line

**constraint** order.orderItem.substitutionStatus.**card** (0);

which constrains out (i.e., constrains the cardinality to “0” for) the “substitutionStatus” item within the “orderItem” item within the “order” item of the model.

Figure . An example of cardinality constraints.

**model** OrderNursing **restricts** OrderContainer {

**key** **code**(OrderNursing\_KEY\_ECID);

**constraint** collectionPriority.**card** (0);

**constraint** readingRequest.**card** (0);

**constraint** order.actionVerb.**card** (0);

**constraint** order.sigText.**card** (0);

**constraint** order.routeMethodDevice.**card** (0);

**constraint** order.totalVolume.**card** (0);

**constraint** order.dispenseQuantity.**card** (0);

**constraint** order.refills.**card** (0);

**constraint** order.specimen.**card** (0);

**constraint** order.collectionMethod.**card** (0);

**constraint** order.transportMode.**card** (0);

**constraint** order.whoCollects.**card** (0);

**constraint** order.bodyLocation.**card** (0);

**constraint** order.device.**card** (0);

**constraint** order.directionsForRangeDosing.**card** (0);

**constraint** order.rateContainer.**card** (0);

**constraint** order.orderItem.formulation.**card** (0);

**constraint** order.orderItem.orderedDosageContainer.**card** (0);

**constraint** order.orderItem.ivComponentType.**card** (0);

**constraint** order.orderItem.activeIngredient.**card** (0);

**constraint** order.orderItem.filledSubstance.**card** (0);

**constraint** order.orderItem.substitutionStatus.**card** (0);

}

## Conventions

### Naming of Models

#### General Rules

A “type” code is an identifier (code) in the ECIS code system representing a CEM model[[40]](#footnote-40). For example, “7ccb2a8c-6de8-490b-9818-bda382bc313c” is the type code for the HeartRateMeas CEM. The CEM model name is actually a reference to the type code, i.e., it is a designation of the type code. For example, in the HeartRateMeas CEM, in the first line

**model** HeartRateMeas **is** **statement**

“HeartRateMeas” is the model name, and is a designation representing the type code “7ccb2a8c-6de8-490b-9818-bda382bc313c”.

Mode names should be in “camel case”, beginning with a capital letter. An exception to the camel case rule occurs when the model name contains an acronym, in which case a series of capital letters is permitted.

Models must be given a globally unique name less than 256 characters. Model names should be descriptive yet concise.

#### Special Endings

Several kinds of models that extend/restrict a parent model or that follow a template, are given special endings:

* The names of “procedure” models (i.e., models that extend the “Procedure” model) should end in “Proc”. (See the Procedures section.)
* The names of “assertion” models (i.e., models that 1) assert the existence of a condition in a patient or the occurrence of an event, 2) follow the pattern of key = “Assertion” and data = <the thing being asserted>) and 3) extend/restrict one of the parent Assertion models should end in “Assert”. (See the Assertions section.)
* The names of “evaluation” models (i.e., models in which the “key” is the name of a test/assessment/evaluation of some property/characteristic, usually of the patient, and “data” is the coded or text result of the test/assessment/evaluation) should end in “Eval”. (See the Evaluations section.)
* The names of “measurement” models (i.e., models in which the “key” is the name of a test/assessment/evaluation of some property/characteristic, usually of the patient, and “data” is the numeric or physical quantity result of the test/assessment/evaluation) should end in “Meas”. (See the Measurements section.)
* The names of “association” models (semantic links, panels, lists, collections) should have endings reflecting the type of association (“Link” for semantic links, “Panel” for panels, “List” for lists, and “Collection” for collections).
  + Naming conventions for semantic links are TBD.

#### Lab Result Models

Laboratory result models have a special convention. They are named using the components of the LOINC code that maps to the CEM’s key, appended with “LabObs”. The values of the six LOINC axes are concatenated to form the name. Examples are:

ABOGroupTypePtBldNomLabObs

CalciumMCncPtSerPlasQnLabObs

Glucose1HPost100GGlucosePOMCncPtSerPlasQnLabObs

PlateletsNCncPtBldQnAutomatedCountLabObs

NEEDS COMPLETION

#### Institution- and Implementation-specific Names

CEMs are logical models, and hence their names should not generally reflect a particular institution or implementation. Occasionally, though, an institution or implementation may need a specific constraint on an existing CEM. An implementation- or institution-specific model that restricts an existing CEM may conceivably reflect an implementation or institution in its name.

This might also be true for Panels. Panels are (sometimes arbitrary) collections of CEMs, and may sometimes represent implementation- or institution-specific use cases. Hence, their names may reflect those very specific use cases.

Even in these cases, though, if the specific grouping of CEMs reflected by the panel might conceivably be used in another use case, a generic name (not reflecting institution or implementation) should be selected. For example, suppose Central Valley Healthcare needs a panel containing Heart Rate, Blood Pressure, Height, and Weight. Even though Central Valley Healthcare is the motivator for creating the panel, this collection of CEMs might very well be useful in other settings also. Therefore, a name like “HeartRateBloodPressureHeightWeightPanel” would be preferable to a name like “CentralValleyHealthcareVitalsPanel”.

### References to Terminology

As explained in the References to Codes in CEMs section in the Introduction, every reference to a code in a CEM is assumed to be maintained as a designation of the code in the terminology server. These designations must be globally unique.

#### Codes

References to codes used as type codes (the identifiers for the models) are as described above in the Naming of Models section.

References to “keys” are of the form “XYZ\_KEY\_ECID”, where XYZ is in “camel case”, first letter capitalized. In the HeartRateMease CEM, the line

**key** **code**(HeartRate\_KEY\_ECID)

HeartRate\_KEY\_ECID is a designation of the key code.

All other references to single codes (with the exception of type codes, as described above) are of the form “XYZ\_ECID”, where XYZ is in “camel case”, first letter capitalized.

#### Value Sets

All references to value sets in models should be of the form, “XYZ\_VALUESET\_ECID”. The form “XYZ\_DOMAIN\_ECID” is deprecated. In the BodyLocation CEM, in the line

**data** CD **code**.**card**(0..1) **code**.**domain**(BodyLocation\_VALUESET\_ECID)

“BodyLocation\_VALUESET\_ECID” is the designation of the concept representing the body location value set.

If a value set is used as a key code in a model it should be of the form “XYZ\_VALUESET\_KEY\_ECID.

### Ordering of Elements

While CDL enforces no rules about ordering of elements within a CEM, the ordering convention is as follows:

* Key
* Data or items
* Qualifiers
* Modifiers
* Attributions
* Constraint elements (i.e., elements that begin with “constraint”)

## Modifications to Models

Modifications to a model that has been used for data storage may only be performed if they are “backward compatible”. The definition of a backward compatible change is operational; a change is backward compatible if a data instance that was validated against the model and found valid before the change would still be valid if it were validated against the model after the change. If a change would be non-backward compatible, it must not be made; instead, a new model must be created.

A backward compatible change may be characterized as a broadening change or an addition. In contrast a non-backward compatible change is a restricting change or a deletion. As examples, the following are backward compatible changes:

* Additions of a qualifier/modifier/attribution
* Widening of cardinality of qualifiers/modifiers/attributions

The following are examples of non-backward compatible changes which require creation of a new model instead of modifying an existing model:

* Change of data type
* Change of units of measure
* Deletion of qualifier/modifier
* Replacement of one qualifier/modifier/item with another
  + Note: this can be viewed as a deletion and an addition, and consequently, because of the deletion, is grounds for creation of a new CEM.
* Combination of multiple qualifiers into a single qualifier[[41]](#footnote-41)
  + Note: this can be viewed as multiple deletions and an addition, and consequently, because of the deletions, is grounds for creation of a new CEM.
* Splitting of a single qualifier into multiple qualifiers[[42]](#footnote-42)
  + Note: this can be viewed as a deletion and an addition, and consequently, because of the deletion, is grounds for creation of a new CEM.
* Restriction of cardinality of items/qualifiers

A more complete list of backward-compatible and non-backward-compatible changes is given in Appendix A – Model and Terminology Changes and Backward Compatibility.

An addition of concepts to a referenced value set is a broadening change that requires no modification to the CEM. A deletion of concepts from a referenced value set is a restricting change and hence, strictly speaking, should result in the creation of a new CEM. This point is under discussion.

At the time of this writing, no version numbering scheme for models exists.

# Creating Terminology for Use in CEMs

## Key Codes and Type Codes

NEEDS COMPLETION

## Value Sets

Value sets are created in conjunction with the models; in fact, the sole purpose of creating the ECIS code system was to provide the codes and value sets needed by the CEMs that were created. Creating models and terminology independently often creates a semantic mismatch.

NEEDS COMPLETION

## Further Constraint of Value Sets

The value set referenced in a CEM’s definition may be further constrained when that CEM is referenced by another CEM. For example, the MethodDevice component model references the MethodDevice\_VALUESET\_ECID, i.e., the valid values for “data”[[43]](#footnote-43) in the MethodDevice component are those constituting the valueset with designation MethodDevice\_VALUESET\_ECID. However, when MethodDevice is used (referred to) as a qualifier within another model, the referencing model may further constrain the MethodDevice value set to a subset more appropriate for use within the referencing model. The heart rate measurement model, for instance, constrains the valid set of values for MethodDevice to be the HeartRateMethodDevice value set, which contains just that subset of values from the MethodDevice value set that are appropriate for the measurement of a heart rate. This situation is illustrated in Figure 12. In the top of the figure, we see the MethodDevice model, in which data is constrained to the MethodDevice\_VALUESET\_ECID. At the bottom of the figure, we see the MethodDevice model being used within the HeartRateMeas model as a qualifier, and the statement

**constraint** methodDevice.CD.**code**.**domain** (HeartRateMethodDevice\_VALUESET\_ECID);

further constraining MethodDevice’s data from its original constraint to a further constrained value set[[44]](#footnote-44).

Sometimes the key of a model is constrained not to a specific code, but to a value set of codes. This happens especially when it is expected that the model will be used as the base for another, child model. (See the section entitled Inheritance via Extends and Restricts Statements.) In the child model, the key’s original value set is further constrained either to a specific code in the original value set or to a further constrained value set. Figure 13 illustrates this situation. The top of the figure shows that the partial model StandardLabObs has a key domain. The bottom on the figure shows the child model StandardLabObsQuantitative, which extends StandardLabObs. In StandardLabObsQuantitative, the statement

**model** MethodDevice **is** **component** {

**key** **code**(MethodDevice\_KEY\_ECID);

**data** CD **code**.**card**(0..1) **code**.**domain**(MethodDevice\_VALUESET\_ECID);

}

**model** HeartRateMeas **is** **statement** {

**key** **code**(HeartRate\_KEY\_ECID);

. . .

**qualifier** MethodDevice methodDevice **card**(0..1);

. . .

**constraint** methodDevice.CD.**code**.**domain** (HeartRateMethodDevice\_VALUESET\_ECID);

}

Figure . MethodDevice used in HeartRateMeas as an example of a value set constraint.

**key** **domain**(StandardLabObsQuantitative\_KEY\_VALUESET\_ECID);

further constrains the original key domain of StandardLabObs to the sub value set “StandardLabObsQuantitative\_KEY\_VALUESET\_ECID”.

In these cases of specifying an additional constraint on a value set, the new value set must represent a proper subset of the original value set.

A key statement may be restricted by another key statement provided the constraint described is a restriction on the original constraint.

**partial** **model** StandardLabObs **is** **statement** {

**key** **domain**(StandardLabObs\_KEY\_VALUESET\_ECID);

. . .

**model** StandardLabObsQuantitative **is** **statement** **extends** StandardLabObs {

**key** **domain**(StandardLabObsQuantitative\_KEY\_VALUESET\_ECID);

. . .

# Inheritance in CEMs and CDL

CDL supports two types of inheritance[[45]](#footnote-45): 1) inheritance via Reference Classes, and 2) inheritance via “extends” and “restricts” statements.

## Inheritance via Reference classes

Reference classes in CDL are high level classes which CEM constraint models (i.e., the models representing Heart Rates, Medication Orders, Patients, Breath Sounds findings, etc.) constrain. In the heart rate measurement constraint model “HeartRateMeas” of Figure 2, for instance, the phrase “is statement” declares that HeartRateMeas inherits from the “statement” reference class. In so doing, it inherits the attributes of that class.

Figure . StandardLabObsPQ as an example of a further constraint of a key value set.

Reference classes attempt to capture the attributes common to a class of models. The attributes tend to be what might be regarded as “implementation” and “metadata” attributes (e.g., the time the instance was stored in the database, the identifier of the owning patient, etc.), as opposed to attributes of the logical model[[46]](#footnote-46).

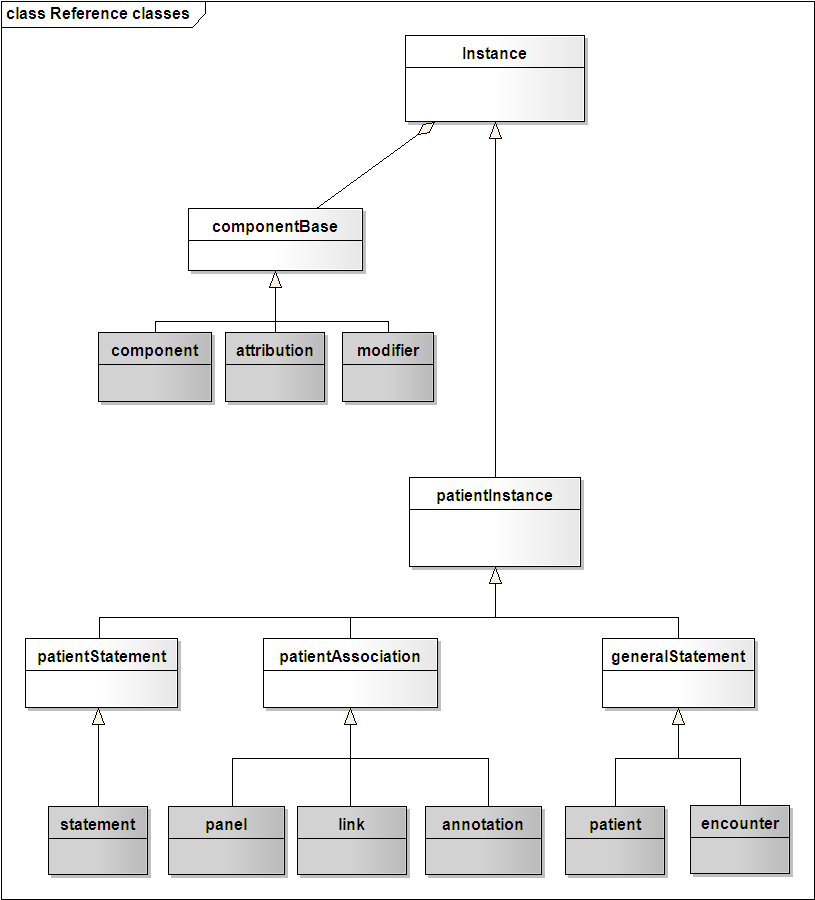
The most common reference classes used in CEM modeling are “statement” (used for all patient data statements – measurements, evaluations, and assertions), “component” (used as qualifiers or as items in compound statements), “modifier”, “attribution”, “panel”, and “semantic link”. Figure 7 shows a subset of the reference classes available to CEM models. The gray, “leaf” classes are those that a modeler would use in a CEM. The parent classes are not used directly.

The definitions of the reference classes for the Qualibria implementation[[47]](#footnote-47) are found in the Qualibria Constraint Definition Language (CDL) Language Guide.

Rules for using reference classes in authoring CEMs:

* The name of the reference model in the “is” statement must match a valid reference model.
* Use “statement” for patient data statements – models representing clinical data elements that “stand on their own”, e.g., measurements, findings, etc. (See the Statements section for a discussion of statements.)
* Use “component” for models that do not “stand on their own”; i.e., that only have meaning within the context of a statement and will be used as qualifiers/items. (See the Components section for an explanation of components and the Statement vs. Component section for a discussion of the differences between statements and components.)
* Use “modifier” for modifiers. (See the Modifiers section.)
* Use “attribution” for attributions. (See the Attribution section for a discussion of attributions.)
* Use “panel” for panels. (See the Panels section.)
* Use “link” for semantic links. (See the Semantic Links section.
* A model extending or restricting another (“base”) model must either include no “is” statement or declare the same reference class as the model it’s extending/restricting.

Figure . A subset of the reference classes available to CEM models.



## Inheritance via Extends and Restricts Statements

Inheritance via “extends” and “restricts” statements entails creating a “base” or “parent” model which will be used as the basis for additional “child” models.

The child models will nominally contain all of the elements in the base model. In a model definition, stating

restricts <name of base model>

may be interpreted as, “the model being defined contains all of the base model’s elements, but constraints will be placed on those elements as described in this definition”.

In contrast, the statement

extends <name of model>

may be interpreted as, “the model being defined contains all of the base model’s elements, plus any additional elements listed below”. When an author expects a model to be extended by a child model, the author should create the base model with the word “partial” in the first line. For example, in the StandardLabObs model, the first line is:

partial model StandardLabObs is statement {

The “partial” indicates that the model is not suitable for instantiation in a system[[48]](#footnote-48) – its purpose is just to serve as the basis for extension by other models.

A model definition that restricts another model may only constrain the base model’s elements. A model definition that extends another model may add elements in addition to also constraining the base model’s elements.

The example of Figure 15 illustrates. In the figure, the model OutputVolumeMeas represents the base model. (Note the base model is defined as “partial”, even though a child model can restrict a base model that is not defined as “partial”. Most often, a base model will not be instantiated – only the child models will – so declaring the base model as “partial” is common practice.)

The UrineOutputVolumeMeas model restricts OutputVolumeMeas, indicated by the “***restricts OutputVolumeMeas***” clause in the first line. The UrineOutputMeas model inherits all of the qualifiers, modifierss, and attributions of the OutputVolumeMeas model. In addition, it imposes three constraints:

* The key in OutputVolumeMeas is constrained to the value set represented by “OutputVolume\_KEY\_VALUESET\_ECID “. UrineOutputMeas further constrains the key by setting it to a specific code in that value set, “UrineOutputVolume\_KEY\_ECID”.
* The figure shows the definition of the MethodDevice component model, which is used as a qualifier in OutputVolumeMeas. With the statement

Figure . An example of restriction of a parent model

**“Base” or “parent” model (bold italics added):**

partial model OutputVolumeMeas is statement {

key domain(***OutputVolume\_KEY\_VALUESET\_ECID***);

data PQ;

qualifier MethodDevice methodDevice card(0..1);

qualifier SourceIdentifier sourceIdentifier card(0..1);

qualifier BodyLocationPrecoord bodyLocationPrecoord card(0..1);

. . .

constraint PQ.unit.domain (VolumeUnits\_VALUESET\_ECID);

constraint PQ.unit.normal (Milliliters\_ECID);

constraint methodDevice.CD.code.domain (***OutputMethodDevice\_VALUESET\_ECID***);

}

**Component models referred to in the base model (bold italics added):**

model MethodDevice is component {

key code(MethodDevice\_KEY\_ECID);

data CD code.card(0..1) code.domain(***MethodDevice\_VALUESET\_ECID***);

}

model BodyLocationPrecoord is component {

key code(BodyLocationPrecoord\_KEY\_ECID);

data CD code.card(0..1) code.domain(***BodyLocationPrecoord\_VALUESET\_ECID***);

}

**“Child” model (bold italics added):**

model UrineOutputVolumeMeas is statement ***restricts OutputVolumeMeas*** {

key code(***UrineOutputVolume\_KEY\_ECID***);

constraint methodDevice.CD.code.domain (***UrineOutputMethodDevice\_VALUESET\_ECID***);

constraint bodyLocationPrecoord.CD.code.domain (***UrineOutputBodyLocationPrecoord\_VALUESET\_ECID***);

}

data CD code.card(0..1) code.domain(***MethodDevice\_VALUESET\_ECID***);

in the MethodDevice definition, the value set is constrained to “MethodDevice\_VALUESET\_ECID “. When the OutputVolumeMeas model uses MethodDevice, the statement

constraint methodDevice.CD.code.domain (***OutputMethodDevice\_VALUESET\_ECID***);

further constrains the value set of the methodDevice qualifier to the value set represented by “OutputMethodDevice\_VALUESET\_ECID”. UrineOutputMeas inherits the methodDevice qualifier, and constrains the value set still further. With the statement

constraint methodDevice.CD.code.domain (UrineOutputMethodDevice\_VALUESET\_ECID);

it further constrains its value set to the value set represented by “UrineOutputMethodDevice\_VALUESET\_ECID”.

* The figure shows the definition for the BodyLocationPrecoord component model, which is used as a qualifier in OutputVolumeMeas. In BodyLocationPrecoord, the statement

data CD code.card(0..1) code.domain(***BodyLocationPrecoord\_VALUESET\_ECID***);

constrains the data to the value set represented by “BodyLocationPrecoord\_VALUESET\_ECID”. UrineOutputMeas inherits the bodyLocationPrecoord qualifier, but with the statement

constraint bodyLocationPrecoord.CD.code.domain (UrineOutputBodyLocationPrecoord\_VALUESET\_ECID);

it further constrains its value set to the value set represented by “UrineOutputBodyLocationPrecoord\_VALUESET\_ECID”.

Figure 16 illustrates a child model (AbdominalTendernessAssert) that extends a base model (ClinicalAssert). Note that ClinicalAssert is declared “partial”, which is required in order for AbdominalTendernessAssert to legally extend it. AbdominalTendernessAssert inherits all the elements of ClinicalAssert, and adds additional elements (qualifiers bodyLocationPrecoord, alleviatingFactor, and exacerbatingFactor, severity, and dateOfResolution). Note also that it constrains the data of the base model to “AbdominalTenderness\_ECID”. (A model that extends a base model is also allowed to make constraints on the base model.) With the statement

constraint bodyLocationPrecoord.CD.code.domain (Abdomen\_VALUESET\_ECID);

it constrains the newly added qualifier’s data, as any model is allowed to do.

#### Rules: inheritance via extends and restricts

* A model used as the base for another model that extends it must declare itself to be “partial”
* It is assumed that a “partial” model will not be instantiated.

Figure . An illustration of a child model extending a base model.

* A model that restricts a base model may not then be extended by another model

**“Base” or “parent” model:**

partial model ClinicalAssert is statement {

key code(Assertion\_KEY\_ECID);

data CD code.card(0..1) code.domain(ClinicalAssertion\_VALUESET\_ECID);

qualifier DateOfOnset dateOfOnset card(0..1);

modifier Subject subject card(0..1);

modifier Uncertainty uncertainty card(0..1);

modifier RiskForInd riskForInd card(0..1);

attribution Observed observed card(0..1);

attribution ReportedReceived reportedReceived card(0..1);

attribution Verified verified card(0..1);

}

**“Child” model (bold italics added):**

model AbdominalTendernessAssert extends ClinicalAssert {

constraint CD.code.code(***AbdominalTenderness\_ECID***);

qualifier BodyLocationPrecoord bodyLocationPrecoord card(0-M);

qualifier AlleviatingFactor alleviatingFactor card(0-M);

qualifier ExacerbatingFactor exacerbatingFactor card(0-M);

. . .

qualifier Severity severity card(0..1);

qualifier DateOfResolution dateOfResolution card(0..1);

constraint bodyLocationPrecoord.CD.code.domain (Abdomen\_VALUESET\_ECID);

}

* A model that extends a base model may also constrain the base model
* A model that restricts a base model may not also extend the base model

# CEM structures

The most commonly used CEM structures and their usage will be described below.

## Statements

A statement model, i.e., a CEM with an “is statement” declaration, describes a complete medical sentence that can be referenced in a standalone fashion. A statement’s “value” is composed of either a single “data” component[[49]](#footnote-49) or one or more “items” (where each item references a component CEM[[50]](#footnote-50)). A statement may not contain both a data component and an item or items[[51]](#footnote-51). In addition, a statement may contain qualifiers (which are references to component CEMs), modifiers (references to modifier CEMs[[52]](#footnote-52)), or attributions (references to attribution CEMs[[53]](#footnote-53)).

A statement may only be referenced in another model if that referencing model is an association (panel, collection, etc.). When an association references a statement, it refers to the statement as an item.

The CDL elements that make up a statement are as follows:

* The model name and reference class statement
* A key element (required unless the CEM is extending/restricting another CEM in which the key is stated
* data or items (required unless the CEM is extending/restricting another CEM in which the data or items are stated)
* qualifiers, modifiers, and attributions
* constraint elements

These CDL elements are summarized in Table 1.

|  |  |  |
| --- | --- | --- |
| Element | Explanation | Required? |
| **model** SomeModel **is** **SomeReferenceClass** | Constrains the type to be the code represented by the designation “SomeModel”; constrains the reference class to be “SomeReferenceClass”. (See the section.) | This is required in every model.  Optionally, the line may begin with “partial” if this model is to be used as the parent of another model. (See the section.)  Optionally, the line may end with an “extends” or “restricts” statement. (See the section.) |
| **key** **code**(SomeText\_KEY\_ECID)  OR  **key** **domain**(SomeSet\_KEY\_VALUESET\_ECID) | Constrains the key to be (in the first line) the code represented by the designation “Sometext\_KEY\_ECID” or (in the second line) a code drawn from the value set represented by the designation “SomeSet\_KEY\_VALUESET\_ECID”. | One or the other of these two statements is required in every model unless the model is extending/restricting another model (see the section).  If the model is extending/restricting another model, a key constraint is optional (used if the key of the parent model needs to be restricted). |

Table . Elements in CEMs Constraining Statement, Component, Modifier, Attribution, Patient, or Encounter Reference Classes

|  |  |  |
| --- | --- | --- |
| Element | Explanation | Required? |
| **data** <data type>  OR  **item** SomeCEM someLabel **card**(<cardinality>) | Constrains the CEM to have either data or one or more items.  In the case of data, also constrains the data to be of a certain data type.  In the case of an item, specifies a valid item for this CEM, and by so doing constrains the universe of items. | Either data or one or more item statements is required unless the model is extending/restricting another model.  If the model is extending/restricting another model, data is only permitted if it is constraining the parent model’s data.  If the model is restricting another model, item is not permitted.  If the model is extending another model, item is permitted (to “add” new items to the model).  In all cases of extension/restriction, the extension/restriction must not result in both data and items in the same model. |
| **qualifier** SomeCEM someLabel **card**(<cardinality>) | Specifies a valid qualifier for this CEM, and by so doing constrains the universe of qualifiers. | Optional, appears as needed to specify qualifiers |
| **modifier** SomeCEM someLabel **card**(<cardinality>) | Specifies a valid modifier for this CEM, and by so doing constrains the universe of modifiers. | Optional, appears as needed to specify modifiers |

|  |  |  |
| --- | --- | --- |
| Statement | Explanation | Required? |
| **attribution** SomeCEM someLabel **card**(<cardinality>) | Specifies a valid attribution for this CEM, and by so doing constrains the universe of attributions. | Optional, appears as needed to specify attributions |
| **constraint** <object to be constrained> <constraint> | Specifies an object to be constrained and the manner in which it is to be constrained. See accompanying text for further discussion.  Used most commonly for:   * further constraining a value set originally constrained in another CEM * further constraining the normal code or domain of a unit of measure originally constrained in another CEM * further constraining a cardinality originally constrained in another CEM | Optional, appears as additional constraint is needed |

### Statements Differentiated by Value Choice

One way to categorize statements is by their value choice – a single “data” or multiple “items”.

#### Simple Statement

A simple statement is a statement whose meaning is conveyed by a single value (i.e., its “value choice” is a single “data” component), with associated modifiers and qualifiers. The data’s type is a CDL data type.

An example of a simple statement is the *HeartRateMeas* CEM.

**model** HeartRateMeas **is** **statement** {

**key** **code**(HeartRate\_KEY\_ECID);

**data** PQ;

**qualifier** MethodDevice methodDevice **card**(0..1);

**qualifier** BodyLocation bodyLocation **card**(0..1);

. . .

**modifier** Subject subject **card**(0..1);

**attribution** Observed observed **card**(0..1);

. . .

}

The “value” of an instance of *HeartRateMeas* is conveyed by its “data” component.

#### Compound Statement

A statement whose meaning is conveyed by multiple values (with associated modifiers and qualifiers) is called a “compound” statement. Its value choice is a list of items rather than a single data component. The meaning of the compound statement is conveyed by the values of its items interpreted together.

An example of a compound statement is the *MechanicalVentilationPeriod* CEM.

**model** MechanicalVentilationPeriod **is** **statement** {

**key** **code**(MechanicalVentilationPeriod\_KEY\_ECID);

**item** StartTime startTime **card**(0..1);

**item** EndTime endTime **card**(0..1);

**qualifier** Comment comment **card**(0-M);

**attribution** Observed observed **card**(0..1);

. . .

}

The “value” of an instance of *MechanicalVentilationPeriod* is conveyed by the combination of the start time and end time.

### Statements Differentiated by Clinical Usage and Semantics

Another way to categorize statements is by their clinical usage and semantics. Statements can fall into several usage and semantics categories:

* Measurements
* Evaluations
* Assertions
* Procedures

Certain structural differences exist between these types of statements. At times it is difficult to choose between them.

#### Measurements

A measurement holds the name of a “test” in the key (e.g., “heart rate measurement”, “serum glucose lab test”, etc.) and the test result value in “data”. Another way to view the situation is the key holds a "question" (e.g., "what is the heart rate?", "what is the serum glucose?") and “data” holds the answer.

Data for measurement models is comprised of a PQ[[54]](#footnote-54) (physical quantity) or CO[[55]](#footnote-55) (coded ordinal) value, and data is often constrained to a valid unit of measurement domain. Any clinical element (a test, a study, a procedure, etc.) that fits this pattern of a name and a numeric value with a unit of measure is modeled with a measurement template. Figure 17 shows the heart rate measurement model as an example of a measurement CEM.

A Measurement CDL template exists, which contains the common qualifiers, modifiers, and attributions desired in a measurement model. The modeler can open this template, modify it as necessary, and save the result as a new model.

##### Calculations

A qualifier may be used to indicate that a model instance represents “calculated” data (as opposed to directly measured data). Either a true/false “calculated indicator” qualifier can be added to a model or a “method” qualifier, one of whose values is “calculated” can be added. Alternatively, the fact that a model instance represents calculated data can be captured in the name of the model, i.e., by pre-coordinating “calculated” into the name.

In the first case (using the qualifier), some model instances will represent calculated data while others will not (depending on the value of the qualifier). In the second case (precoordinating “calculated” into the name of the model), all instances of the model will represent calculated data – the semantics of the model itself are that the model represents calculated data. The choice of which approach to use is somewhat subjective, determined by anticipated usage.

If the common usage is that all instances – whether calculated or measured – are retrieved and viewed together, the model should represent the generic element, and “calculated” should be a qualifier. If the common usage is that calculated data are treated (retrieved, viewed, analyzed) separate from directly measured data, it implies that semantically, the calculated data and measured data are different, and two separate models should be used.

For example, calculated minute volumes, which are the *expected* measured minute volumes obtained from a ventilated patient are important to separate from the observed *actual* measured minute volumes. The *expected* minimum mandatory minute volume is calculated based on the set respiratory rate and tidal volume. Therefore, calculated minute volume and actual measured minute volume would be represented by different models.

Even though calculations are not technically “measurements”, their structure is consistent with the “measurement” template, and hence a calculation model ends in “Meas”. shows the CalculatedMinuteVolumeMeas model. Note “calculated” is precoordinated into the name of the model, and the model name ends in “Meas”.

**model** HeartRateMeas **is** **statement** {

**key** **code**(HeartRate\_KEY\_ECID);

**data** PQ

**qualifier** MethodDevice methodDevice **card**(0..1);

**qualifier** BodyLocation bodyLocation **card**(0..1);

**qualifier** BodyPosition bodyPosition **card**(0..1);

**attribution** Observed observed **card**(0..1);

**constraint** methodDevice.CD.**code**.**domain** (HeartRateMetDev\_VALUESET\_ECID);

**constraint** PQ.unit.**domain** (NumberRateUnits\_VALUESET\_ECID);

**constraint** PQ.unit.**normal** (PerMinute\_ECID);

. . .

}

Figure . The HeartRateMeas CEM as an example of a measurement.

##### NumericSettingMeas:

There is sometimes the need to document what an equipment parameter is “set” to, e.g., the respiratory rate setting on a mechanical ventilator. These cases are not measurements of patient parameters, but observations of machine settings. Nevertheless, because their structure follows that of the measurement template, they are grouped with measurement models, and their names end in “SettingMeas”.

Figure 8 shows the model used to capture a mechanical ventilator’s respiratory rate setting. A different model (the RespiratoryRateMeas model) is used to capture the ventilated patient’s actual respiratory rate.

#### Evaluations

An evaluation model is a model that evaluates an attribute or characteristic of a patient or a patient-related situation, drawing its value from a set of coded values. Hair color, urine description, cardiac rhythm, and cervical consistency are examples. In each case, the coded values are adjectives that describe the property.

The existence of the property in the patient is assumed, e.g., it is assumed that the patient has a hair color, and the model instance contains the “value” of the hair color (blond, brown, black, etc.).

In other cases, instead of evaluating or describing a patient property or characteristic, an evaluation describes or identifies the type of person, place, method, instrument, etc. involved in an event or action related to the care of the patient. Examples are the method (laparoscopy, open, etc.) by which a procedure is performed, the type of person (family member, clinician, etc.) accompanying a patient to an encounter, or the type of place (skilled nursing facility, home, etc.) to which a patient is discharged after an encounter[[56]](#footnote-56).

An Evaluation CDL template exists, which contains the common qualifiers, modifiers, and attributions desired in an evaluation model. The modeler can open this template, modify it as necessary, and save the result as a new model.

Figure 20 shows the MuscleToneEval model, an example of an evaluation model.

##### Guideline: Use of “Normal”

A commonly encountered question in modeling an evaluation is how to represent “normalcy”. The meaning of “normal” depends on the characteristic being evaluated. For instance, “normal” skin temperature is “warm”. “Normal” for nasal passage moisture is “moist”. Because of its ambiguity, it would be best to not store “normal”, and not present it as an option to users. However, in order to support real use cases and desired practices, we have resigned ourselves to including a value of “normal” in an evaluation’s value set if clinicians request it. This will avoid having to use two models to fulfill a single evaluation.

If it is used, though, it should be specifically defined, e.g., in the case of “skin temperature”, “normal” might be a designation of the concept, but the definition is “warm”.[[57]](#footnote-57) This also implies that rather than creating a universal “normal” concept, modelers should create multiple, more specific concepts (“warm”, “moist”, etc.) that can be used as appropriate in various value sets.

An extension of this discussion of “normalcy” is the desire to express an overall summary “impression” of “normal”, for example, in a back examination, an ear examination, etc. In this case, multiple properties of the organ or system are being evaluated simultaneously. In this case, the guideline is to create an “impression” model -- a simple model that expresses whether an organ or system is “normal” or “abnormal”[[58]](#footnote-58). Its structure is that of an "Evaluation", with the data constrained to the NormalAbnormal domain. As described above for including “normal” as a value in an evaluation model that evaluates a specific property, the guideline is to create specific “normal” concepts (one for a back examination, one for an ear examination, etc.) and define what “normal” means in each of those use cases.

An ImpressionEvaluation CDL template exists, which contains the common qualifiers, modifiers, and attributions desired in an impression evaluation model. The modeler can open this template, modify it as necessary, and save the result as a new model. The naming convention is no different than for any evaluation; the model ends in “Eval”.

#### Assertions

**model** CalculatedMinuteVolumeMeas **is** **statement** {

**key** **code**(CalculatedMinuteVolume\_KEY\_ECID);

**data** PQ;

**qualifier** MethodDevice methodDevice **card**(0..1);

**qualifier** ReferenceRangeNar referenceRangeNar **card**(0..1);

**modifier** Subject subject **card**(0..1);

**attribution** Observed observed **card**(0..1);

. . .

**constraint** PQ.unit.**domain** (SIVolumeUnits\_VALUESET\_ECID);

**constraint** PQ.unit.**normal** (Liters\_ECID);

}

Figure . The CalculatedMinuteVolumeMeas model as an example of a calculated model.

Assertions may:

**model** RespiratoryRateSettingMeas **is** **statement** {

**key** **code**(RespiratoryRateSetting\_KEY\_ECID);

**data** PQ;

**modifier** Subject subject **card**(0..1);

**attribution** Observed observed **card**(0..1);

**attribution** ReportedReceived reportedReceived **card**(0..1);

**attribution** Verified verified **card**(0..1);

**constraint** PQ.unit.**domain** (NumberRateUnits\_VALUESET\_ECID);

**constraint** PQ.unit.**normal** (PerMinute\_ECID);

}

Figure . The RespiratoryRateSettingMeas model as an example of a setting model.

* assert the presence (or absence) of a condition in a patient, for example:
  + ChestPainAssert asserts the presence of chest pain
  + EdemaAssert asserts the presence of edema
* assert the presence (or absence) of an apparatus, for example:
  + a monitoring or administration device, such as an oximeter or an infusion pump
  + a therapeutic apparatus, such as compression stockings or a warm blanket
* assert the occurrence (or non-occurrence) of an event in which the patient was involved, for example:
  + an auto accident
  + a natural disaster

**model** MuscleToneEval **is statement** {

**key code**(MuscleTone\_KEY\_ECID);

**data** CD **code.card**(0..1) **code.domain**(MuscleTone\_VALUESET\_ECID);

**qualifier** BodyLocation bodyLocation **card**(0..1);

**qualifier** BodyPosition bodyPosition **card**(0..1);

**qualifier** Aggregate aggregate **card**(0..1);

**qualifier** RelativeTemporalContext relativeTemporalContext **card**(0-M);

**qualifier** PatientPrecondition patientPrecondition **card**(0-M);

**modifier** Subject subject **card**(0..1);

**attribution** Observed observed **card**(0..1);

**attribution** ReportedReceived reportedReceived **card**(0..1);

**attribution** Verified verified **card**(0..1);

**constraint** bodyLocation.CD.**code.domain**

(MusculoskeletalBodyLocation\_VALUESET\_ECID);

}

Figure . MuscleToneEval as an example of an Evaluation.

The pattern followed for assertions is

* Include the term “Assert” at the end of the CE type name.
* The Key is set to “Assertion\_KEY\_ECID”.
* The Data is a CD, set to a code representing the condition/event being asserted as existing or occurring.

An Assertion CDL template exists, which contains the common qualifiers, modifiers, and attributions desired in an assertion model. The modeler can open this template, modify it as necessary, and save the result as a new model.

See Appendix B – Rationale for Assertion pattern for a discussion of the rationale.

##### Clinical Assertions

Clinical Assertions assert the existence of clinical conditions, diseases, etc. in the patient. The partial model ClinicalAssert (whose key is “Assertion”) is used as the parent. A specific Clinical Assertion extends ClinicalAssert with any additional qualifiers necessary for the specific data element. It also constrains the data code of ClinicalAssert to a code representing the condition or disease being asserted.

Figure 21 shows the example of PainAssert. Other examples are CoughAssert, NauseaAssert, and DiabetesMellitusTypeOneAssert.

##### Event Assertions

Event Assertions are used to assert that an event involving the patient occurred, e.g., auto accident, poisoning, burns, etc. It is expected that the concepts used to populate “data” would be mappabe to concepts in the SNOMED event domain.

It might be noted that procedure models also represent events (ventilator setup, wound dressing change, IV placement, etc.) The difference is that Procedures represent some healthcare intervention performed during the management of care. Consequently, a “performer” of the procedure is commonly captured. In contrast, events modeled by Event Assertions happened to the patient outside the care of the patient (in fact, they are often what caused the patient to require care), and there is no performer or the performer is not pertinent information.

A specific Event Assertion extends EventAssert with any additional qualifiers necessary for the specific data element. It also constrains the data code of EventAssert to a code representing the kind of event being asserted.

Figure 23 shows MotorVehicleCrashAssert as an example of an Event Assertion.

##### Observation Assertions

Observation Assertions are used to assert the presence of an apparatus or piece of equipment relative to the care of the patient (e.g., warm blanket, IV pump, compression hose, etc.) or a condition of such apparatus or equipment (e.g., tube leak, ventilator piston centered, chest tube patent, etc.). A specific Observation Assertion extends ObservationAssert with any additional qualifiers necessary for the specific data element. It also constrains the data code of ObservationAssert to a code representing the apparatus or equipment observed to be present.

Figure . The ClinicalAssert and the PainAssert CEMs

partial model ClinicalAssert is statement {

key code(Assertion\_KEY\_ECID);

data CD code.card(0..1) code.domain(ClinicalAssertion\_VALUESET\_ECID);

qualifier DateOfOnset dateOfOnset card(0..1);

qualifier Aggregate aggregate card(0..1);

. . .

}

model PainAssert is statement extends ClinicalAssert {

constraint CD.code.code(Pain\_ECID);

. . .

}

Figure 23 shows the ObservationAssert model and the ChestTubePatentAssert model as an example of a CEM that extends the ObservationAssert model.

partial model EventAssert is statement {

key code(Assertion\_KEY\_ECID);

data CD code.card(0..1) code.domain(EventAssertion\_VALUESET\_ECID);

qualifier DateOfOnset dateOfOnset card(0..1);

. . .

}

model MotorVehicleCrashAssert is statement extends EventAssert {

constraint CD.code.code(MotorVehicleCrash\_ECID);

qualifier MotorVehicleRestraintUsed motorVehicleRestraintUsed card(0..1);

qualifier AirbagDeployed airbagDeployed card(0..1);

}

Figure . The EventAssert and the MotorVehicleCrashAssert CEMs.

A summary of guidelines for using assertions follows:

* Don't create assertion models for any “normal” statements (e.g., “Normal heart rhythym”, “Normal breath sounds”, “Normal reflexes”, etc.) unless we have a use case/request.
* Disorders/conditions/findings that exist in a patient fall into the assertion pattern.
* Include “Assert” at the end of the model name for assertions.
* The Type code will be the condition with the suffix - Assert, eg., EdemaAssert. The Key code will be Assertion\_Key\_ECID.

Figure . The ObservationAssert and the ChestTubePatentAssert CEMs.

partial model ObservationAssert is statement {

key code(Assertion\_KEY\_ECID);

data CD code.card(0..1) code.domain(ObservationAssertion\_VALUESET\_ECID);

qualifier StartTime startTime card(0..1);

. . .

}

model ChestTubePatentAssert is statement extends ObservationAssert {

constraint CD.code.code(ChestTubePatent\_ECID);

qualifier BodyLocationPrecoord bodyLocationPrecoord card(0..1);

. . .

}

* Population of the negation indicator (“NegationInd”) in an instance of an Assertion model negates the thing being asserted, e.g., the meaning of an instance of “CoughAssert” where “NegationInd” is TRUE is “The Patient does not have a cough”. (See Negation Indicator Modifier.)

##### Guideline: Evaluation Versus Assertion

In many cases, the decision between using the evaluation template and the assertion template is intuitive and straight forward. “Urine color”, for example, is clearly best modeled as an evaluation – the property being evaluated is the color of the patient’s urine, and the value (data) of the evaluation is the set of codes representing the colors that will be observed. To model urine color as an assertion would require the creation of unnatural precoordinated concepts – the key would be “assertion”, and data would be populated by a set of codes such as “amber urine” (meaning “the patient has amber urine”), “clear urine”, etc.

However, this highlights the fact that any evaluation model may be transformed into an assertion model. (Conversely, any assertion model may be transformed into an evaluation model.) In the case of urine color, the decision is intuitive. But in other cases, the decision is less clear.

For example, “heart rhythyms” (bradycardic, tachycardic, etc.) may be modeled as multiple assertion models (bradycardia, tachycardia, etc.) or as a “heart rhythyms” evaluation model whose data is constrained to a value set (containing “bradycardic”, “tachycardic”, etc.).

The general guideline is if it’s most natural to think of the data element as a noun – as a condition or state that exists in the patient – model as an assertion or set of assertions. If the statement about the patient is most naturally thought of as a name/value pair (i.e., a noun representing the property and an adjective representing the value), such as “hair color” = (“black”, “brown”, “blonde”), then model it as an evaluation.

Another hint is if the desire is to represent an abnormal condition in the patient, use an assertion model. Evaluations are most often used to capture descriptions of a patient property where some or all of the valid values are perfectly normal in a healthy patient. For example, if the skin is being evaluated for warmth, color and moisture level the model will be SkinCharacteristicsEval.

This discussion highlights the importance of isosemantic models. (See the discussion in the section Isosemantic models.) Even if one model or set of models can be agreed upon for the storage model (e.g., assertion models for “bradycardia” and “tachycardia” instead of an evaluation model with “bradycardic” and “tachycardic” as values), inevitably there will be use cases (data entry, messaging, reporting, etc.) for the other model. An essential (as of now unfulfilled) requirement is for a mechanism of identifying isosemantic models, managing isosemantic groups, and transforming between them.

Another variation of this issue can be illustrated by the example of modeling “cause of death”.

Options for modeling “Cause of death”:

* + A “cause of death” model instance (where “cause of death” follows the “evaluation” pattern, with “data” drawn from a value set of possible causes of death) owned by a patient
  + A statement (assertion)model instance for “drowning”, “myocardial infarction”, etc., semantically linked to a patient, with a link type of “cause of death”

NEEDS COMPLETION

#### AssertionGroup

At times, patient disorders or conditions are naturally grouped together. Notable examples are breath sounds (rales, rhonchi, wheeze, etc.) and heart sounds (murmur, click, rub, etc.). It is tempting to model such groups as “evaluations” (see Evaluations section), e.g., a “breath sounds” evaluation with values “rales”, “rhonchi”, and “wheeze”, or a “heart sounds” evaluation with values “murmur”, “click”, and “rub”. In this view, “rales” would be a *description* of the patient’s breath sounds.

The other way to view such situations is that “rales”, “wheeze”, and “rhonchi” are patient conditions, essentially no different from “diabetes mellitus type 1”, “myocardial infarction”, and “anxiety”, meriting separate assertion models. In this view, “breath sounds” is not a patient property being described, but just a convenient IS-A parent, or “classifier”, or “grouper”. In this view, “rales” is not a *description* of breath sounds, but instead *is a type of* breath sound.

This latter view is favored, to the end that all patient conditions/disorders are modeled as assertions.

The fact that “rales” *is a type of* breath sound can be leveraged to allow creation of a single, compact model for “breath sounds” (instead of separate models for “rales”, “ronchi”, “wheeze”, etc.), with a value set for “data”, including “rales”, “ronchi”, “wheeze”, etc. This is possible only if the model’s structure (i.e., its valid data type and its valid qualifiers, modifiers, attributions, and items) is the same for any value in the value set[[59]](#footnote-59). If they differ for some value, that value needs its own assertion model.

The BreathSoundsAssert model is shown in Figure 24.

##### Guideline: Noun versus Noun (Statement versus Code)

Figure . The BreathSoundsAssert CEM

model BreathSoundsAssert is statement extends ClinicalAssert {

constraint CD.code.domain(BreathSounds\_VALUESET\_ECID);

qualifier MethodDevice methodDevice card(0..1);

qualifier BodyLocation bodyLocation card(0..1);

qualifier AlleviatingFactor alleviatingFactor card(0..1);

qualifier ExacerbatingFactor exacerbatingFactor card(0..1);

. . .

}

### Procedures

Procedure models are created by extending the generic “Procedure” CEM. Procedure models are used for actions taken related to the care of a patient. The actions represented are not necessarily *surgical* procedures. (It is expected that one procedure model, yet to be created, will be a “surgical procedure” model, which can be further extended or constrained and which will contain the common attributes of a surgical procedure.) The actions might be any patient care-related actions, e.g. peripheral IV placement, delivery of a warm blanket, dressing change, ambulation, patient education, etc.

It is acknowledged that attributions (see the Attributions section) also represent actions. The difference is that Procedures represent “focal” actions (actions that are the focus of entries in the patient record), whereas Attributions are subordinate actions performed relative to patient data (e.g., a documentation action or an observation action that results in or that generates the data).

An example of a procedure model is shown in Figure 25. In the top of the figure, part of the Procedure model is shown. Note that its pattern is similar to the Assertion model. (See the Assertions section.) The key is "Procedure", and the data represents the action performed. The Procedure model has minimal qualifiers, modifiers, and attributions commonly applicable to a Procedure.

At the bottom of the figure is a model representing a central line insertion. It extends the partial Procedure model with a qualifier and modifier applicable to a central line insertion.

Note that the Procedure model contains an “observed” attribution; this supports situations in which there is an observer of the procedure as well as a performer.

The negationInd is utilized if one wishes to record a procedure as not being done.

Figure . An example procedure model.

**partial** **model** Procedure **is** **statement** {

**key** **code**(Procedure\_KEY\_ECID);

**data** CD **code**.**card**(0..1) **code**.**domain**(Procedure\_VALUESET\_ECID);

**qualifier** StartTime startTime **card**(0..1);

**qualifier** EndTime endTime **card**(0..1);

. . .

**attribution** Observed observed **card**(0..1);

**attribution** Scheduled scheduled **card**(0-1);

**attribution** Performed performed **card**(0-1);

**attribution** ReportedReceived reportedReceived **card**(0..1);

**attribution** Verified verified **card**(0..1);

}

**model** CentralLineInsertionProc **is** **statement** **extends** Procedure {

**constraint** CD.**code**.**code**(CentralLineInsertion\_ECID);

**qualifier** SourceIdentifier sourceIdentifier **card**(0..1);

**modifier** Subject subject **card**(0..1);

}

#### Guideline: Procedure versus Assertion

At times the information that a clinician needs to record is the observation of the *result* of an action. For example, a nurse charts that the patient has a warm blanket or that an IV line is in place. In both cases, a procedure is implied (i.e., that someone placed the warm blanket in the room and someone inserted the IV line). It is tempting to store “patient has warm blanket” and “IV line in place” as instances of a procedure model, where the procedure has both a “performed” attribution and an “observedResult” attribution. The same model could then be used to capture 1) the act (procedure) of giving a patient a warm blanket or 2) the observation that the patient has a warm blanket. In case 1), the “performed” attribution would be used to capture who gave the patient the warm blanket, when it was given, and where it was given. The “observedResult” attribution would not be populated. In case 2), the “observedResult” attribution would be used to capture who observed the result of the action (i.e., that the patient now has a warm blanket), when it was observed, and where the observation was made. The “performed” attribution would not be populated, because who gave the blanket, when it was given, and where it was given is unknown or can only be assumed.

This has the advantage of bringing together all mentions of a “warm blanket” – both documentation that the blanket was given and that the blanket is in place – under one model, potentially simplifying queries. However, it is awkward and unnatural to think of “patient has warm blanket” as a warm blanket placement procedure where the performer and placement time are unknown. Consequently, this pattern has been rejected. “Patient has warm blanket”, “IV line in place”, and other such observations should be modeled as assertions.

##### Guideline: Procedure versus Measurement

In measurement models a procedure instance is implied but not necessarily stored, i.e. for a heart rate measurement, the value of the measurement is stored, but there is an assumption that someone performed the “procedure” of measuring the heart rate.

In certain circumstances, the desire might be just to document that a heart rate measurement was performed – regardless of the resulting measurement. It might be tempting, therefore, to cover both this uncommon procedure use case and the much more common measurement use case with a single model – some sort of hybrid of a Procedure model and a Measurement model, with both an Observed attribution and a Performed attribution. This would have the advantage of allowing a single query to find all instances of heart rate measurement procedures and heart rate measurement results.

However, given that the result would be a rather awkward combination, and that the procedure use case is uncommon, the recommendation is to model measurements using the Measurement pattern (e.g., HeartRateMeas) and (if the use case arises) model the procedure using a Procedure model (e.g., HeartRateMeasurementProc).

NEEDS COMPLETION: If a group of similar related procedures all have the same structure they can be created with a Value Set as the data for the procedure. An example of this is TemperatureControlProc which has as its data a list of procedures from the Temperature Control Procedure Value Set. This includes things like WarmBlanketApplicationProc, IcePackApplicationProc, etc.

Also the qualifier for Patient Response with items like tolerated well, didn't tolerate well, is utilized in the procedure template.

## Components

A component, i.e., a CEM that inherits from the “component” reference class via an “is component” declaration, is a data element that cannot stand on its own independent of a statement or association. An instance of a component is only meaningful within the context of a statement or association.

As is the case for statements, a component’s “value” is composed of either a single “data” component[[60]](#footnote-60) or one or more “items” (where each item references a component CEM[[61]](#footnote-61)). A component with a single data is called a “simple component” while a component with items is called a “compound component”.

Also as is the case for statements, a component may not contain both a data component and an item or items[[62]](#footnote-62). Similar to statements, a component may contain qualifiers (which are references to other component CEMs) or modifiers (references to modifier CEMs[[63]](#footnote-63)).

Although the CDL language does not prohibit a component from containing attributions, a modeler should not add an attribution to a component. The rationale is that it is assumed that a component will simply “inherit” the attribution information of its containing statement or panel.

The CDL elements that make up a component are the same as those in statement and are as follows:

* The model name and reference class statement
* A key element (required unless the CEM is extending/restricting another CEM in which the key is stated
* data or items (required unless the CEM is extending/restricting another CEM in which the data or items are stated)
* qualifiers, modifiers, and attributions
* constraint elements

These CDL elements are summarized in Table 1.

### Components Used as Qualifiers

A component model can be used as a qualifier in any CEM model, or as an item in a compound statement or compound component. A component used as a qualifier provides more information about the value of the instance, as opposed to determining the meaning of the instance. It does not fundamentally change the semantics of the instance. For example, the component “BodyLocation” used as a qualifier within the “HeartRateMeas” model gives more information about the heart rate measurement (namely, the location from which the measurement was taken). Figure 25 illustrates the MethodDevice and BodyLocation components being used as qualifiers within the HeartRateMeas model.

Other examples include:

* Qualifying an assertion of a condition with the condition’s severity
* Qualifying an assertion of a condition with the condition’s date of resolution
* Qualifying a measurement with an indication of its change since a previous measurement

Figure . Use of qualifiers.

For a discussion of large value sets used in component models see the “Large Value Sets in Component Models” section.

model HeartRateMeas is statement {

key code(HeartRate\_KEY\_ECID);

data PQ;

qualifier MethodDevice methodDevice card(0..1);

qualifier BodyLocation bodyLocation card(0..1);

. . .

}

### Components Used as Items

A component used as an item contributes to the value of the model. For example, in Figure 26, AllergyIntolerance is a compound statement[[64]](#footnote-64) whose “value” is composed of the items Substance, SubstanceCategory, ReactionToSubstance, and Ingredient. None of the items stands on its own – collectively they constitute the Allergy/Intolerance. Consequently, each is defined (in its separate definition) as a “component”.

Similarly, Strength is an example of a compound component. Strength itself does not stand on its own, but is used inside other CEMs (e.g., “activeIngredient”, which in turn is used inside other CEMs). Its “value” is composed of the items DiscreteStrength, DiscreteVolume, AlternateDiscreteStrength, DiscreteTime, and AggregateStrength, each of which is defined as a component.

Figure . Compound statements.

model AllergyIntolerance extends PrescribingGuidance {

key code(AllergyIntolerance\_KEY\_ECID);

item Substance substance card(1);

item SubstanceCategory substanceCategory card(1);

item ReactionToSubstance reactionToSubstance card(0-M);

item Ingredient ingredient card(0-M);

. . .

}

model Strength is component {

key code(Strength\_KEY\_ECID);

item DiscreteStrength discreteStrength card(0..1);

item DiscreteVolume discreteVolume card(0..1);

item AlternateDiscreteStrength alternateDiscreteStrength card(0..1);

item DiscreteTime discreteTime card(0..1);

item AggregateStrength aggregateStrength card(0..1);

}

### Guideline: Creating a New Component CEM versus Reusing an Existing CEM

At times, when confronted with the decision of whether to reuse an existing component CEM or create a new CEM, the efficiency of reuse competes with the objective of clear semantics. The general guiding principle is that a CEM represents a distinct semantic concept. A new component model needs to be created if a new semantic concept needs to be represented.

One should *not* create a new CEM if all that is needed is one of the following:

1. Constraint (sub-setting) of the value set of the CEM’s data (e.g., BodyLocation’s data is further constrained when used in HeartRateMeas).
2. Further constraint of properties of the CEM’s data (e.g., when a component CEM is used within a statement, ST.max is decreased, ST.min is increased, PQ.value.max is decreased, etc. beyond what is declared in the component CEM) .
3. Constraint of one of the above (a value set or property) at any level within a component CEM, for example, when a component CEM is used within a statement and the data of a qualifier within a qualifier within the component CEM needs to be constrained.

The rationale for not creating a new model in these situations is that constraining of the various properties of the model do not change the “meaning” represented by the model.

When the semantics of a component model is common and uniform across its use in multiple models, it allows software to leverage this fact and provide access to the component regardless of the model. For example, if a “status” component is present in both order models and result/fulfillment models, software can access that component by name for both types of models. If the modeler added a new component (“resultStatus”) to the Result model, instead of reusing the existing “status” model that the Order model used, this polymorphism would not be possible.

On the other hand, you *do* create a new component CEM when the new use case requires any other difference besides those listed above. For example, if a CEM *almost* suits a new use case, except:

1. the new use case would require the existing CEM to have an additional qualifier/modifier/item/attribution that would not be applicable across all other uses of the CEM
2. in the new use case, certain qualifiers/modifiers/items/attributions of the existing CEM would not apply
3. the new use case would require the existing CEM to have a different data type
4. the new use case would require the existing CEM to have a different unit of measure
5. the new use case would require the existing CEM to have a broadening of a property (a greater st.max, a greater PQ.value.max, etc.)
6. the new use case would require a broadening of cardinality on any of the existing CEM’s qualifier/modifiers/items/attributions
7. the new use case would require any of the above at any level of nesting within the existing CEM
8. the name of the existing CEM would be at all ambiguous in the new use case. For example, we favor creating multiple variations of “start time” (onsetDate, effectiveTime, etc.) to clearly express the semantics of the time when any ambiguity might exist otherwise.

### Guideline: Modeling as a component or a statement

The assumption is that a component model cannot be used to store independent patient data instances. Its instances can only be stored within another model; the component model is dependent, and the containing model provides the context that gives meaning to the component instance.

Given this, it is sometimes difficult to determine whether a given clinical concept should be modeled as a component or a statement. Sometimes, the decision is clear. “Body location”, for instance, is clearly a component model. (An instance of “body location” stored alone would spawn the question, “body location of what?”) “Heart rate” is clearly a piece of information clinicians will want to store independently.

However, for a myriad of clinical elements the decision is not as clear. For example, is “ventilator FiO2 setting” a fact that will be stored independently, or just qualifying information provided in the context of a ventilator check? Is “tube patency” a component of a tube check model or a fact that will be recorded independently?

The guideline is to tend toward creating statements. “Ventilator FiO2 setting” may possibly be charted independently. And if it is needed as part of a ventilator check model, the ventilator can be modeled as a panel of independent statements. Likewise, “tube patency” may possibly be charted independently, and a “tube check” containing a “tube patency” assessment may be modeled as a panel.

This highlights, however, the need for an isosemantic modeling strategy. (See the Isosemantic models section.) A “tube patency” statement is just a model isosemantic with a “tube check” model with a component item of “patency”.

One exception to the guideline exists in the case of features of a finding. For example, if a model for “pain quality” is desired, it should be modeled as a qualifier or component item within a “pain assertion” model. The rationale is that creating a “pain quality” model implies that the patient has “pain” without explicitly stating it. The preference is to explicitly state conditions/disorders/findings, and state features of the finding as components of that statement.

## Modifiers

A modifier, i.e., a CEM that inherits from the “modifier” reference class via an “is modifier” declaration, is a structure which, when populated, significantly changes the interpretation of the value choice of the containing component or statement. For example, the Subject modifier indicates that the subject of a model instance is someone other than the patient of record. The Negation modifier indicates that the statement being made by the model instance is being explicitly negated.

A modifier is structurally identical to a component. As is the case for components (and statements, also, for that matter), a modifier’s “value” is composed of either a single “data” component[[65]](#footnote-65) or one or more “items” (where each item references a component CEM[[66]](#footnote-66)). Also as is the case for components (and statements), a modifier may not contain both a data component and an item or items[[67]](#footnote-67). Presently, all modifier are “simple”, having “data” instead of “items”.

Like statements and components, a modifier may contain qualifiers (references to component CEMs). While the CDL language permits a modifier model to be modified by another modifier, no use case is envisioned. Also, even though the CDL language does not prohibit a modifier from containing attributions, a modeler should not add an attribution to a modifier. The rationale is that it is assumed that a modifier will simply “inherit” the attribution information of its containing statement or panel.

Semantically, a modifier, like a component, cannot stand on its own, independent of a statement or association; an instance of a modifier is only meaningful within the context of a statement or association.

As noted, the difference between a modifier and component is semantic, rather than structural. The ramification of modifiers is that an implementation of CEMs needs to support queries that filter out any instances in which a modifier is populated, since the common query use case will be for instances with no modifiers (instances that pertain to the patient of record, instances that are not negated, etc.) and ignore instances with modifiers. Making Subject and Negation modifiers instead of components highlights their significance in interpreting the instance and allows an implementation to distinguish them from components.

## Attributions

The Attribution CEM defines a reusable structure for representing an action related to the CEM within which the Attribution is contained. The Attribution CEM contains the who, when, where, why, and how information related to the action. For example, “Observed” is a commonly-used attribution. When Observed is used in the heart rate measurement model, for example, it represents information related to the act of observing, which resulted in the heart rate measurement instance.

The semantics of an attribution is similar to a qualifier in that an attribution is never instantiated by itself but as an element of another model.

This section describes the Attribution structure and common Attributions (i.e., CEMs that extend or restrict Attribution) so the modeler will:

1. understand what existing Attributions are available and how to use them
2. understand Attributions such that if she needs to create a new Attribution (i.e., a new CEM that extends or restricts Attribution), she can do so.

### Attribution Structure

The elements of the attribution structure, shown in Figure 27, are described below. One of Attribution’s qualifiers -- Participant – itself contains qualifiers, so Participant’s structure is also shown in Figure 16. The IndividualPerson, IndividualOrganization, and IndividualApplication qualifiers within Participant themselves have items, so those CEMs are also shown. With the exception of PersonName in IndividualPerson, all other qualifiers/items (in Attribution, Participant, IndividualPerson, IndividualOrganization, and IndividualApplication) are simple key-data structures and hence are not shown in Figure 16. PersonName contains components for first name, last name, etc., but for simplicity, the PersonName model is not shown individually.

The Attribution’s start time (and end time, if the action is not a point-in-time action) and participant (more specifically, the individual person within the Participant) are the most important and commonly populated elements of the Attribution.

#### Key

The key code for Attributions is always Action\_KEY\_ECID. The particular action being specified in an attribution is identified in the model’s data.

#### Data

“Data”, the “value” of the Attribution, is a code that expresses the action being represented. In Attribution, data is constrained to the value set of possible values. In a particular specialization of Attribution, data is set to a specific code in the value set. For example, in Observed, data is constrained to the ECID meaning “observed”.

#### StartTime

“StartTime” is a timestamp indicating the start of the action. This is an important, commonly captured element in the Attribution.

#### EndTime

“EndTime” is a timestamp indicating the end of the action. If an Attribution instance represents an action that takes place at a point in time, it is an implementation decision whether EndTime should be left unpopulated, or whether it should be populated with the same value as StartTime. The recommendation, though, is the latter. This makes it explicit that the action occurred at a point in time. (An unpopulated EndTime might mean that the action occurred at a single point in time, or that the action is still ongoing.)

#### Participant

Figure . The Attribution CEM

**partial** **model** Attribution **is** **attribution** {

**key** **code**(Action\_KEY\_ECID);

**data** CD **code**.**required** **code**.**domain**(Attribution\_VALUESET\_ECID);

**qualifier** StartTime startTime **card**(0..1);

**qualifier** EndTime endTime **card**(0..1);

**qualifier** Participant participant **card**(0-M);

**qualifier** PatientLocation patientLocation **card**(0..1);

**qualifier** ProviderLocation providerLocation **card**(0..1);

**qualifier** Reason reason **card**(0-M);

**qualifier** ActionMethod actionMethod **card**(0..1);

}

**model** Participant **is** **component** {

**key** **code**(Participant\_KEY\_ECID);

**data** CD **code**.**card**(0..1) **code**.**domain**(ParticipationType\_VALUESET\_ECID);

**qualifier** Role role **card**(0-M);

**qualifier** IndividualPerson individualPerson **card**(0-1);

**qualifier** IndividualApplication individualApplication **card** (0-1);

**qualifier** IndividualOrganization individualOrganization **card** (0-1);

}

**model** IndividualPerson **is** **component** {

**key** **code**(IndividualPerson\_KEY\_ECID);

**item** PersonIdentifier personIdentifier **card**(0-M);

**item** PersonName personName **card**(0-M);

}

**model** IndividualApplication **is** **component**{

**key** **code**(IndividualApplication\_KEY\_ECID);

**item** ApplicationId applicationId **card**(0-1);

**item** ApplicationName applicationName **card**(0-1);

}

**model** IndividualOrganization **is** **component**{

**key** **code**(IndividualOrganization\_KEY\_ECID);

**item** OrganizationId organizationId **card**(0-1);

**item** OrganizationName organizationName **card**(0-1);

}

“Participant” is a person, application, or organization that takes some part in (i.e., participates in) the action. The structure of Participant is shown in Figure 27. Participant.data and Participant.role provide more information about the nature of the participation. Other qualifiers within Participant reflect the identity of the participant.

**Data** – NEEDS COMPLETION the kind of participant, i.e., a code indicating the nature of the participant’s involvement in the action. Participant.data values are drawn from the ParticipationType Valueset. The most common type is “performer”. For example, in an Observed attribution, a Participant type of “Performer”, indicates the “performer of the act of observing”, i.e., the “observer”. If some type of participation different from “performer of the indicated action” needs to be conveyed, other types are available, e.g., Admitter, Attender, Consultant, etc.[[68]](#footnote-68) New Attributions may require new participation types. A given individual may engage in multiple types of participations – e.g., one may be the performer of an observation, the performer of a documentation, the admitter in a registration, etc. The Participant.data is scoped by the Attribution action. For example, “Ed is the performer of the observation action, i.e., *within the context of* the observation action. Furthermore, Participant.data only has meaning within the context of the action. After completion of the action, “performer” has no meaning. (See in contrast the description of “role” below.)

Participant.data is roughly equivalent to Participation.typeCode in the HL7 v3 RIM.

**Role** - the role of the participant in the action. For example, “physician”, “patient”, “spouse”, etc. As is the case for Participant.data, the participant’s role is scoped by the action represented by the Attribution. For example, “Ed is a nurse *in the context of* this observation action”. However, unlike Participant.data, a Participant’s role tends to persist outside the temporal bounds of the Attribution action – Ed was a nurse before his observation action and will continue to be a nurse after the observation action is completed. In contrast, Participant.data only has meaning during the Attribution action.

Role values are more permanent that Participation types – they are clinical credentials, familial relationships, etc.

A particular individual may play different roles in different actions.

Role will not commonly be captured in most Attribution instances.

Role.data is roughly equivalent to Role.class and Role.code in the HL7 v3 RIM.

**IndividualPerson** - identification of a person acting as a Participant in the action. Figure 14 shows that IndividualPerson contains the name of the person who is the participant in this action and an identifier for the person. From an EMR perspective, this is an important element in most Attributions. The EMR needs to capture the identity of the clinician performing an action.

**IndividualApplication** - identification of an application acting as a Participant in the action. Figure 16 shows that IndividualApplication contains the name of the application that is the participant in this action and an identifier for the application. This is a less commonly populated element in the Attribution since most Participants are individuals, not applications.

**IndividualOrganization** - identification of an organization acting as the Participant in the action. Figure 14 shows that IndividualOrganization contains the name of the organization that is the participant in this action and an identifier for the organization. This is a less commonly populated element in the Attribution since most Participants are individuals, not organizations.

#### PatientLocation

“PatientLocation” represents the location of the patient when the action took place[[69]](#footnote-69). This is a less commonly populated element in the Attribution.

#### ProviderLocation

“ProviderLocation” represents the location of the provider when the action took place[[70]](#footnote-70). This is a less commonly populated element in the Attribution.

#### Reason

“Reason” represents the (coded) purpose for performing the action. In most Attributions, e.g., “Observed”, the reason will not be an important, commonly populated element.

## Associations

CDL supports a reference class called “PatientAssociation”, which is an association of patient-owned data instances. The most commonly-used specializations of PatientAssociation are Panels, Semantic Links, and annotations. Less commonly used are lists and collections. This discussion will just address panels and semantic links and annotations.

The structure of one of these CEMs is much the same as that of a statement or component, with the following notable differences:

* In a panel and semantic link, only items are permitted – not “data”. The items represent pointers[[71]](#footnote-71) to independent instances. The panel or semantic link represents an association between these items. (A model that extends or restricts one of these association models may just “inherit” the parent’s items, and not actually list an item in its definition.)
* In contrast, an annotation refers to both “data” and one or more items. The annotation is the only CEM in which this is allowed. Annotations are used for “comments” -- an annotation represents an association between its “data” (the comment) and the item or items.
* Panels, semantic links, and annotations permit an additional constraint on their items – that of specifying/constraining the item’s “role” in the association. The syntax is as follows:

**item** SomeModel SomeLabel **card(**<cardinality>) role.**code**(SomeRole\_ECID);

where “SomeRole\_ECID” is a code representing the role this item plays in the association.

The CDL elements pertinent to panel, semantic link, and annotation CEM are shown in Table 2 and Panels, Semantic Links, and Annotations will be described in further detail.

Table . CDL Elements used in CEMs where the reference class is a child of PatientAssociation (semanticLink, panel, collection, list, annotation)

|  |  |  |
| --- | --- | --- |
| Element | Explanation | Required? |
| **model** SomeModel **is** **SomeReferenceClass** | Constrains the type to be the code represented by the designation “SomeModel”; constrains the reference class to be “SomeReferenceClass” where “SomeReferenceClass” is a child of PatientAssociation (semanticLink, panel, collection, list). (See the section.) | This is required in every model.  Optionally, the line may begin with “partial” if this model is to be used as the parent of another model. (See the section.)  Optionally, the line may end with an “extends” or “restricts” statement. (See the section.) |
| **key** **code**(SomeECID\_KEY\_ECID)  OR  **key** **domain**(SomeSet\_KEY\_VALUESET\_ECID) | Constrains the key to be (in the first line) the code represented by the designation “SomeECID\_KEY\_ECID” or (in the second line) a code drawn from the value set represented by the designation “SomeSet\_KEY\_VALUESET\_ECID”. | One or the other of these two statements is required in every model unless the model is extending/restricting another model (see the section).  If the model is extending/restricting another model, a key constraint is optional (used if the key of the parent model needs to be restricted). |
| **data** <ST or CD> | Constrains the CEM to have data (instead of items) and constrains the data to be of a certain data type. | The only association model that is allowed to have “data” is an Annotation (i.e., comment). An Annotation will always have a “data” statement, unless it is extending/constraining another Annotation where “data” was already declared, and it does not need to further constrain the original “data”.  An Annotation is the only CEM that will have both data and an item or items. |

|  |  |  |
| --- | --- | --- |
| Statement | Explanation | Required? |
| **item** SomeCEM someLabel **card**(<cardinality>) **role**.code(CommentsTarget\_ECID) | An item statement constrains (or multiple item statements collectively constrain) the universe of available items to just those items valid for this CEM.  An item in a PatientAssociation, unlike in a statement or component, may specify a “role” constraint, indicating the role the item plays in the association. | If the model is restricting another model, item is not permitted.  If the model is extending another model, item is permitted. |
| **qualifier** SomeCEM someLabel **card**(<cardinality>) | Specifies a valid qualifier for this CEM, and by so doing constrains the universe of qualifiers. | Optional |
| **modifier** SomeCEM someLabel **card**(<cardinality>) | Specifies a valid modifier for this CEM, and by so doing constrains the universe of modifiers. | Optional |
| **attribution** SomeCEM someLabel **card**(<cardinality>) | Specifies a valid attribution for this CEM, and by so doing constrains the universe of attributions. | Optional |

|  |  |  |
| --- | --- | --- |
| Statement | Explanation | Required? |
| **constraint** <object to be constrained> <constraint> | Specifies an object to be constrained and the manner in which it is to be constrained. See accompanying text for further discussion.  Used most commonly for:   * further constraining a value set originally constrained in another CEM * further constraining a cardinality originally constrained in another CEM | Optional, appears as additional constraint is needed |

### Panels

A panel is a CEM that is a list of individual CEM statement models. The individual CEMs are listed as items. A panel is a specialization of the Association reference class.

A panel is used in cases where the items assume the same provenance and attribution, are assumed to be mutually dependent, and are created as part of a single event.

Each model referenced as an item in a panel must be a subclass of statement or panel[[72]](#footnote-72) (i.e., “is statement” or “is panel”), not component.

A panel CEM can represent what is known in clinical practice as a battery or panel.

Example:

* A Basic Metabolic Panel is an example of a common lab panel. A BMP contains statements representing glucose, sodium, chloride, and other lab measurement CEMs, which could each exist independently of the enclosing panel.
* Clinical scoring concepts can be portrayed using panels. For example the APGAR score used for scoring the clinical state of infants at birth consists of measures of activity, appearance, grimace, pulse, respiration and then the total score. Since the associated score with each of the elements is important to know as well as the total score, a panel model can capture these elements well.

#### Propagation

Propagation of the qualifiers, modifiers, and attributions referenced by a panel to the panel’s individual items deserves separate discussion. To illustrate the principle, suppose a BloodPressure panel contains a “BodyLocation” qualifier (meaning the body location from which the blood pressure measurement was taken), a SystolicBloodPressure item and a DiastolicBloodPressure item. In an instance of the panel, the BodyLocation is interpreted as applying both to the panel instance as a whole and to each individual item (the systolic blood pressure and the diastolic blood pressure) in the panel. (More specifically, BodyLocation applies to the “data” of each individual item in the panel, i.e., BodyLocation provides more information or context with which to interpret the data of this measurement.)

Similarly, suppose a model for a lab result panel contains a “SpecimenCollected” Attribution (which specifies who collected the specimen that was analyzed to produce the result, when it was collected, where it was collected, etc.), a “Potassium” item, and a “Sodium” item. In an instance of this panel, its SpecimenCollected instance is interpreted as applying to the panel as a whole, but also to the individual Potassium instance (or, more exactly, its “data”) and the Sodium instance (or its “data”).

PanelA

Qual1

Qual2

Mod1

Attr1

Item1

Item2

Suppose the PanelA model contains Qualifier 1, Qualifier 2, Modifier 1, Attribution 1, Item 1, and Item 2, as shown. Suppose further that an instance of the PanelA model is populated as shown.

PanelA

Qual1

Data = A

Qual3

Data = R

Qual2

Data = B

Mod1

Data = P

Attr1

StartTime = Q

Participant

Data = Z

Item1

Data = X

Item2

Data = Y

Propagation that the instance is interpreted as if

PanelA

Qual1

Data = A

Qual3

Data = R

Qual2

Data = B

Mod1

Data = P

Attr1

StartTime = Q

Participant

Data = Z

Item1

Data = X

Qual1

Data = A

Qual3

Data = R

Qual2

Data = B

Mod1

Data = P

Attr1

StartTime = Q

Participant

Data = Z

Item2

Data = Y

Qual1

Data = A

Qual3

Data = R

Qual2

Data = B

Mod1

Data = P

Attr1

StartTime = Q

Participant

Data = Z

In other words, populating qualifier 2 with “B” at the panel level is in a sense a “shorthand” for saying Item 1 and Item 2 both contain qualifier 2 with value “B”. Qualifier 2 “propagates” to Item1 and Item2 of the panel.

For the remainder of this discussion, “propagates” means “is interpreted as if it were contained by each item in the panel”.

Notes:

* Panel qualifiers, modifiers, and attributions propagate to each of the panel’s items.
* If a panel contains another panel, the upper panel’s qualifiers, modifiers, and attributions propagate to the contained panel and then propagate to each item in the contained panel.
* Panel qualifiers, modifiers, and attributions do not propagate to qualifiers, modifiers, and attributions contained in the items. For example, suppose the following:

Panel1

Qualifier1

Item1

Qualifier2

Attribution1

This is interpreted as:

Panel1

Qualifier1

Item1

Qualifier1

Qualifier2

Attribution1

But not as:

Panel1

Qualifier1

Item1

Qualifier1

Qualifier2

Qualifier1

Attribution1

Qualifier1

Qualifier1 does not “propagate” to Qualifier2 or Attribution1. By definition, propagation only applies to contained items (statements).

* Clearly, two different implementation options exist:
  + Upon storage, instances of panel qualifiers/modifiers/attributions are copied down to the panel’s contained item statements upon storage.
  + The panel’s qualifiers/modifiers/attributions are not stored with each contained item statement, but upon retrieval of a contained item statement instance, they are added to the retrieved item statement instance.
* For a qualifier/modifier/attribution of a panel to propagate to a contained item, the qualifier/modifier/attribution must be valid in the definition of the contained item. This can be checked for at compile time.
* In a panel instance before propagation, an instance of the same qualifier/modifier/attribution is not permitted to exist both at the panel level and at the contained statement level. This eliminates the need to decide which qualifier/modifier (the one at the panel or the one at the statement) takes precedence.
* Panels are not permitted to have a negation modifier, hence propagation of negation from panel instance to contained statement instance is a moot topic.
* The same rules apply for propagation of a panel’s qualifiers/modifiers/attributions to a contained panel, namely:
  + A containing panel instance's qualifiers, modifiers, and attributions propagate to each of its contained panel instances.
  + A qualifier/modifier/attribution of a panel instance cannot propagate to a contained panel instance if that panel’s model does not permit that qualifier/modifier/attribution.
  + In a panel instance, an instance of the same qualifier/modifier/attribution is not permitted to exist both at the containing panel level and at the contained panel level.
  + Panels are not permitted to contain the negation modifier, hence propagation of negation from containing panel instance to contained panel instance is a moot topic.

### CDL support for propagation (conduction)

CDL supports making these propagations or “conductions” explicit via the “conduct” keyword. An example is shown in Figure 28.

*EXAMPLE*

**model** SomePanel **is** panel **extends** SomeBaseModel

{

**qualifier** SomeQualifierType label1;

**qualifier** SomeQualifierType label2;

**item** SomeStatementType label3

label3.somelabel1.**conduct**(label1)

label3.somelabel2.**conduct**(label2);

}

Figure . Conduction

In this CDL example, SomePanelModel has qualifiers “label1” and “label2” at the panel level. The statement

**item** SomeStatementType label3

label3.somelabel1.**conduct**(label1)

label3.somelabel2.**conduct**(label2);

does two things:

1. It declares SomePanelModel to contain an item of type SomeStatementType (identified by label3).
2. It declares that SomePanel’s qualifiers label1 and label2 should get “conducted” to label3’s somelabel1 and somelabel2 qualifiers, respectively. In other words, SomePanel’s label1 qualifier maps semantically to label3’s label1 qualifier, and hence label1’s value should propagate to label3.somelabel1. Similarly, SomePanel’s label2 qualifier maps semantically to SomeStatementType’s label2 qualifier and hence label2’s value should propagate to label3.somelabel2. The requirement is that within the definition of SomeStatementType, there are qualifiers identified by somelabel1 and somelabel2, and that they have the same datatype as label1 and label2, respectively.

#### Guidelines for Use

* Panels are well suited for groups of observations that are routinely taken together. For example:
  + Ventilator checks (including ventilator type, ventilator mode, pressure settings, etc.)
  + Specimen descriptions (including size, color, etc.)
  + Fluid output observations (including fluid amount, fluid description, etc.)
* All items in a panel must be defined (i.e., in their own CEM definitions) as statements or panels (i.e., they must inherit from the “statement” or “panel” reference classes via the declaration “is statement” or “is panel”)[[73]](#footnote-73).
* Panel qualifiers, modifiers, and attributions are assumed to propagate down to the panel’s items. In CDL, an author can make propagation explicit, via the “propagate” statement.
* A panel can extend or restrict a parent CEM.
* Use CDL’s “conduct” statement to implement propagation.
* Make sure that important contextual information exists not just at the panel level but at the individual item/statement level also. Remember that individual items may be retrieved independent of the panel.

### Semantic Links

A semantic link is an association between instances of CEMs. As an example, a semantic link could associate a Throat Culture instance that was positive for Streptococcus pyogenes with the resulting Order for Penicillin. The type and key of the semantic link define the type of relationship, e.g., a semantic link in which the source instance is the cause of the target instance might have type = “CausesLink” and key = “Causes”.

A semantic link also allows the modeler to specify the “roles” of the instances being associated. In the case of the “CausesLink” example, one instance might be defined as playing the role of “cause” and the other instance might be defined as playing the role of “effect” in the association.

is an example of a semantic link that associates two health issues – one that subsumes the other[[74]](#footnote-74). Note that the link is defined with “is semanticLink” i.e., the CEM specializes the semanticLink reference class. The semanticLink reference class in turn is a specialization of the patientAssociation reference class, which represents an association of patient-owned instances. (Panel, explained in the Panels section, also specializes this class.)

Each instance in a semantic link assumes a role (either the subsuming health issue or the subsumed health issue) in the association. This link might be used, for example, if a clinician charted a health issue of “fever” for a patient, and then later charts a health issue of “sepsis” for that patient, and deems that the sepsis issue *subsumes* the fever issue.

Other examples of semantically-linked instances might include:

* Link the following health issues instances:
  + First health issue “Fall from roof”
  + After assessment “Broken Femur”
  + After more care “Mobility impaired due to external fixator”
* Link a medication order instance with a problem instance in the patient’s problem list that is the reason for the order.
* Link a surgery instance with a problem instance that was a complication of the surgery.

Semantic links are most often made between two instances, although the structure supports association of any number of instances. (Subsequent discussion will assume the common use case of a semantic link associating only two instances.)

Semantic links are used to make “loose” associations. In the subsumes example, one health issue instance does not own the other and one is not thought to contain the other. If one instance is deleted, it has no effect on the other.

A semantic link is not necessarily created at the same time as the instances it is associating. The link *may* be made at the same time, but it may be made long after either instance is created. And since it is a loose association, the link may be made without updating either instance.

As is the case for any CEM, a semantic link model may contain qualifiers, modifiers, and attributions. In this sense, a semantic link is like an “association class” in UML, which allows an association between two classes to have its own attributes. The example of Figure 16, for example, contains a “documented” attribution.

An outstanding issue is we need to be able to declare an associated instance in a semantic link as being able to be any one of a group or domain of CEM types, e.g., the “complication” semantic type can have as its source instance an instance of either the SurgicalProcedure CEM or the NursingIntervention CEM.

Figure . Semantic Link example

**model** HealthIssueSubsumptionLink **is** **semanticLink**{

**key** **code**(HealthIssueSubsumptionLink\_KEY\_ECID);

**item** HealthIssue parentHealthIssue **card**(1) role.**code**(Subsuming\_ECID);

**item** HealthIssue childHealthIssue **card**(1..M) role.**code**(Subsumed\_ECID);

**attribution** Documented documented **card**(0..1);

}

### Qualifiers versus Panels versus Semantic Links

There are three common ways to represent associations between CEM instances. The first way is to use a qualifier. For example, an “Attender” qualifier in an “Encounter” CEM can contain a “pointer” to a provider. This is illustrated in Figure 31. Figure 31 shows a PatientEncounter CEM that contains an “attender” item. The Attender CEM is just a pointer to a Provider instance. The figure shows two alternate solutions for the Attender. In the first, the pointer is an internal identifier (an IREF data type) where the IREF is constrained to point to a model type of “Provider”. In the second, the pointer is an external identifier (an II data type), where the pointer’s “type” is constrained to the set of codes that are Provider id types.

The second way to represent an association is via a Panel. As explained in the Panels section, a Panel is an association in which a shared context is assumed between the associated instances – a panel is stored as a whole, thus its associated instances share the same attribution information.

Figure . Illustration of a pointer

**model** PatientEncounter **is** **encounter**{

**key** **code** (PatientEncounter\_KEY\_ECID);

**item** PatientEncounterType patientEncounterType **card**(0..1);

**item** StartTime startTime **card**(0..1);

**item** EndTime endTime **card**(0..1);

. . .

**item** Attender attender **card** (0..M);

. . .

}

**model** Attender **is** **component** {

**key** **code**(Attender\_KEY\_ECID);

**data** IREF **type**(Provider);

}

**model** Attender **is** **component** {

**key** **code**(Attender\_KEY\_ECID);

**data** II;

**constraint** II.type.**domain**(ProviderIdType\_VALUESET\_ECID);

}

The third way to represent an association is via a semantic link. As explained in the “Semantic Links” section, a semantic link is a loose association of CEM instances where the association has a “type” and the instances have “roles”.

Consequently, use a qualifier approach for “strong” associations, i.e., when one CEM is considered to be an attribute of another CEM. The “pointed-to”, or referenced CEM qualifies, or gives more information about the pointing CEM. Updating the referenced CEM necessarily updates the containing CEM.

Use a panel when you need to represent a structure that contains a group of independent instances, where the containing structure and instances will be stored at the same time[[75]](#footnote-75) and the instances will share context (e.g., the same observer, the same observed time, etc.).

The semantic link approach is used for looser associations. The semantic link instance is independent from the source and target instances; it just contains pointers to those instances. When the association needs to be created independent of the instances (i.e., one or both of the instances already exist and the link will be made afterwards) without updating the instances, use a semantic link. A semantic link is appropriate when the association is not generally applicable to all instances of the respective classes; e.g., a semantic link can represent an association between a surgical procedure instance and a complication, but all surgical procedure instances are not associated with a complication instance.

Factors to consider when making the choice:

* Semantic links are by their nature bi-directional.
* Qualifiers don’t really work well for joining two existing instances.
* Semantic links don’t require a change to stored data instances.
* A semantic link should never change the meaning of the source or target instance, thus it should always be safe to ignore semantic links.

NEEDS COMPLETION:

*There are two types of semantic links. One type is a link between two instances where at least one of the instances pre-existed. e.g., create a discharge dx link between this dx and an already created encounter, create a supported by link between this new dx and an already existing observation, create a “fulfills” link between this new procedure and an already-existing order.*

*On the other hand, there are other links that are between instances that are created at the same time, e.g., create an “associated” link between chest pain and shortness of breath, create a “complication” link between a surgery and an assertion. It’s likely that people will commonly want this latter type of link returned with the source statement, like qualifiers – i.e., the common query should return a statement’s qualifiers as well as all links of this 2nd type. In contrast, it isn’t expected that the first type of link would always be retrieved with the statement instance. So should type 2 just be modeled as a qualifier?*

*If both types are modeled as semantic links, how should the “type” of link (type 1 or type 2) be indicated? We think that this is static information (a design time choice) – instances of a given semantic are always either type 1 or type 2 – some of its instances aren’t type 1 while others are type 2. So we could indicate a link type’s “type” in terminology. But it also might be a useful denormalization to indicate the link’s “type” in the model.*

## Annotations (Comments)

A comment is additional information related to a CEM instance that is supplemental information that does not affect the meaning of the value. For example, attached to a medication order or administration instance, there may be a pharmacist comment, a nurse comment, etc. Pharmacists will want to see the pharmacist comment and nurses the nurse comment.

A comment may be text or coded.

NEEDS COMPLETION (INCLUDING CDL EXAMPLE): CDL comments are still being discussed. The following reflects current thinking about comments:

* Comments play no part in the workflow. If the element under consideration does play a part in the workflow, make it a qualifier instead.
* Comments are different from qualifiers because they propagate differently from panel to item. A qualifier instance at the panel overrides an instance of the same qualifier on an item in the panel. In contrast, a comment instance at the panel level is additive to any comment of the same type at the item level.
* Comments should by default be valid in any model, but any given model should be able to constrain the comment out if they shouldn’t be permitted.
* Comments may be attached to a component instance.
* Comments can be CD or ST.

# Examples of Commonly Used CEM structures

## Components

Some commonly used components will be discussed in more detail below. (The headings say, “Qualifier”, which is the most common usage, although each component may also be used as an item in a compound statement).

### Aggregate Qualifier

The Aggregate qualifier represents the totality of a statement. It is a precoordination of all the information in the statement’s key and value choice, plus its qualifiers, modifiers, and attributions.

One use case for the Aggregate qualifier is to capture the verbatim text from which a CEM instance was normalized via natural language processing. For example, the free text before processing might have said, “childhood onset of mild, chronic asthma”, which through natural language processing was normalized to the following:

AsthmaAssert

Key = Assertion

Data = Asthma

Qualifier Chronicity

Key = Chronicity

Data = Chronic

Qualifier Severity

Key = Severity

Data = Mild

Qualifier Onset

Key = Onset

Data = Childhood

Qualifier Aggregate

Key = Aggregate

Data = “childhood onset of mild, chronic asthma”

Another use case might be to capture the precoordinated text (or precoordinated code) that a clinician enters in a problem list or an ordering application, even though the resulting storage instance breaks down the text (or code) into a more postcoordinated storage.

The Aggregate model’s data type is CD, because in certain use cases the precoordinated “meaning” of the statement is drawn from a finite, predictable set of meanings, represented as coded values in a value set. In cases where the meanings are not a manageable set of codes, the CD’s originalText property can be used to capture free text.

As an example, suppose a system submits “18 fr. Foley catheter inserted using sterile technique,” and this is normalized to an instance of a “Catheter Insertion” CEM. This string would be stored in aggregate. The CEM may have a qualifier for “catheter type”, which would be populated with “Foley”. The CEM might not have a “method” qualifier (which, if it existed, could capture “sterile technique”). That information would only be preserved in the aggregate. This illustrates that the Aggregate is not guaranteed to be a lossless representation of the CEM instance – it may contain more information than the CEM instance. In other situations, it may contain less (e.g., the CEM instance may contain attribution information not reflected in the precoordinated string). It only represents precoordinated information from which the CEM instance was generated.

### Body Location/Body Location Pre-coordinated Qualifiers

The bodyLocation and bodyLocationPrecoord qualifiers are used to indicate a part of a patient’s body pertinent to the measurement, evaluation, procedure, or assertion that is being addressed by the CEM. Their exact meanings are context-dependent. For example, Table 3 shows the context-dependent meaning of BodyLocation or BodyLocationPrecoord used within various CEMs.

Table . Meaning of body location in various models

|  |  |
| --- | --- |
| **Containing CEM** | **Meaning of BodyLocation/BodyLocationPrecoord** |
| HeartRateMeas | The body location at which the measurement was taken |
| PainAssert | The body location at which pain is felt |
| PneumaticCuffPlacementProc | The body location at which the cuff is placed |
| SkinAbrasionAssert | The body location at which the abrasion is observed |

BodyLocation and BodyLocationPrecoord differ in that BodyLocationPrecoord contains just a precoordinated data value (“left foot”, “right foot”, “left upper quadrant”, “right lower quadrant”, etc.) while BodyLocation contains the BodySide qualifier (the body side of the body location – left or right) and the BodyLaterality[[76]](#footnote-76) qualifier (any further spatial orientation characteristic of the body location – inferior, superior, distal, proximal, upper, lower, medial, lateral, etc.). In other words, BodyLocationPrecoord draws from a value set of precoordinated values, while BodyLaterality takes a postcoordinated approach, distributing the information across its data and qualifiers. The two qualifiers are shown in Figure 32.

BodyLocation is utilized when either the domain of values for the model instance would be very large or when there is a special requirement to be able to query by body laterality or body side. For instance, in the pain assertion model, attempting to formulate and maintain a list of precoordinated body locations at which pain can be felt would be an unwieldy task. Another example is the W*oundDrainOutputFluidVolumeMeas* model, in which the value set of possible body locations (locations at which a wound can exist) would be very large.

On the other hand, body location within a “breath sounds assertion” model (meaning the lung location at which the breath sounds are observed) consists of only a few possible values – even precoordinating “right”, “left”, “lower”, and “upper” in all possible combinations – so BodyLocationPrecoord would be used. Postcoordinating the side and laterality present no appreciable benefit. Similarly, for heart rate measurement body locations, capturing “wrist” with a body side qualifier of “right” or “left” provides no appreciable benefit over just capturing precoordinated values of “right wrist” and “left wrist”.

Figure . BodyLocation and BodyLocationPrecoord

model BodyLocation is component {

key code(BodyLocation\_KEY\_ECID);

data CD code.card(0..1) code.domain(BodyLocation\_VALUESET\_ECID);

qualifier Aggregate aggregate card(0..1);

qualifier BodyLaterality bodyLaterality card(0..1);

qualifier BodySide bodySide card(0..1);

}

model BodyLocationPrecoord is component {

key code(BodyLocationPrecoord\_KEY\_ECID);

data CD code.card(0..1) code.domain(BodyLocationPrecoord\_VALUESET\_ECID);

}

It should be noted that just because the values in a value set are precoordinated does not mean the “side” or “laterality” information is lost. It would be expected that the terminology concept “right wrist” stored in the terminology server would contain enough information to support the inference that “right wrist” involves the right side of the body.

### Body Position Qualifier

The bodyPosition qualifier is used to indicate the position of the subject at the time the data addressed by the model is captured. Body position is utilized when the subject’s body position can make a difference in data interpretation. For example, when a blood pressure is taken it may be important to know if the subject is standing or sitting.

### Route, RouteMethodDevice and MethodDevice Qualifiers

A ‘method’ is the process used to administer a substance or obtain a finding. A ‘route’ is the particular pathway something, (usually a substance), was administered, or obtained. A ‘device’ is the physical object used to make a measurement, deliver a substance, or evaluate a finding.

RouteMethodDevice includes not only the way something (usually a substance) was administered, but also the physical object used to make a measurement or deliver a substance. A thermometer could be used via several routes, and values can be added like "otic thermometer", "axillary thermometer", etc, rather than requiring the user or application to record a device (thermometer) and a body location separately.

MethodDevice includes the process used to administer a substance or obtain a finding, as well as the physical object used to make a measurement, deliver a substance, or evaluate a finding.

When to use MethodDevice vs. RouteMethodDevice vs. Route and why we precoordinate NEEDS COMPLETION

Method examples include: Z-Track, non-invasive blood pressure (NIBP), auscultation, percussion, palpation, and endoscopy.

Route examples include: Tracheal, intravenous, nasal, etc.

RouteMethodDevice examples include: Endotracheal tube, nasogastric tube, bronchoscope, and endoscope.

### AssociatedCondition Qualifier

The AssociatedCondition qualifier is used to capture pertinent information related to the interpretation of the model and its value. Some examples:

* “Preopererative” and “postoperative” vital signs
* “shoes on” and “shoes off” height measurements
* “postprandial”
* “post void” urine residuals
* “pre treatment” and “post treatment” Forced Expiratory Volumes
* “unclothed” and “clothed” weight measurements
* “Pre exercise” and “post exercise” vital signs

This qualifier’s value set covers a broad range of values. Each model in which it is used should constrain the broad value set to just those values pertinent to the given model. For example, when the AssociatedCondition qualifier is used in the WeightMeas CEM, a constraint statement should be added that constrains the AssociatedCondition value set to a smaller valueset (called “WeightAssociationCondition\_VALUESET\_ECID” or something of the sort) that constrains the values to just “clothed” and “unclothed”.

If when creating a new model, a modeler cannot think of values for associatedCondition, leave the qualifier out.

### Periodicity Qualifier

The Periodicity qualifier captures the nature of an event with respect to its recurrence as a coded value. For example, “continuous” and “recurrent” are values of the value set.

### Course Qualifier

The Course qualifier captures the pattern of change being undergone by the disease, disorder, finding, or observation as a coded value. For example, an instance of the FeverAssert model could indicate that the fever is “improving” or “worsening”.

### Severity Qualifier

The Severity qualifier is used to capture the seriousness of a disease, disorder, finding, or observation as a coded value. For example, an instance of the *PenetratingTraumaAssert* model might indicate a severity of “critical”.

In some assertion models, the severity qualifier is replaced with a qualifier that represents a numeric scoring system. For example, PainAssert contains a PainSeverityScore qualifier instead of a Severity qualifier.

### Alleviating Factor Qualifier

The AlleviatingFactor qualifier is utilized mainly in assertion models. It is used to record what makes a condition better, such as “resting” or “keeping weight off of”. For example, in an instance of the NauseaAssert model a value for alleviating factor may be “limiting movement”.

The AlleviatingFactor model has a large value set. The “Large Value Sets in Component Models” section discusses treatment of this and other such value sets.

### Exacerbating Factor Qualifier

The ExacerbatingFactor qualifier is analogous to the AlleviatingFactor qualifier. It is utilized mainly in assertion models. It is used to record what makes a condition worse, such as “exercise” or “bearing weight”. For example, in an instance of the NauseaAssert model, a value for exacerbating factor may be “standing”.

The ExacerbatingFactor model has a large value set. The “Large Value Sets in Component Models” section discusses treatement of this and other such value sets.

### Date of Onset Qualifier

The DateOfOnset qualifier is utilized when recording a condition that has or has had a start date. Data in the model is a time stamp (TS) data type. For example, in an instance of the PainAssert model, DateOfOnset may be 10/31/2010, to indicate that the pain was first observed on 10/31/2010.

### Date of Resolution Qualifier

The DateOfResolution qualifier is utilized when recording a condition that has gotten better or is no longer an issue. Data in the model is a time stamp (TS) data type. For example, in an instance of the PainAssert model, DateOfResolution may be 10/31/2010, to indicate that the pain was observed to no longer an issue on 10/31/2010.

### Associated Signs and Symptoms Qualifier

The associated signs and symptoms qualifier is utilized to capture conditions which are associated with a model instance[[77]](#footnote-77). For example, DiabetesMellitusTypeOneAssert could have associated signs and symptoms of “poor eyesight.” 74

The AssociatedSignsAndSymptoms qualifier has a potentially large value set. The “Large Value Sets in Component Models” section discusses treatment of this and other such value sets.

### Abnormal Interpretation Qualifier

NEEDS COMPLETION -- This qualifier is being deprecated in favor of “Interpretation”.

This model stemmed from the need to capture more than just whether or not an instance represents abnormality (via an “abnormalFlag”), but in addition to capture an “interpretation”. It is utilized mostly in measurement models where the data value is interpreted as abnormal for the situation.

For example, in the model HeartRateMeas a data value of 200 beats per minute (bpm) is recognized as an abnormal value, and could have an abnormal interpretation of above upper panic levels or above normal depending upon the subject.

### Delta Flag Qualifier

The DeltaFlag qualifier is utilized to indicate a noticeable or important change in data in the current instance of the model compared to a previous instance. It is utilized primarily in measurement models.

For example, an instance of the Potassium model may have a value of 3.4 meq/L. If the patient’s previous potassium value was 5.0 meq/L, a user or an instrument may wish to indicate that this new potassium value represents a significant change, and hence may populate the DeltaFlag qualifier with the value, “significant change down”[[78]](#footnote-78).

### ReferenceRangeNar Qualifier

The ReferenceRangeNar qualifier is utilized mainly in laboratory and measurement models to capture the reference range as stated by the lab system or measurement device. For example, for a Potassium value, a lab system may send the string, “3.5-5.0”, meaning that within its population of patients, measured by its instruments, a specified large percentage (usually 95%) of normal healthy patients’ potassium values falls within the 3.5-5.0 meq/L range. This string, as sent by the lab system when reporting a potassium value, is captured in the ReferenceRangeNar qualifier of the potassium instance.

The “Nar” in “ReferenceRangeNar” refers to the fact that the reference range value is captured as “narrative” data, i.e., a string. In contrast, ReferenceRangePQ breaks apart the reference range into a low boundary and a high boundary, each expressed as a PQ data type. Similarly, ReferenceRangeCO, ReferenceRangeIVLPQ, and ReferenceRangeRTO break apart a reference range into CO, IVLPQ, and RTO elements, respectively. (See discussion in the “ReferenceRange<data type> Qualifier” section below.)

### ReferenceRange<data type> Qualifier

The ReferenceRangeCO, ReferenceRangeIVLPQ, ReferenceRangePQ, and ReferenceRangeRTO qualifiers are used mainly in laboratory and measurement models to capture the reference range as stated by the lab system or measurement device. For example, for a Potassium value, a lab system may send the string, “3.5-5.0”, meaning that within its population of patients, measured by its instruments, a specified large percentage (usually 95%) of normal healthy patients’ potassium values falls within the 3.5-5.0 mEq/L range. This string, as sent by the lab system when reporting a potassium value, would be captured by the ReferenceRangePQ qualifier, since a potassium value is expressed in mEq/L, which is a PQ expression. (A PQ, or physical quantity, consists of a real numeric value and a code representing a unit of measure.) The string would be broken into a “low bound” (3.5 mEq/L in the example, captured in the LowerBoundPQ item within the ReferenceRangePQ qualifier) and a “high bound” (5.0 mEq/L in the example, captured in the UpperBoundPQ item within the ReferenceRangePQ qualifier).

Similarly, the ReferenceRangeCO qualifier is used when the lower and upper bounds are CO data types (e.g., a reference range such as “1+ to 3+”), the ReferenceRangeIVLPQ qualifier is used when the lower and upper bounds are each an interval of PQs (e.g., a reference range such as “3.0 – 4.0 mEq/L to 7.0 – 8.0 mEq/L”) and the ReferenceRangeRTO qualifier is used when the lower and upper bounds are ratios (e.g., a reference range such as “1:1000 to 10:1000”).

Contrast this with an approach in which the reference range is not broken into low and high bounds, but is captured as a single string, e.g., “3.0 – 5.0 mEq/L”. This string approach is taken by the ReferenceRangeNar approach (discussed in the “ReferenceRangeNar Qualifier” section). The ReferenceRange<data type> approach, which is a “post-coordinated” approach that breaks down the reference range into low and high bounds, is preferred, since it is more usable by decision support[[79]](#footnote-79). The ReferenceRangeNar approach may be used in addition to present the complete picture (i.e., capture both the string that was submitted and broken-down, atomic data).

## Attributions

Some commonly used attributions will be discussed below. Also, “Appendix B – Attribution Examples” presents examples of usage for these attributions.

### Observed

The Observed attribution captures information related to the act of observing. The common use case for including Observed in a CEM is when the CEM represents an observation -- used loosely to mean a Measurement (as described in the Measurements section), an Evaluation (as described in the “Evaluations” section), or an Assertion (as described in the “Assertions” section). In these cases, Observed captures the time of the observation and the performer of the observation (as well as the patient/provider locations and a reason if needed).

Another, less common, usage of Observed is in a Procedure model. In this case, Observed captures a person other than the procedure performer, who observed the performance of the procedure, and the time she observed the procedure (as well as patient/provider location and reason). An example is a nurse observing a physician insert a chest tube or place an IV. A Performed Attribution would capture who performed the procedure, but the Observed Attribution would be included in the model in addition to support cases in which a requirement exists to also capture who observed the procedure being performed.

Examples of the use of the Observed attribution are found in Appendix B – Attribution Examples: Observed Attribution Examples.

### Performed

The Performed attribution captures information related to the act of performing. The common use case for including Performed in a CEM is when the CEM represents a procedure. Performed then captures who performed the procedure and when the procedure was performed (as well as patient/provider location and reason). This will always be instantiated, either automatically as the person logged on or entered by the user when the performer is not the person logged in.

Examples of the use of the Performed attribution are found in Appendix B – Attribution Examples: Performed Attribution Examples.

### ReportReceived

The ReportedReceived captures information in a situation where the information contained in the containing statement is the result of an exchange between a clinician and someone other than the patient. An example is the mother of the patient providing medical history information to the clinician.

In this situation, a ReportRecieved instance contains two ‘whos’ (participants): The reporter (mother) and the receiver (clinician).

Another example is a paramedic in an emergency room reporting patient data to the clinician.

An example of the use of the ReportedReceived attribution is found in Appendix B – Attribution Examples: ReportedReceived Attribution Example.

### SpecimenCollected

The SpecimenCollected attribution captures information related to the act of gathering a sample to be sent for analysis and can be used in lab CEMs as well as applicable procedure CEMs. An example is shown in Appendix B – Attribution Examples: SpecimenCollected/SpecimenReceivedByLab/Resulted Attribution Example.

### SpecimenReceivedByLab

The SpecimenReceivedByLab attribution captures information related to a laboratory’s receipt of a specimen for processing. An example is shown in Appendix B – Attribution Examples: SpecimenCollected/SpecimenReceivedByLab/Resulted Attribution Example.

### Resulted

The Resulted attribution captures information related to the act of fulfilling a lab order, and only applies to labs. An example is shown in Appendix B – Attribution Examples: SpecimenCollected/SpecimenReceivedByLab/Resulted Attribution Example.

### Corrected

The Corrected attribution captures information related to the correction of a data model instance, e.g., a lab result whose value needs to be corrected or a blood pressure whose time needs to be corrected. Examples are shown in Appendix B – Attribution Examples: Corrected Attribution Examples

### Verified

The Verified attribution captures information related to the verifcation of a CEM instance for correctness. This may be instantiated either at the time of CEM instantiation by the logged-on user (as in the case of verifying a medication order) or entered by a second user (as in the case of verifying a blood transfusion, wasted medication or insulin dose). Examples are shown in Appendix B – Attribution Examples: Verified Attribution Examples.

### Ordered

The Ordered attribution captures information related to the act of ordering. An example is shown in Appendix B – Attribution Examples:Ordered Attribution Example.

### Discontinued

The Discontinued attribution captures information related to the act of stopping an existing order and applies only to orders. An example is shown in Appendix B – Attribution Examples: Discontinued Attribution Example.

### CounterSigned

The Countersigned attribution captures information related to the act of one entity reviewing and agreeing to a documentation performed by another entity. It might be used, for instance, for a physician to agree with orders written by a medical student. An example is shown in Appendix B – Attribution Examples: Countersigned Attribution Example.

### PutOnHold

The PutOnHold attribution captures information related to the act of placing an order on hold and applies only to orders. This is populated as a user puts an order on hold. An example is shown in “Appendix Appendix B – Attribution Examples”: “PutOnHold Attribution Example.”

### TakeOffHold

The TakeOffHold[[80]](#footnote-80) attribution captures information related to the act of removing the hold on an order and applies only orders. This is populated as a user takes an order off hold. An example is shown in : TakeOffHold Attribution Example.

### Other Attributions

Some other attributions are listed below:

#### Inferred

The Inferred attribution captures information related to the act of ‘assuming’ or ‘deducing’ information based on other information that may be sent and is currently used in only two CEMs -- MechanicalVentilationPeriod and UnitLOSPeriod. In the case of MechanicalVentilationPeriod the length of time on a mechanical ventilator is calculated using the start time of ventilation and the end time of ventilation and is therefore inferred, although the ventilated length of time is never stored. In the case of UnitLOSPeriod, the Length of Stay (LOS) is calculated using a start and end time. The actual calculated LOS is never stored.

#### IncubationStarted

The IncubationStarted attribution represents information related to the act of activating the “incubation” of a sample for microbiology. It is currently only found on MicroPanels.

#### Interpretated

The Interpreted attribution represents information related to the act of interpreting a (radiology) result. It is currently only on RadiologyResultPanel.

#### Transcribed

The Transcribed attribution represents information related to the act of transcribing a report. It is currently only on RadiologyResultPanel.

#### Vaccinated

The Vaccinated attribution represents information related to the act of vaccinating. It is currently only on InfluenzaVaccinationAssert.

Table . Summary of attribution models and models in which they are commonly used.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Attribution** | **Procedure** | **Assertion** | **Evaluation** | **EventAssertion** | **ImpressionEvaluation** | **Measurement** | **NumericSetting** | **Lab/Micro** | **Order** |
| **Corrected** | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| **CounterSigned** | No | No | No | No | No | No | No | No | Yes |
| **Discontinued** | No | No | No | No | No | No | No | No | Yes |
| **Documented\*** | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| **ImageTaken\*** | Yes | Yes | Yes | ? | Yes | No | No | No | No |
| **IncubationStarted\*** | No | No | No | No | No | No | No | Yes | No |
| **Inferred\*** | ? | ? | ? | ? | ? | Yes | Yes | No | No |
| **Interpreted\*** | Radiology Result Panel – may apply to others | | | | | | | | |
| **Observed** | Yes | Yes | No | Yes | No | No | No | No | No |
| **Ordered** | No | No | No | No | No | No | No | No | Yes |
| **Performed** | Yes | Yes | Yes | Yes | Yes | Yes | Yes | No | No |
| **PutOnHold** | No | No | No | No | No | No | No | No | Yes |
| **ReportReceived** | Yes | Yes | Yes | Yes | Yes | No | No | No | No |
| **Resulted** | No | No | No | No | No | No | No | Yes | No |
| **SpecimenCollected** | Yes | No | No | No | No | Yes | No | Yes | No |
| **SpecimenReceivedByLab** | No | No | No | No | No | Yes | No | Yes | No |
| **TakeOffHold** | No | No | No | No | No | No | No | No | Yes |
| **Transcribed\*** | Radiology Result Panel – may apply to others This would also apply to dictated notes that are transcribed. | | | | | | | | |
| **Vaccinated\*** | Used only on InfluenzaVaccinationAssert – may apply to others. | | | | | | | | |
| **Verified** | No | Yes | Yes | Yes | Yes | Yes | Yes | Yes | No |

## Modifiers

Some common modifiers are discussed below.

### Subject Modifier

The subject modifier is used to indicate that the data represented by a model instance pertains to someone other than the patient that owns the instance. Two examples will illustrate:

* Suppose during the prenatal examination of a pregnant woman, the fetal heart rate is observed to be 165 bpm. An instance of HeartRateMeas is stored in the woman’s record -- the woman is the “patient of record”, and “owns” the instance. However, to clarify that this heart rate instance is the fetus’s and not the patient’s, the subject modifier is used. “Data” of the subject modifier is populated with a code that means “fetus”.
* Another example is recording of family history information. To record that the patient’s grandmother suffered from heart disease, a heart disease model instance is stored in the patient’s record with a subject of “grandmother”[[81]](#footnote-81).

The subject modifier is necessary because the CEM strategy is to use the same heart rate model regardless of whether the subject is the patient or the fetus and to use the same heart disease model regardless of whether the subject is the patient or the patient’s grandmother. This reduces the proliferation of “pre-coordinated” models (e.g., “patient’s heart disease” and “grandmother’s heart disease”; “patient heart rate” and “fetal heart rate”; etc.) and ensures that a single CEM is used to represent all instances of a clinical concept[[82]](#footnote-82).

### Negation Indicator Modifier

NEEDS COMPLETION

The negation indicator modifier is used to negate what is stated in the data of an assertion model. For example, a model instance whose key is “Assertion” and value (data) is “alopecia” asserts that the patient has alopecia. If, however, the negation modifier is populated, the meaning of the instance becomes, “the patient does not have alopecia”. Simlarly, in *BreathSoundsAssertGroup* wheezing could be chosen from the data domain, and if the negationInd is changed to true, then this means the subject does not have wheezing breath sounds.

Negation negates the value choice of the model.

Guidelines:

* Value set of coughs cannot include “no cough” with the negation indicated included.
* Use of negation indicator would indicate that whatever the specific value (from the value set) is not present (not that there just isn't a cough)

### Uncertainty Modifier

NEEDS COMPLETION

This modifier is used to express levels of uncertainty.

The Uncertainty value set includes these values at this time:

Estimated, question of, unknown, rule out, unknown uncertainty, unlikely, unstable, unsure, confirmed by biopsy, definite, likely, not identified.

For example: In *ChestPainAssert*  the likelihood of the patient having actual chest pain may be unclear due to several other symptoms, so the value ‘unsure’ could be used in the model instance.

# Authoring issues

Authoring CEMs within the constraints of the CEM strategy and the CDL language leave the author with considerable latitude. The author is sometimes faced with difficult choices to make in modeling. Throughout this document, these issues are addressed in “Guidelines” sections.

Issues addressed in modeling and guidelines for addressing the issues are listed below:

## Creating a New Component CEM Versus Reusing an Existing CEM

At times, it is difficult to decide whether to create a new component model or to reuse an existing model. For guidance, consult the section “Creating a New Component CEM Versus Reusing an Existing CEM”.

## Modeling a Statement as an Evaluation versus an Assertion

It might be observed that by adjusting the name and organization of an assertion model, it might alternatively be modeled as an evaluation model (and vice versa). Guidelines for making this choice are found in the section “Evaluation VS. Assertion.”

## Creating a new statement CEM versus reusing a CEM

NEEDS COMPLETION

## Use of “normal” in a value set

A question that frequently arises in modeling evaluations is how to express that the property, organ, or system is “normal”. For guidance, see the section entitled Guideline: Use of “Normal”.

## Representing that data is absent

NEEDS COMPLETION

## Use of nullFlavor versus adding “unknown” or “other” to a value set

NEEDS COMPLETION

## Use of negation versus “not present” in a value set

NEEDS COMPLETION

## Modeling as a component or a statement

For a discussion, see the section “Guideline: Modeling as a component or a statement.”

## New Subtypes versus Generic Subtypes

THE FOLLOWING NEEDS COMPLETION

Do NOT create a new type if you're just doing one of the following:

* Constraining (subsetting) the value set of "data"
  + For example, if we had a model for respiratory failure with key=assertion and data = respiratoryFailure\_VALUESET\_\_ECID. The rule says if you want to represent "respiratory arrest" (where "resipiratory arrest" is in respiratoryFailure\_VALUESET\_ECID), and everything else in the model would be the same (same data type, same qualifiers with same cardinalities, etc.) this doesn’t call for a new "respiratory arrest" model that's the same as the "respiratory failure" model except with data constrained to “respiratory\_arrest\_ECID”. It's just the respiratory failure model with data = "respiratory arrest".
* Constraining properties like st.max, st.min, pq.max, etc.
* Constraining one of the above in some nested part of a model (e.g., if you want to constrain the "data" within one of the models qualifiers, that's not grounds for creating a new type).

You DO create a new subtype when:

* You need to constrain out a qualifier from the base model
* You want to constrain the data type that the base has
  + In this case, though, the data type on the parent would have to be “ANY”, since ANY is the only data type that has children (i.e., is constrainable)
* You constrain the cardinality of a qualifier (not backward compatible anymore)

## Large Value Sets in Component Models

NEEDS COMPLETION

The AlleviatingFactor, ExacerbatingFactor, and AssociatedSignAndSymptom component CEMs have value sets that are potentially very large. There are at least two strategies for using these CEMs and their value sets within other models.

1. Constrain the value set to the appropriate sub-value set. For example, when the HeadacheAssert model uses AlleviatingFactor as a qualifier, it may constrain the value set to HeadacheAlleviatingFactor, and create HeadacheAlleviatingFactor to contain just those values that commonly alleviate a headache (e.g., “darkness”, “lying motionless”, etc.). Note that AlleviatingFactor is a CWE model, so if the model is bound to a UI, and the UI user does not find the desired value in the value set, he can enter it as free text.
2. Do not constrain the value set any further. For example, when the HeadacheAssert model uses AlleviatingFactor as a qualifier, allow the qualifier to use the entire AlleviatingFactor value set. If this approach is used, when this model is bound to a UI, the UI would need to rely on good search/navigation mechanisms to allow a user to find the value he needs.

### Advantages and disadvantages of each approach

Approach 1 (constraint of the value set) advantages:

* Advantage:
  + The resulting constrained value sets are more appropriate to the use case.
* Disadvantage:
  + In constraining the value set, some coded values may be excluded. If the model is bound to a UI, a user will be required to free text the value when there was actually a code for the value, it just wasn’t included in the constrained value set.

Approach 2 (unconstrained value set)

* Advantage:
  + More coded values are included, decreasing the likelihood that the user of a UI will need to free text for codes that actually exist.
* Disadvantage:

## 

## Isosemantic models

NEEDS COMPLETION

### Practicality of “one model” for each data element

Examples of different granularity options:

* Pneumonia:
  + Pneumonia assertion
  + respiratory disorder
  + Finding/disorder
  + Assertion
  + CEM
* Potassium lab result:
  + Electrolyte
  + PQ lab result
  + Standard lab result
  + Measurement
  + CEM

### Derivations/summaries and calculations

Examples:

* Calculations
  + A-a gradient
  + BMI
  + Current age
  + Age (at some event)
* Derivations/summaries
  + Tobacco used
  + No known allergies
  + Suicide attempted

### “Context” or “Usage” models

Examples:

* Chief complaint
* Admit diagnosis
* Cause of death
* Complication of surgery
* Problem

### “UI” models

These models are utilized for UI display only and the actual instance information is stored in another model.

For example: A fetal heart rate is recorded in a HeartRateMeas model with ‘fetus’ as the subject. This way the value will be stored with the mother’s clinical data. To display fetal heart rate in an application a FetalHeartRateMeas model is utilized.

## Precoordination versus postcoordination

NEEDS COMPLETION

* Method included in the lab name or treated as a qualifier?
* Clinical drug versus drug name/strength/route
* Attribute names to include class name? – “Patient status” or just “status”? “Heart rate body location” or just “body location”? “Migraine alleviating factor” or just “alleviating factor”? Related to reuse.
* “BodyLocationPrecoord” or “BodyLocation”?

When do we create a “post-void urine output” model vs. a “urine output” model with a qualifier that indicates “post-void”?

what about “challenge glucose” with a qualifier of “challenge period” vs. “glucose” with qualifiers of “challengeflag” and “challenge period” vs. a “glucose post-10 hr fast” model?

* e.g., specimen description had very granular parts (body location, volume, turbidity, color, viscosity, etc.), we combined them. Rationale:
  + Some things people wanted to say were combinations of these parts, e.g., “Purulent” is a combination of colors, smells, etc.
  + It became unusable -- users didn’t want to go through multiple axes of
* when do we precoordinate choices, and when do we separate out different axes?
  + when there are only 20 or so or fewer practical precoordinated terms
  + when it would be hard to separate out by axis (e.g., route/method/device, clammy extremity finding
  + when it’s not important to search/retrieve by axis

## Denormalization

NEEDS COMPLETION

* Adding attributes that are “categories”
  + Drug category: store “penicllin” as well as “vancomycin”
  + Language: store “french” as well as “french canadian”
  + Rationale:
    - Performance optimization
    - Categorization may change over time; we want to capture what the categorization was at the time of storage
* Adding structures that really represent separate, related activities
  + SpecimenCollected in a lab result
  + Fulfillment in an order
* Related to what to put in the terminology/knowledge versus what to put in the model

## What to put in the model versus what to leave in terminology or knowledge

NEEDS COMPLETION

## Semantic links versus “pointer” attributes

NEEDS COMPLETION

## Qualifiers versus Panels versus semantic links

For guidance in choosing between using qualifiers, panels, and semantic links, see the section Qualifiers vs. Panels vs. Semantic Links.

## Generic versus specific models

NEEDS COMPLETION

### When to create separate models versus use the same model

## Use case differences in models

NEEDS COMPLETION

* “metadata”, sometimes use case-specific
* Use case-specific logical model attributes
* Implementation-specific attributes

## Modeling the implied procedure

See the section entitled, “Guideline: Procedure versus Measurement.”

## Separating the key code from the value set head concept – question answer

NEEDS COMPLETION

# Appendix A – Model and Terminology Changes and Backward Compatibility

The table below lists possible changes to CEMs and terminology and an indication of whether the changes are backward compatibility or not.

|  |  |  |
| --- | --- | --- |
| **Change** | **non-backward compatible** | **backward-compatible** |
| **Model Changes** | | |
| Change to a model that reflects a modification to the CDL language (addition, removal, modification of a CDL feature) | x |  |
| Creation of a new reference class | x |  |
| Creation of a new non-reference model |  | x |
| Change to a model's "is" statement (e.g., changing "is statement" to "is Patient") | x |  |
| Change to a model's "extends" or "restricts" statement (e.g., changing "extends Patient" to "extends PatientEncounter") | x |  |
| Change of a model from partial to non-partial | x |  |
| Change of a model from non-partial to partial | x |  |
| Addition of an optional qualifier/modifier/attribution/item to a model |  | x |
| Addition of a required qualifier/modifier/attribution/item to a model | x |  |
| Deletion of a qualifier/modifier/attribution/item from a model | x |  |
| Value Set restriction for partial attributes from the reference model-example status restriction from instance reference class. | x |  |
| Increase in cardinality of a qualifier/modifier/attribution/item |  | X |
| (Special Case of the Above) Increase in cardinality of a qualifier/modifier/attribution/item from "1" |  | X |
| Restriction of cardinality of a qualifier/modifier/attribution/item (including restrictions that make an originally optional element now required) | x |  |
| Widening of a data component's physiologic range |  | x |
| Restriction of a data component's physiologic range | x |  |
| Removal of a value set constraint from a model |  | x |

|  |  |  |
| --- | --- | --- |
| **Change** | **non-backward compatible** | **backward-compatible** |
| **Model Changes (continued)** | | |
| Addition of a value set constraint to a model | x |  |
| Broadening change of units of measure (eg. specific value to value set that includes previous specific unit) |  | x |
| Change of a data component's data type | x |  |
| Change a CD data component from "CNE" (i.e., a CD where the code is required) to a "CWE" (i.e., a CD where the code is optional) |  | x |
| Change a CD data component from "CWE" (i.e., a CD where the code is optional) to a "CNE" (i.e., a CD where the code is required) | x |  |
| Change to a data component's default value | x |  |
| Narrowing change to a value set constraint (for example, a narrowing unit of measure constraint) | x |  |
| Splitting of a single qualifier/modifier/attribution/item into multiple qualifiers/modifier/attributions/items [Note: this can be viewed as a deletion and an addition, and consequently, because of the deletion, is non-backward compatible and grounds for creation of a new CEType] | x |  |
| Combination of multiple qualifiers/modifiers/attributions/items into a single qualifier/modifier/attribution/item [Note: this can be viewed as multiple deletions and an addition, and consequently, because of the deletions, is non-backward compatible and grounds for creation of a new CEType] | x |  |
| Replacement of one qualifier/modifier/attribution/item with another [Note: this can be viewed as a deletion and an addition, and consequently, because of the deletion, is non-backward compatible and grounds for creation of a new CEType] | x |  |

|  |  |  |
| --- | --- | --- |
| **Change** | **non-backward compatible** | **backward-compatible** |
| **Model Changes (continued)** | | |
| Change to model name | x |  |
| Change of a key concept reference (e.g., changing "key code (XXX\_KEY\_ECID)" to "key code (YYY\_KEY\_ECID)" | x |  |
| Change to the order of qualifiers, attributions, modifiers, and items in the model |  | x |
| Change to the labels used in a model (e.g., changing "qualifier BodyLocation bodylocation card(0..1)" to "qualifier BodyLocation bl card(0..1)" |  | x |
| Deletion of a model | x |  |
| Change of a code constraint to a domain (value set) constraint |  | x |
| Change of a domain (value set) constraint to a code constraint | x |  |
| Change of a domain (value set) constraint to refer to another valueset, where the new valueset is a superset of the former valueset |  | x |
| Change of a domain (value set) constraint to refer to another valueset, where the new valueset is a subset of the former valueset | x |  |
| Change of a domain constraint to refer to another, non-overlapping valueset | x |  |
| Move of a qualifier/attribution to a reference class (must retain the same name and attributes) |  | x |
| Any addition to a reference class (follow same rules as non-reference class changes) Is this one both? | x | x |
| Combination of two models into one (essentially a new model that replaces multiple previously released models) | x |  |

|  |  |  |
| --- | --- | --- |
| **Change** | **non-backward compatible** | **backward-compatible** |
| **Terminology Changes** | | |
| **Concepts** |  |  |
| Inactivation of a concept that's a member of a value set, where the affected member has a clinical element display, i.e., it is referenced in a model | x |  |
| Inactivation of a concept that's a member of a value set, where the affected member does not have a clinical element display, i.e., it is not referenced in a model | x |  |
| **Designations/Definitions** |  |  |
| New clinical element displays (assuming the applicable model changes are also made) |  | x |
| Any modification (besides activation) to a Clinical element display (assuming any applicable change has been made in the model that refers to the display and the meaning of the concept isn't changing) |  | x |
| Addition/update/inactivation of designations (except CEM Display) |  | x |
| Moving a Clinical Element display to a different entity/concept | x |  |
| Addition/update/inactivation of definitions (must always maintain one active definition for each concept) |  | x |
| **Relationships** |  |  |
| Addition/activation/inactivation of relationships (not elsewhere specified as a non-backward compatible change) |  | x |
| Inactivation of a hierarchical relationship (IS\_A) where the source concept has a clinical element display | x |  |
| Inactivation of a relationship where a value set would be narrowed by the change (if value set is referenced by a model) | x |  |

|  |  |  |
| --- | --- | --- |
| **Change** | **non-backward compatible** | **backward-compatible** |
| **Terminology Changes (continued)** | | |
| **Mappings** |  |  |
| Changes to Mappings (needs further elaboration; discuss with Interop) |  |  |
| **Value Sets** |  |  |
| Addition of a concept to a value set (i.e., activation of the relationship that includes the concept in the value set), where the value set is referenced by a model |  | x |
| Removal of a concept from a value set (i.e., inactivation of the relationship that includes the concept in the value set), where the value set is referenced by a model | x |  |
| **Properties** |  |  |
| Addition/update/inactivation of properties (Are there any property inactivations that would be breaking? Not currently, but would follow similar rules as designations as they are used more) |  | x |

# Appendix B – Rationale for Assertion pattern

Alternatives considered: NEEDS COMPLETION

The rationale for this pattern was twofold:

* + 1. It was judged to be most consistent with the semantics of terminologies we expect to map “data” to. NEEDS COMPLETION
    2. The pattern followed the conclusions of the HL7 TermInfo effort. NEEDS COMPLETION

If key =”pain” and data=”present”, this would be an evaluation model, where the “pain” code really means “evaluation of pain” (as opposed to the physiologic condition of pain) and “present” is the “result” of or “answer” to the evaluation. In contrast, an assertion model is a definite affirmation that the state is present or the event occurred.[[83]](#footnote-83) Having the key code represent the condition present in the patient (and not evaluation of that condition) allows us to more cleanly map to standard terminologies.

# Appendix C – Attribution Examples

The following examples illustrate the use of attributions in CEMs. The examples are “logical”, i.e., they do not attempt to represent a particular implementation technology. They also do not attempt to show all elements of the attribution or the models in which they are contained.

## Observed Attribution Examples

The Observed attribution captures the act of observing that resulted in the information of the containing CEM.

**Observed Example 1**

A heart rate measurement is made by a nurse. Below is a simplified depiction of the resulting instance of the HeartRateMeas model. The model is a measurement model, observed captures the who, when, where, and why of the act of observing which resulted in the measurement data. This is the most common use of the observed attribution.

HeartRateMeas

Key:<code that means “heart rate measurement”>

Data: 70

Attribution Observed // who, when and where of observing the measurement

Key:<code that means “attribution”>

Data: <code that means “observed”>

Qualifier StartTime // start of the *observation* of the measurement

Key:<code that means “start time”>

Data: 2012-04-04T16:00Z

Qualifier EndTime

Key:<code that means “end time”>

Data: 2012-04-04T16:00Z

Qualifier Participant

Key:<code that means “participant”>

Data: <code that means “performer”> // who performed the observation

Qualifier Role

Key:<code that means “role”>

Data: <code that means “nurse”>

Qualifier IndividualPerson

Key:<code that means “individual person”>

Item PersonIdentifier

Key:<code that means “person identifier”>

Data: <identifier for the performer>

**Observed Example 2**

A chest tube is inserted by a physician. A nurse observes the procedure. Below is a simplified depiction of the resulting instance of the ChestTubeInsertion model. The model is a procedure model. The most important attribution in a procedure model is the Performed attribution, which captures the who, when, where, and why of the act of performing the procedure. The Observed attribuiton plays a secondary role, capturing the who, when, where, and why of the act of observing a procedure being performed.

ChestTubeInsertion

Key:<code that means “procedure”>

Data: <code that means “chest tube insertion”>

Attribution Performed // who, when, and where of performing the procedure

Key:<code that means “attribution”>

Data: <code that means “performed”>

Qualifier StartTime // start of the *performance* of the procedure

Key:<code that means “start time”>

Data: 2012-04-04T15:55Z

Qualifier EndTime

Key:<code that means “end time”>

Data: 2012-04-04T16:55Z

Qualifier Participant

Key:<code that means “participant”>

Data: <code that means “performer”> // who performed the performance of

// the procedure

Qualifier Role

Key:<code that means “role”>

Data: <code that means “physician”>

Qualifier IndividualPerson

Key:<code that means “individual person”>

Item PersonIdentifier

Key:<code that means “person identifier”>

Data: <identifier for the performer>

Attribution Observed // who, when and where of observing the procedure

Key:<code that means “attribution”>

Data: <code that means “observed”>

Qualifier StartTime // start of the *observation* of the procedure. (Note the times imply that

//only a portion of the entire procedure was observed.)

Key:<code that means “start time”>

Data: 2012-04-04T16:00Z

Qualifier EndTime

Key:<code that means “end time”>

Data: 2012-04-04T16:30Z

Qualifier Participant

Key:<code that means “participant”>

Data: <code that means “performer”> // who performed the observation of

// the procedure

Qualifier Role

Key:<code that means “role”>

Data: <code that means “nurse”>

Qualifier IndividualPerson

Key:<code that means “individual person”>

Item PersonIdentifier

Key:<code that means “person identifier”>

Data: <identifier for the performer>

**Observed Example 3**

A nurse is performing a rape exam on a patient. The investigating officer is present to ensure the chain of evidence is maintained. The model is a procedure model. Again, Performed is the most important attribution, Observed supports the uncommon use case of someone observing the procedure being performed.

RapeExam

Key:<code that means “procedure”>

Data: <code that means “rape examination”>

Attribution Performed // who, when, and where of performing the procedure

Key:<code that means “attribution”>

Data: <code that means “performed”>

Qualifier StartTime // start of the *performance* of the procedure

Key:<code that means “start time”>

Data: 2012-04-04T15:55Z

Qualifier EndTime

Key:<code that means “end time”>

Data: 2012-04-04T16:55Z

Qualifier Participant

Key:<code that means “participant”>

Data: <code that means “performer”> // who performed the procedure

Qualifier Role

Key:<code that means “role”>

Data: <code that means “nurse”>

Qualifier IndividualPerson

Key:<code that means “individual person”>

Item PersonIdentifier

Key:<code that means “person identifier”>

Data: <identifier for the performer>

Attribution Observed // who, when and where of observing the procedure

Key:<code that means “attribution”>

Data: <code that means “observed”>

Qualifier StartTime // start of the *observation* of the procedure

Key:<code that means “start time”>

Data: 2012-04-04T15:55Z

Qualifier EndTime

Key:<code that means “end time”>

Data: 2012-04-04T16:55Z

Qualifier Participant

Key:<code that means “participant”>

Data: <code that means “performer”> // who performed the observation of

// the procedure

Qualifier Role

Key:<code that means “role”>

Data: <code that means “investigating officer”>

Qualifier IndividualPerson

Key:<code that means “individual person”>

Item PersonIdentifier

Key:<code that means “person identifier”>

Data: <identifier for the performer>

**Observed Example 4**

A patient arrives on the inpatient floor with an IV from the surgical suite. Computer documentation is not live in surgery. The admitting nurse must document the presence of an IV line in order to document an assessment of the IV site. The nurse is not aware of the actual person who inserted the line, nor is he interested in documenting the who, when, where, or why of the act of performing the placement procedure. He is just documenting that an IV line is present. This model is thus an assertion model, asserting the presence of an IV (not documenting the procedure of placing an IV).

IVAssert

Key: <code for “assertion”>

Data: <code that means “IV line present”>

Attribution Observed // who, when and where of observing the presence of the line

Key:<code that means “attribution”>

Data: <code that means “observed”>

Qualifier StartTime // start of the *observation* of the line

Key:<code that means “start time”>

Data: 2012-04-04T16:00Z

Qualifier EndTime

Key:<code that means “end time”> // same as start time since this is a point-in-time

//observation

Data: 2012-04-04T16:00Z

Qualifier Participant

Key:<code that means “participant”>

Data: <code that means “performer”> // who performed the observation

Qualifier Role

Key:<code that means “role”>

Data: <code that means “nurse”>

Qualifier IndividualPerson

Key:<code that means “individual person”>

Item PersonIdentifier

Key:<code that means “person identifier”>

Data: <identifier for the performer>

## Performed Attribution Examples

The Performed attribution captures the who, when, and where information related to performing some action. It is used within a procedure model (i.e., a model in which key = “procedure” and data = the name of the procedure), to indicate who performed the procedure, when it was performed, and where it was performed,

**Peformed Example 1**

A chest tube is inserted by a physician. Note that this example is the same as Example 2 in the section, except no Observed attribution is populated because in this example no observation of the procedure is being recorded – only the performance.

ChestTubeInsertion

Key:<code that means “procedure”>

Data: <code that means “chest tube insertion”>

Attribution Performed // who, when, and where of performing the procedure

Key:<code that means “attribution”>

Data: <code that means “performed”>

Qualifier StartTime // start of the *performance* of the procedure

Key:<code that means “start time”>

Data: 2012-04-04T15:55Z

Qualifier EndTime

Key:<code that means “end time”>

Data: 2012-04-04T16:55Z

Qualifier Participant

Key:<code that means “participant”>

Data: <code that means “performer”> // who performed the procedure

Qualifier Role

Key:<code that means “role”>

Data: <code that means “physician”>

Qualifier IndividualPerson

Key:<code that means “individual person”>

Item PersonIdentifier

Key:<code that means “person identifier”>

Data: <identifier for the performer>

**Performed Example 2**

A licensed practical nurse documents a breath sound assessment.

BreathSoundAssessment

Key:<code that means “procedure”>

Data: <code that means “breath sound assessment”>

Attribution Performed // who, when, and where of performing the procedure

Data: <code that means “performed”>

Qualifier StartTime // start of the *performance* of the procedure

Key:<code that means “start time”>

Data: 2012-04-04T15:55Z

Qualifier EndTime

Key:<code that means “end time”>

Data: 2012-04-04T16:55Z

Qualifier Participant

Key:<code that means “participant”>

Data: <code that means “performer”> // who performed the procedure

Qualifier Role

Key:<code that means “role”>

Data: <code that means “lpn”>

Qualifier IndividualPerson

Key:<code that means “individual person”>

Item PersonIdentifier

Key:<code that means “person identifier”>

Data: <identifier for the performer>

## ReportedReceived Attribution Example

ReportedReceived is used to capture information in a situation where the information represented by the containing statement was received from another party (as opposed to directly from the patient).

In this situation, a ReportRecieved instance contains two ‘whos’, i.e., participants : The reporter (mother) and the receiver (clinician).

**ReportedReceived Example 1**

A mother reports a history of headache in her child.

HeadacheAssert

Key: <code that means “assertion”>

Data: <code that means “headache assert”>

Attribution ReportedReceived // who, when, and where of reporting and receiving the

// information regarding the headache

Key: <code that means “attribution”>

Data: <code that means “reported-received”>

Qualifier StartTime // start of the *reporting and receiving* of the procedure

Key:<code that means “start time”>

Data: 2012-04-04T15:55Z

Qualifier EndTime

Key:<code that means “end time”>

Data: 2012-04-04T15:55Z

Qualifier Participant

Key:<code that means “participant”>

Data: <code that means “reporter”> // who reported the information

Qualifier Role

Key:<code that means “role”>

Data: <code that means “mother”>

Qualifier IndividualPerson

Item PersonName

. . .

Qualifier Participant

Key:<code that means “participant”>

Data: <code that means “receiver”> // who received the information

Qualifier Role

Key:<code that means “role”>

Data: <code that means “clinician”>

Qualifier IndividualPerson

Key:<code that means “individual person”>

Item PersonIdentifier

Key:<code that means “person identifier”>

Data: <identifier for the receiver>

## SpecimenCollected/SpecimenReceivedByLab/Resulted Attribution Example

SpecimenCollected, SpecimenReceivedByLab, and Resulted are three attributions found in the lab result models (StandardLabObs and its data type-specific specializations), so the following example of their usage presents them together. SpecimenCollected represents the who, when, and where information related to the collection of the specimen whose analysis resulted in the instance of the CEM, while SpecimenReceivedByLab represents the who, when, and where information related to the receipt of the specimen by the lab. Resulted represents the who, when, and where information related to generating the lab result.

**SpecimenCollected/SpecimenReceivedByLab/Resulted Example 1**

A blood specimen is collected pursuant to running a glucose test.

StandardLabObsQuantitative

Key: <code that means “glucose serum or plasma”>

Data: <value of the glucose in mg/dL>

Attribution SpecimenCollected // who, when, and where related to the specimen collection

Key: <code that means “attribution”>

Data: <code that means “specimen collected”>

Qualifier StartTime // start of the specimen collection

Key: <code that means “start time”>

Data: 2012-04-04T15:55Z

Qualifier EndTime // likely the same as the start time

Key: <code that means “end time”>

Data: 2012-04-04T15:55Z

Qualifier Participant

Key: <code that means “participant”>

Data: <code that means “performer”> // who performed the collection

Qualifier Role

Key: <code that means “role”>

Data: <code that means “phlebotomist”>

Qualifier IndividualPerson

Key: <code that means “individual person”>

Item PersonIdentifier

Key: <code that means “person identifier”>

Data: <identifier for the performer>

Attribution SpecimenReceivedByLab // who, when, and where related to the receipt of the

// specimen by the lab

Key: <code that means “attribution”>

Data: <code that means “specimen received by lab”>

Qualifier StartTime // time the specimen was received. (Since “receipt” is a point-in-time event,

// the start time and end time will be the same.)

Key: <code that means “start time”>

Data: 2012-04-04T15:55Z

Qualifier EndTime

Key: <code that means “end time”>

Data: 2012-04-04T15:55Z

Qualifier Participant

Key: <code that means “participant”>

Data: <code that means “performer”> // who or what received the specimen

Qualifier Role

Key: <code that means “role”>

Data: <code that means “phlebotomist”>

Qualifier IndividualPerson

Key: <code that means “individual person”>

Item PersonIdentifier

Key: <code that means “person identifier”>

Data: <identifier for the receiver>

Attribution Resulted // who, when, and where related to generating the lab result

Key: <code that means “attribution”>

Data: <code that means “resulted”>

Qualifier StartTime // time the lab result was generated. (Since generating the result is a

// point-in-time event, the start time and end time will be the same.)

Key: <code that means “start time”>

Data: 2012-04-04T15:55Z

Qualifier EndTime

Key: <code that means “end time”>

Data: 2012-04-04T15:55Z

Qualifier Participant

Key: <code that means “participant”>

Data: <code that means “performer”> // who or what performed the generation of

// the result

Qualifier Role

Key: <code that means “role”>

Data: <code that means “lab technician”>

Qualifier IndividualPerson

Key: <code that means “individual person”>

Item PersonIdentifier

Key: <code that means “person identifier”>

Data: <identifier for the receiver>

## Corrected Attribution Examples

Corrected represents the who, when, and where information related to correcting a previously stored instance.

**Corrected Example 1**

The lab previously reported the potassium level to be 2.4, but corrected the value to 4.2.

StandardLabObsQuantitative

Key: <code that means “potassium serum or plasma”>

Data: <value of the potassium in mEq/L>

Attribution Corrected // who, when, and where related to the correction

Key: <code that means “attribution”>

Data: <code that means “corrected”>

Qualifier StartTime // time of the correction. (Since correcting is a point-in-time event,

// the start time and end time will be the same.)

Key: <code that means “start time”>

Data: 2012-04-04T15:55Z

Qualifier EndTime

Key: <code that means “end time”>

Data: 2012-04-04T15:55Z

Qualifier Participant

Key: <code that means “participant”>

Data: <code that means “performer”> // who performed the correction

Qualifier Role

Key: <code that means “role”>

Data: <code that means “lab technician”>

Qualifier IndividualPerson

Key: <code that means “individual person”>

Item PersonIdentifier

Key: <code that means “person identifier”>

Data: <identifier for the performer>

**Corrected Example 2**

The nurse charted the heart rate measurement to be 70 at 16:00, but changed the time to 16:15. Note this is the same as Observed Example 1, with the start and end time changed and the additional Corrected attribution added.

HeartRateMeas

Key:<code that means “heart rate measurement”>

Data: 70

Attribution Observed // who, when and where of observing the measurement

Key:<code that means “attribution”>

Data: <code that means “observed”>

Qualifier StartTime // start of the *observation* of the measurement

Key:<code that means “start time”>

Data: 2012-04-04T16:15Z

Qualifier EndTime

Key:<code that means “end time”>

Data: 2012-04-04T16:15Z

Qualifier Participant

Key:<code that means “participant”>

Data: <code that means “performer”> // who performed the observation

Qualifier Role

Key:<code that means “role”>

Data: <code that means “nurse”>

Qualifier IndividualPerson

Key:<code that means “individual person”>

Item PersonIdentifier

Key:<code that means “person identifier”>

Data: <identifier for the performer>

Attribution Corrected // who, when and where of correcting the measurement

Key:<code that means “attribution”>

Data: <code that means “corrected”>

Qualifier StartTime // start of the *correction* of the measurement

Key:<code that means “start time”>

Data: 2012-04-04T18:00Z

Qualifier EndTime

Key:<code that means “end time”>

Data: 2012-04-04T18:00Z

Qualifier Participant

Key:<code that means “participant”>

Data: <code that means “performer”> // who performed the correction

Qualifier Role

Key:<code that means “role”>

Data: <code that means “nurse”>

Qualifier IndividualPerson

Key:<code that means “individual person”>

Item PersonIdentifier

Key:<code that means “person identifier”>

Data: <identifier for the performer>

## Verified Attribution Examples

Verified captures the who, when, and where information related to verifying the piece of data represented by the containing model instance.

**Verified Example 1**

The MD orders a MS at 2-4 mg IV PRN. The pharmacist enters the lab into the computer. The nurse must verify against the written order that the medication is in the computer correctly**.**

OrderMedCont

Key:<code that means “continuous medication order”>

Item Order

Key:<code that means “order”>

Item OrderableItem

Key:<code that means “orderable item”>

Data: <code that means “morphine sulfate”>

Item Formulation

Key:<code that means “formulation”>

Data:<code that means “ampule”>

. . .

. . .

. . .

Attribution Verified[[84]](#footnote-84) // who, when and where of verifying the order

Key:<code that means “attribution”>

Data: <code that means “verified”>

Qualifier StartTime // start of the verification of the order

Key:<code that means “start time”>

Data: 2012-04-04T16:15Z

Qualifier EndTime

Key:<code that means “end time”> // end time is same as start time since the action is

// a discrete, point-in-time action

Data: 2012-04-04T16:15Z

Qualifier Participant

Key:<code that means “participant”>

Data: <code that means “performer”> // who performed the verification

Qualifier Role

Key:<code that means “role”>

Data: <code that means “nurse”>

Qualifier IndividualPerson

Key:<code that means “individual person”>

Item PersonIdentifier

Key:<code that means “person identifier”>

Data: <identifier for the performer>

**Verified Example 2**

A patient needs one unit of packed red blood cells. Two clinicians check the unit against the patient information to verify the correct unit is being given.

BloodTransfusion

Key:<code that means “blood transfusion”>

. . .

Attribution Verified // who, when and where of verifying the transfusion

Key:<code that means “attribution”>

Data: <code that means “verified”>

Qualifier StartTime // start of the verification of the transfusion

Key:<code that means “start time”>

Data: 2012-04-04T16:15Z

Qualifier EndTime

Key:<code that means “end time”> // end time is same as start time since the action is

// a discrete, point-in-time action

Data: 2012-04-04T16:15Z

Qualifier Participant // first verifier

Key:<code that means “participant”>

Data: <code that means “performer”> // who performed the verification

Qualifier Role

Key:<code that means “role”>

Data: <code that means “nurse”>

Qualifier IndividualPerson

Key:<code that means “individual person”>

Item PersonIdentifier

Key:<code that means “person identifier”>

Data: <identifier for the performer>

Qualifier Participant // second verifier

Key:<code that means “participant”>

Data: <code that means “performer”> // who performed the verification

Qualifier Role

Key:<code that means “role”>

Data: <code that means “nurse”>

Qualifier IndividualPerson

Key:<code that means “individual person”>

Item PersonIdentifier

Key:<code that means “person identifier”>

Data: <identifier for the performer>

## Ordered Attribution Example

Ordered captures the who, when, and where information related to ordering the piece of data represented by the containing model instance.

**Ordered Example 1[[85]](#footnote-85)**

A CBC is ordered.

OrderLab

Key:<code that means “lab order”>

Item Order

Key:<code that means “order”>

Item OrderableItem

Key:<code that means “orderable item”>

Data: <code that means “complete blood count”>

. . .

. . .

. . .

Attribution Ordered // who, when and where of ordering the lab

Key:<code that means “attribution”>

Data: <code that means “ordered”>

Qualifier StartTime // start of the ordering of the lab

Key:<code that means “start time”>

Data: 2012-04-04T16:15Z

Qualifier EndTime

Key:<code that means “end time”>

Data: 2012-04-04T16:15Z

Qualifier Participant

Key:<code that means “participant”>

Data: <code that means “performer”> // who ordered the lab

Qualifier Role

Key:<code that means “role”>

Data: <code that means “physician”>

Qualifier IndividualPerson

Key:<code that means “individual person”>

Item PersonIdentifier

Key:<code that means “person identifier”>

Data: <identifier for the orderer>

## Discontinued Attribution Example

Discontinued captures the who, when, and where information related to discontinuing the piece of data (an action) represented by the containing model instance.

**Discontinued Example 1**

An order for Morphine is discontinued. (Note this is the same as Verified Example 1, with a Discontinued attribution added.)

OrderMedCont

Key:<code that means “continuous medication order”>

Item Order

Key:<code that means “order”>

Item OrderableItem

Key:<code that means “orderable item”>

Data: <code that means “morphine sulfate”>

Item Formulation

Key:<code that means “formulation”>

Data:<code that means “ampule”>

. . .

. . .

. . .

Attribution Verified[[86]](#footnote-86) // who, when and where of verifying the order

Key:<code that means “attribution”>

Data: <code that means “verified”>

Qualifier StartTime // start of the verification of the order

Key:<code that means “start time”>

Data: 2012-04-04T16:15Z

Qualifier EndTime

Key:<code that means “end time”> // end time is same as start time since the action is

// a discrete, point-in-time action

Data: 2012-04-04T16:15Z

Qualifier Participant

Key:<code that means “participant”>

Data: <code that means “performer”> // who performed the verification

Qualifier Role

Key:<code that means “role”>

Data: <code that means “nurse”>

Qualifier IndividualPerson

Key:<code that means “individual person”>

Item PersonIdentifier

Key:<code that means “person identifier”>

Data: <identifier for the performer>

Attribution Discontinued // who, when and where of discontinuing the order

Key:<code that means “attribution”>

Data: <code that means “discontinued”>

Qualifier StartTime // time of the discontinuation of the order

Key:<code that means “start time”>

Data: 2012-04-04T:22:00Z

Qualifier EndTime

Key:<code that means “end time”> // end time is same as start time since the action is

// a discrete, point-in-time action

Data: 2012-04-04T22:00Z

Qualifier Participant

Key:<code that means “participant”>

Data: <code that means “performer”> // who performed the verification

Qualifier Role

Key:<code that means “role”>

Data: <code that means “nurse”>

Qualifier IndividualPerson

Key:<code that means “individual person”>

Item PersonIdentifier

Key:<code that means “person identifier”>

Data: <identifier for the performer>

## **Documented Attribution Example**

Documented captures the who, when, and where information related to documenting the piece of data represented by the containing model instance.

**Documented Example 1**

A medical student diagnoses the patient with COPD.

COPDAssert

Key:<code that means “assertion”>

Data:<code that means “chronic obstructive pulmonary disease”>

. . .

Attribution Documented // who, when and where of documenting the assertion

Key:<code that means “attribution”>

Data: <code that means “documented”>

Qualifier StartTime // start of the documentation of the assertion

Key:<code that means “start time”>

Data: 2012-04-04T16:15Z

Qualifier EndTime

Key:<code that means “end time”> // end time is same as start time since the action is

// a discrete, point-in-time action

Data: 2012-04-04T16:15Z

Qualifier Participant

Key:<code that means “participant”>

Data: <code that means “performer”> // who performed the documentation

Qualifier Role

Key:<code that means “role”>

Data: <code that means “medical student”>

Qualifier IndividualPerson

Key:<code that means “individual person”>

Item PersonIdentifier

Key:<code that means “person identifier”>

Data: <identifier for the performer>

## **Countersigned Attribution Example**

Countersigned captures the who, when, and where information related to countersigning the piece of data represented by the containing model instance.

**Countersigned Example 1**

A medical student diagnoses the patient with COPD. The attending MD verifies by countersigning that the diagnosis is correct. (Note this example is the same as Documented Example 1, with a Countersigned attribution added.)

COPDAssert

Key:<code that means “assertion”>

Data:<code that means “chronic obstructive pulmonary disease”>

. . .

Attribution Documented // who, when and where of documenting the assertion

Key:<code that means “attribution”>

Data: <code that means “documented”>

Qualifier StartTime // start of the documentation of the assertion

Key:<code that means “start time”>

Data: 2012-04-04T16:15Z

Qualifier EndTime

Key:<code that means “end time”> // end time is same as start time since the action is

// a discrete, point-in-time action

Data: 2012-04-04T16:15Z

Qualifier Participant

Key:<code that means “participant”>

Data: <code that means “performer”> // who performed the documentation

Qualifier Role

Key:<code that means “role”>

Data: <code that means “medical student”>

Qualifier IndividualPerson

Key:<code that means “individual person”>

Item PersonIdentifier

Key:<code that means “person identifier”>

Data: <identifier for the performer>

Attribution Countersigned // who, when and where of countersigning the assertion

Key:<code that means “attribution”>

Data: <code that means “verified”>

Qualifier StartTime // time of the countersigning of the order

Key:<code that means “start time”>

Data: 2012-04-04T17:15Z

Qualifier EndTime

Key:<code that means “end time”> // end time is same as start time since the action is

// a discrete, point-in-time action

Data: 2012-04-04T17:15Z

Qualifier Participant

Key:<code that means “participant”>

Data: <code that means “performer”> // who performed the countersigning

Qualifier Role

Key:<code that means “role”>

Data: <code that means “nurse”>

Qualifier IndividualPerson

Key:<code that means “individual person”>

Item PersonIdentifier

Key:<code that means “person identifier”>

Data: <identifier for the performer>

## **PutOnHold Attribution Example**

PutOnHold captures the who, when, and where information related to putting the piece of data (an order) represented by the containing model instance on hold.

**PutOnHold Example 1**

An order for tube feedings is placed on hold.

OrderNutrition

Key:<code that means “nutrition order”>

Item Order

Key:<code that means “order”>

Item OrderableItem

Key:<code that means “orderable item”>

Data: <code that means “jevity tube feeding”>

. . .

. . .

Attribution Ordered // who, when and where of ordering the tube feeding

Key:<code that means “attribution”>

Data: <code that means “ordered”>

Qualifier StartTime // time the tube feeding was ordered

Key:<code that means “start time”>

Data: 2012-04-04T16:15Z

Qualifier EndTime

Key:<code that means “end time”>

Data: 2012-04-04T16:15Z // same as start time since the action is a discrete,

// point-in-time action

Qualifier Participant

Key:<code that means “participant”>

Data: <code that means “performer”> // who ordered the tube feeding

Qualifier Role

Key:<code that means “role”>

Data: <code that means “physician”>

Qualifier IndividualPerson

Key:<code that means “individual person”>

Item PersonIdentifier

Key:<code that means “person identifier”>

Data: <identifier for the orderer>

Attribution PutOnHold // who, when and where of putting the order on hold

Key:<code that means “attribution”>

Data: <code that means “put on hold”>

Qualifier StartTime // time of putting the order on hold

Key:<code that means “start time”>

Data: 2012-04-05T20:00Z

Qualifier EndTime

Key:<code that means “end time”> // end time is same as start time since the action is

// a discrete, point-in-time action

Data: 2012-04-05T20:00Z

Qualifier Participant

Key:<code that means “participant”>

Data: <code that means “performer”> // who put the order on hold

Qualifier Role

Key:<code that means “role”>

Data: <code that means “physician”>

Qualifier IndividualPerson

Key:<code that means “individual person”>

Item PersonIdentifier

Key:<code that means “person identifier”>

Data: <identifier for the performer>

## TakeOffHold Attribution Example

TakeOffHold[[87]](#footnote-87) represents the who, when, and where information related to taking an order (or other action represented by the containing CEM) off hold, i.e., resuming the action.

**TakeOffHold Example 1**

A patient’s tube feeding is resumed. (Note this is the same as PutOnHold Example1, with a TakeOffHold attribution added.)

OrderNutrition

Key:<code that means “nutrition order”>

Item Order

Key:<code that means “order”>

Item OrderableItem

Key:<code that means “orderable item”>

Data: <code that means “jevity tube feeding”>

. . .

. . .

Attribution Ordered // who, when and where of ordering the tube feeding

Key:<code that means “attribution”>

Data: <code that means “ordered”>

Qualifier StartTime // time the tube feeding was ordered

Key:<code that means “start time”>

Data: 2012-04-04T16:15Z

Qualifier EndTime

Key:<code that means “end time”>

Data: 2012-04-04T16:15Z // same as start time since the action is a discrete,

// point-in-time action

Qualifier Participant

Key:<code that means “participant”>

Data: <code that means “performer”> // who ordered the tube feeding

Qualifier Role

Key:<code that means “role”>

Data: <code that means “physician”>

Qualifier IndividualPerson

Key:<code that means “individual person”>

Item PersonIdentifier

Key:<code that means “person identifier”>

Data: <identifier for the orderer>

Attribution PutOnHold // who, when and where of putting the order on hold

Key:<code that means “attribution”>

Data: <code that means “put on hold”>

Qualifier StartTime // time of putting the order on hold

Key:<code that means “start time”>

Data: 2012-04-05T20:00Z

Qualifier EndTime

Key:<code that means “end time”> // end time is same as start time since the action is

// a discrete, point-in-time action

Data: 2012-04-05T20:00Z

Qualifier Participant

Key:<code that means “participant”>

Data: <code that means “performer”> // who put the order on hold

Qualifier Role

Key:<code that means “role”>

Data: <code that means “physician”>

Qualifier IndividualPerson

Key:<code that means “individual person”>

Item PersonIdentifier

Key:<code that means “person identifier”>

Data: <identifier for the performer>

Attribution TakeOffHold // who, when and where of taking the order off hold

Key:<code that means “attribution”>

Data: <code that means “take off hold”>

Qualifier StartTime // time of taking the order off hold

Key:<code that means “start time”>

Data: 2012-04-07T:07:00Z

Qualifier EndTime

Key:<code that means “end time”> // end time is same as start time since the action is

// a discrete, point-in-time action

Data: 2012-04-07T07:00Z

Qualifier Participant

Key:<code that means “participant”>

Data: <code that means “performer”> // who took the order off hold

Qualifier Role

Key:<code that means “role”>

Data: <code that means “physician”>

Qualifier IndividualPerson

Key:<code that means “individual person”>

Item PersonIdentifier

Key:<code that means “person identifier”>

Data: <identifier for the performer>

1. A designation is a text label representing a code in a controlled terminology. [↑](#footnote-ref-1)
2. This model is being discussed but is not yet created. [↑](#footnote-ref-2)
3. Figure 2 is an example written in the Constraint Definition Language (CDL). This document assumes some familiarity with CDL. For complete CDL information, the reader is referred to the Qualibria Constraint Definition Language (CDL) Language Guide. [↑](#footnote-ref-3)
4. “Attribute” (not to be confused with “attribution”) is a generic term used in this document to refer to a CEM’s item, qualifier, or modifier. A “coded attribute”, therefore, would be an item, qualifier, or modifier whose “data” component is a coded data type (CO or CD). [↑](#footnote-ref-4)
5. A Physical Quantity data type’s primary components are a numeric value and a unit of measure [↑](#footnote-ref-5)
6. To date, min and max constraints have not been used, but the functionality exists. [↑](#footnote-ref-6)
7. This statement is true for Qualibria, the system developed by the GE Healthcare-Intermountain Healthcare partnership. To the extent that the ECIS code system becomes used by other systems (e.g., other GE Healthcare systems), the ECIS code system may have additional purposes and hence additional content. [↑](#footnote-ref-7)
8. This physical separation is not to be interpreted as a logical separation. We strongly believe that authoring of models and authoing of terminology and value sets need to be performed in concert; performing either in a “silo” leads to mismatches in semantics. [↑](#footnote-ref-8)
9. A special “CEM display” context property on the designation distinguishes a designation that is used in a CEM from other designations a concept may possess. [↑](#footnote-ref-9)
10. More exactly, the value set contains the valid values for the “data” (or even more exactly, the “code” property of the “data”) of the AdministrativeMaritalStatus CEM. [↑](#footnote-ref-10)
11. In the Qualibria implementation, the data type is “CODE”, indicating that it is a UUID from the ECIS code system. [↑](#footnote-ref-11)
12. The data type of this code is CS, which is an “internal” data type (i.e., a data type used only within other data types). CS is essentially a string representing a code in an assumed code system. An implementation will predetermine the code system used to represent a CS component and that code system will be assumed and used for all instances of the component. [↑](#footnote-ref-12)
13. Data type “CODE”, which is a UUID in the ECIS code system [↑](#footnote-ref-13)
14. A “CDT” data type [↑](#footnote-ref-14)
15. How that ordering information is stored and retrieved is specific to an implementation, and an application will need to “understand” the implementation’s design in order to leverage the ordering information. [↑](#footnote-ref-15)
16. Data type “double” [↑](#footnote-ref-16)
17. An additional argument might be made for storing the numeric value in every instance -- even though the numeric value associated with a code is usually unchanging over time, there may be circumstances in which a change is necessary. The numeric value associated with code C1 may be X at t1 but Y at t2. In such cases, storing the numeric value with the instance of the data type preserves what the numeric value associated with the code was *at the time this instance was stored*. [↑](#footnote-ref-17)
18. How the numeric property of the code is stored in a particular terminology server is implementation-specific. The storing service or application must be aware of those implemenation’s details. [↑](#footnote-ref-18)
19. Data type “CODE”, which is a UUID in the ECIS code system [↑](#footnote-ref-19)
20. Data type “CODE”, which is a UUID in the ECIS code system [↑](#footnote-ref-20)
21. Data type “double” [↑](#footnote-ref-21)
22. Data type “INT” [↑](#footnote-ref-22)
23. Data type “CODE”, which is a UUID in the ECIS code system [↑](#footnote-ref-23)
24. When for an instance the operator is “=”, it is an implementation decision whether operator is left unpopulated or it must be explicitly populated with the code for “=”. [↑](#footnote-ref-24)
25. Data type “CODE”, which is a UUID in the ECIS code system [↑](#footnote-ref-25)
26. A “PQT” data type [↑](#footnote-ref-26)
27. Data type “CODE”, which is a UUID in the ECIS code system [↑](#footnote-ref-27)
28. A “STNT” data type [↑](#footnote-ref-28)
29. The implementation has latitude on two issues: 1) whether the data should be normalized to a particular timezone (and if so, which timezone – UTC would be the natural choice), or alternatively, if the local time should be stored. A compromise is to store the local time with its offset from UTC, e.g., “201205121800-0500” (meaning UTC – 5 hrs). For the Qualibria implementation, the value is expressed in UTC. 2) how the timestamp should be formatted -- YYYY-MM-DDTHH:mm:ss is the common XML standard, YYYY[MM[DD[HHmm[SS[.SSSS]]]]][zZZZ] is the standard chosen by HL7. [↑](#footnote-ref-29)
30. Data type “CODE”, which is a UUID in the ECIS code system [↑](#footnote-ref-30)
31. When for an instance the operator is “=”, it is an implementation decision whether operator is left unpopulated or it must be explicitly populated with the code for “=”. [↑](#footnote-ref-31)
32. CDL actually allows a file to contain multiple models, but by convention, each file represents a single model. [↑](#footnote-ref-32)
33. “domain” is being deprecated in favor of “value set”. It still appears in the CDL language, however, to denote value set constraints. [↑](#footnote-ref-33)
34. In this situation, if CD.code is not present, then CD.originalText should be required. However, there is currently no way in CDL to specify such a “co-occurrence” constraint. [↑](#footnote-ref-34)
35. Either dashes or double dots are allowed to express ranges. [↑](#footnote-ref-35)
36. A cardinality of “0” is used to express that this element is being constrained out. [↑](#footnote-ref-36)
37. More required elements might be expected in a model that restricts another model for a narrower use case. [↑](#footnote-ref-37)
38. As explained in the Abstract Constraint Model section, every line of a constraint model constrains some feature in the Abstract Instance Model or constrains an element in the current CEM or an element in a CEM referenced by the current CEM. In this sense, a CEM model (a constraint model) is actually not a “model” at all (in the sense of defining structure), but is in reality a list of constraints on a structural model. Certain constraints are declared so commonly in CEMs (key, data, qualifier, etc.) that CDL provides shorthand notation for them. The statements for key, data, and qualifiers/items/attributions/modifiers described previously are all essentially constraints, and could feasibly be written in the standard format for a constraint, but instead are written in a shorthand notation. [↑](#footnote-ref-38)
39. This type of constraint has been recognized as actually a processing instruction, and not a storage constraint. [↑](#footnote-ref-39)
40. This should not be confused with the key code; in the same HeartRateMeas CEM, the key code is a code representing “heart rate measurement”. The key code is the “real world” concept that represents the measurement of a heart rate (and that maps to a code in SNOMED or LOINC). In contrast, the type code is an identifier for the CEM logical model of a heart rate measurement. [↑](#footnote-ref-40)
41. If the semantics of the single qualifier are equivalent to those of the combination of multiple qualifiers, the new CEM and the existing CEM are designated members of the same iso-semantic family. [↑](#footnote-ref-41)
42. Ibid. [↑](#footnote-ref-42)
43. More exactly, the value set contains the valid values for the “code” property of the “data”. See explanation in Data Types and Properties. [↑](#footnote-ref-43)
44. See Data Types and Properties for further explanation. [↑](#footnote-ref-44)
45. A weak form of multiple inheritance [↑](#footnote-ref-45)
46. We have recognized that in the Qualibria implementation undertaken by Intermountain Healthcare and GE Healthcare, we tried to use reference classes both for metadata/implementation attributes and as an optimization mechanism – moving common attributes to the reference class so they could be query-optimized, even if they were part of the “logical” model. We intend to provide a cleaner separation of metadata, implementation and logical attributes in the future [↑](#footnote-ref-46)
47. We have recognized, however, that reference classes are use case-specific, i.e., different use cases call for, if not different reference classes and different hierarchies of reference classes, at least different contents of reference classes. We are in the process of modifying the CEM compiler to be able to use different sets of reference classes depending on the use case during the compilation process. [↑](#footnote-ref-47)
48. Whether or not a model is instantiable is an implementation issue, not a language issue, but the assumption is that any implementation will not allow instantiation of a partial model. [↑](#footnote-ref-48)
49. The CDL language actually permits a statement or component to contain multiple “data” components, but we have not found the use case for this and hence by convention adhere to the original CEM strategy of prohibiting multiple “data” components. If multiple “values” are required, “items” should be used. [↑](#footnote-ref-49)
50. i.e., a CEM that specializes the “component” reference class by declaring itself “is component” [↑](#footnote-ref-50)
51. The CDL language actually permits a statement or component to contain both data and an item or items, but we have not found the use case for this and hence by convention adhere to the original CEM strategy of requiring models to contain either data or item(s) but not both. [↑](#footnote-ref-51)
52. i.e., a CEM that specializes the “modifier” reference class by declaring itself “is modifier” [↑](#footnote-ref-52)
53. i.e., a CEM that specializes the “attribution” reference class by declaring itself “is attribution” [↑](#footnote-ref-53)
54. PQ = physical quantity, a data type that consists primarily of a numeric (real) value and a unit of measure. The unit of measure is captured by a code. See the CDL reference manual for more information on this and other data types. [↑](#footnote-ref-54)
55. CO = coded ordinal, a data type that consists primarily of a code that has an inherent ordering within a set of codes. An example of a component whose data type is CO is “severity”. The value set for severity includes codes for “mild”, “moderate”, and “severe”. The codes would have properties or relationships that capture the ordering among them, i.e., “mild” < “moderate” < “severe”. See the CDL reference manual for more information on this and other data types. [↑](#footnote-ref-55)
56. Even though the word “evaluation” might be a misnomer in these cases, because they fit the same pattern as the previous evaluations, the same word is used to describe them. [↑](#footnote-ref-56)
57. Another problem with “normal” is that the determination of “normalcy” might be use case-specific or institution-specific. Consequently, if a designation of “normal” is used, it should be provided in a specific context. [↑](#footnote-ref-57)
58. It can be confusing to one browsing the models to see what seems to be a fairly substantial clinical area modeled as a simple Impression Evaluation; it hopefully makes more sense once he realizes that all that is being captured is ”normal” versus “abnormal”. [↑](#footnote-ref-58)
59. Indeed, there are probably many disorders/conditions that have subtypes, so a value set for “data” is probably frequently appropriate. [↑](#footnote-ref-59)
60. The CDL language actually permits a statement or component to contain multiple “data” components, but we have not found the use case for this and hence by convention adhere to the original CEM strategy of prohibiting multiple “data” components. If multiple “values” are required, “items” should be used. [↑](#footnote-ref-60)
61. i.e., a CEM that specializes the “component” reference class by declaring itself “is component” [↑](#footnote-ref-61)
62. The CDL language actually permits a statement or component to contain both data and an item or items, but we have not found the use case for this and hence by convention adhere to the original CEM strategy of requiring models to contain either data or item(s) but not both. [↑](#footnote-ref-62)
63. i.e., a CEM that specializes the “modifier” reference class by declaring itself “is modifier” [↑](#footnote-ref-63)
64. More exactly, it extends PrescribingGuidance which is a statement [↑](#footnote-ref-64)
65. The CDL language actually permits a statement or component to contain multiple “data” components, but we have not found the use case for this and hence by convention adhere to the original CEM strategy of prohibiting multiple “data” components. If multiple “values” are required, “items” should be used. [↑](#footnote-ref-65)
66. i.e., a CEM that specializes the “component” reference class by declaring itself “is component” [↑](#footnote-ref-66)
67. The CDL language actually permits a statement or component to contain both data and an item or items, but we have not found the use case for this and hence by convention adhere to the original CEM strategy of requiring models to contain either data or item(s) but not both. [↑](#footnote-ref-67)
68. At present, there are no constraints on Participant.data, e.g., for an instance of “Observed”, Participants may be created with any participation type in “data”. Instead, the Observed model should specify which particular participant types are valid for Participation.data. [↑](#footnote-ref-68)
69. Presently, this is just a string (data type ST). We expect to evolve this to a location identifier. [↑](#footnote-ref-69)
70. Presently, this is just a string (data type ST). We expect to evolve this to a location identifier. [↑](#footnote-ref-70)
71. In panels and semantic links, the CEMs referred to by the items must constrain reference classes that are children of “Instance” (statement, patientEncounter, etc.). It is assumed, however, that a panel and semantic link will be implemented as a set of pointers to instances of those models. [↑](#footnote-ref-71)
72. Actually, an item may refer to any model that constrains any reference class that is a descendent of the “Instance” reference class (see the Inheritance via Reference classes section). Items that are statements or panels are by far the most common use case, though. [↑](#footnote-ref-72)
73. Or, less commonly, as other types of associations. [↑](#footnote-ref-73)
74. Note in Figure 16 that the items in the semantic link reference the HealthIssue CEM. One might conclude that the semantic link is a containment by value relationship rather than a containment by reference relationship, i.e., that the association actually contains the associated objects, rather than just containing pointers to those objects. That is not the case, however. The semantic link’s items are intended to be implemented as pointers to the referenced objects, not as containment of the objects themselves. [↑](#footnote-ref-74)
75. This does not preclude the individual contained instances from being retrieved separately, independent of the panel. It does also not preclude update of an individual instance, although an implementation may decide to update the containing panel whenever an individual instance is updated. [↑](#footnote-ref-75)
76. “laterality” is a misnomer. The qualifier initially just addressed laterality of the body location, but now addresses all spatial orientation. It will be replaced with a properly named qualifier in a future version of the BodyLocation component. [↑](#footnote-ref-76)
77. Whether this should instead be modeled as a semantic link to an instance of a sign and symptom is a much-discussed topic. Making an associated sign and symptom a statement linked to the the primary statement by a semantic link would support the notion of allowing only one way to store a particular piece of data. For example, “nausea” would be stored using the NauseaAssert statement model, whether it was the “primary” observation, or an associated symptom associated with a “primary” observation of “headache” (in which case it would be semantically linked to headache). On the other hand, it can be argued that making an associated and symptom a qualifier of another assertion model is a truer logical solution – the associated sign or symptom is thought of (logically) as a “part of” the primary statement. Its interpretation is within the context of the primary statement. It is true, however, that this approach forces “nausea”, for example, to be stored both as a statement (when it’s the primary observation) and as the value of a qualifier of another assertion (when it’s an associated symptom). An application searching for all instances of the patient’s nausea would have to look for instances of either “nausea” stored as the primary observation or “nausea” stored as the value of another assertion’s associated sign/symptom qualifier. However, we deem that to be not an onerous task. Furthermore, the query could be simplified to “find all instances of any model (statement or qualifier) where data = ‘nausea’” (since this would return both instances of the associated sign/symptom qualifier where data = ‘nausea’ as well as instances of the nausea assertion model). [↑](#footnote-ref-77)
78. Note that the DeltaFlag indicates that this piece of data represents a significant change from a previous value, where the previous value may or may not be accessible to the system. Use of the DeltaFlag qualifier does not require any reference or link to an actual instance of previous data. If a previous instance is available in the system, a semantic link may be created between the current instance and the previous instance to highlight and clarify the change. [↑](#footnote-ref-78)
79. At present, the lab models used by the Qualibria platform use the ReferenceRangeNar approach without the post-coordinated qualifier approach. They need to be migrated to the post-coordinated approach. [↑](#footnote-ref-79)
80. This model does not follow the convention of naming an attribution as a past tense verb. It should have been named, “TookOffHold”. [↑](#footnote-ref-80)
81. If the grandmother is known to the system, a patient Id for the grandmother may also be stored in “subject”. [↑](#footnote-ref-81)
82. The ramification is that an implementation of CEMs will need to support queries that filter out any instances where subject is populated, since the common query will be for data pertaining to the patient of record. [↑](#footnote-ref-82)
83. This pattern follows the conclusions of the HL7 TermInfo effort. [↑](#footnote-ref-83)
84. Verified is not presently in the order model, but should be added. [↑](#footnote-ref-84)
85. In this example, the containing model is an “order”. It is somewhat peculiar to have an “ordered” attribution on an “order”. The other option considered was to simply promote the attribution’s attributes (start time, end time, participant, etc.) up to the order level, resulting in Order.startTime, for example, instead of Order.ordered.startTime. For the sake of consistency, however, we decided to keep the attribution structure and have “Order” contain “ordered”. An analogous situation exists in a procedure model, which contains the attribution “performed”. We could have promoted “start time” up to the Procedure directly. This challenge occurs whenever the containing model is itself an action, as “order” and “procedure” are. Do we place another, somewhat artificial level of “action” (i.e., an attribution) inside the action, or do we place the who, when, and where information directly on the outer action? As described, we chose to insert the attribution for consistency with other models. [↑](#footnote-ref-85)
86. Verified is not presently in the order model, but should be added. [↑](#footnote-ref-86)
87. Note the name of this attribution, according to our convention of using a past tense verb, should really be “TookOffHold”. [↑](#footnote-ref-87)