$project.name

$DocumentTitle

Author: $Author

Revision: $Revisions.lastChild.name

|  |  |
| --- | --- |
| $CompanyName |  |
| $CompanyAddress | $date.get(“MMMMM dd, yyyy”) |

**Approval**

|  |
| --- |
| The original of this document is approved and signed by:  Name:  Surname:  Title:  Date:  Signature: |

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Description** | **Author** |
| #forrow($rev in $Revisions)  $rev.name | $rev.date | $rev.description | $rev.author #endrow |
|  |  |  |  |

| Table of Contents |
| --- |
| Microsoft Word users please click here and press F9 to create Table of Contents.  OpenOffice.org users please remove this text and select Insert Table of Content from menu. |

| Table of Figures |
| --- |
| Microsoft Word users please click here and press F9 to create Table of Contents.  OpenOffice.org users please remove this text and select Insert Table of Content from menu. |

# **Introduction**

## **Purpose**

$Purpose

## **Scope**

$Scope

## **Motivation**

[Stan to provide]

## **About the Model**

The CIMI model consists of structural patterns and constraints applied on those patterns:

1. The ***CIMI Reference Model*** specifies the classes, attributes, and allowed relationships that define the model’s primitive types, complex types, data structures, and clinical patterns.
2. The ***CIMI Archetype Library*** is composed of archetype hierarchies that progressively constrains the patterns defined in the reference model and whose leaf-level archetypes ultimately form the *Detained Clinical Model* layer of the CIMI model.

Each part of the CIMI model is persisted in a format that conforms to two OpenEHR specifications:

1. The ***Basic Meta Modeling (BMM)*** language is used to define and persist the CIMI Reference Model[[1]](#endnote-1).
2. The ***Archetype Definition Language (ADL2)*** is used to specify persistent and computable constraints on the reference model[[2]](#endnote-2).

The CIMI model follows a strict rule in the usage of the abovementioned two OpenEHR specifications in order to cleanly delineate the boundary between the reference model and the archetype hierarchies. *The basic meta-modeling language shall be used to specify the classes, attributes, and relationships that make up the model. The archetype definition language shall be used to define the constraints on the reference model* ***but shall not be used to define new model classes and attributes.*** In other words, the CIMI model specifies classes and attributes explicitly in the reference model and does not offer a way to extend the model within archetypes.

This approach represents a departure from the approach taken by OpenEHR which allows the definition of classes and attributes within archetypes using the metamodel constructs of ITEM, CLUSTER, and ELEMENT and by FHIR whose extension mechanism allows for the definition of new attributes and structures within resource profiles. The motivation for this decision is that, unlike FHIR and the OpenEHR models, both physical models, CIMI is a logical model and thus does not rely on extensions to provide for additional expressivity beyond that provided by the CIMI Reference Model. In other words, the CIMI Logical Model favors expressivity over economy of structure and delegates model extensibility to its underlying physical model target. For instance, attributes that exist in the CIMI Reference Model but that do not exist in the FHIR Core Model shall be mapped to the appropriate FHIR extensions.

### The CIMI Reference Model

The CIMI Reference Model is designed to be modular and currently consists of three layers:

1. The Core CIMI Reference Model defines the model’s primitives, its core types, and its core patterns, namely, the PARTY and PARTICIPATION patterns.
2. The CIMI Foundational Reference Model defines the foundational underpinnings of the CIMI model. This structure aligns with the ISO 13606 EHR and the OpenEHR Reference Models. It also defines three top-level hierarchies: CLUSTER, COMPOSITION, and CONTENT. It is from these hierarchies that all downstream CIMI classes are derived.
3. The CIMI Clinical Reference Model builds upon these two lower layers to provide the structural patterns upon which CIMI preferred archetypes are built.

This modular approach allows for additional, more domain-specific layers to be added in future or for alternate iso-semantic patterns to be introduced at the appropriate level in the model. Over time, it can be expected that the lower level reference model modules will become more stable while higher-level modules may undergo more flux.

#### What is a reference model or ‘clinical’ pattern

A *reference model pattern* is a structural pattern (a single class or group of classes) that can be constrained by archetypes in order to define a family of related and consistent models. The set of allowable clinical patterns comprise the CIMI Reference Model.

### The CIMI Archetype Hierarchies

The proposed CIMI archetype hierarchies form the second part of the CIMI model. These hierarchies serve two key purposes:

1. They enable the progressive application of constraints on reference clinical patterns including the specification of terminology constraints that assign formal meanings to both model attributes and their range.
2. They permit the definition of sets of models whose members vary solely on the constraints they apply to a common underlying reference model pattern.

Archetypes can specialize more general archetypes in ADL. They do so by progressively constraining the underlying reference pattern in a manner that is consistent with and not contradictory to the constraints specified in archetypes higher up in the hierarchy.

Examples of constraint refinements are listed below:

1. A top-level archetype restricts the range of Ingredient.substanceCode to the set of all concepts subsumed by the SNOMED CT concept ‘Pharmaceutical/biologic product’. A downstream specialization of this archetype restricts the Ingredient.substanceCode to ‘Metoprolol’.
2. A top-level archetype assigns the SNOMED CT concept ‘Procedure site (attribute)’ as the meaning of the attribute Procedure.site. A downstream specialization of this archetype further constrains this meaning to ‘Procedure site – Direct (attribute)’, a concept subsumed by ‘Procedure site (attribute)’.
3. A downstream archetype refines the cardinality of a container attribute from 0..\* to 2..5.
4. A downstream archetype *constrains out* a reference pattern attribute by setting its existence to 0..0.
5. A downstream archetype constrains the datatype of an attribute from DATA\_VALUE to QUANTITY.

Detailed Clinical Models (DCMs) typically reside at the bottom of CIMI archetype hierarchies. The cumulative constraints applied on a DCM are intended to be precise enough to allow for the unambiguous exchange of interoperable clinical information and thus constitute highly specific constraints on the underlying reference model.

## Modeling Principles

The following principles guide CIMI’s modelling approach:

1. CIMI favors a *design-by-specialization* over a *design-by-constraint* approach. This approach can be summarized as follows: if a class has a number of specializations, each requiring a different set of attributes, common attributes are represented in the parent class while children attributes are added to the appropriate specializations. An alternative approach may be to include the union of all attributes in a single class and constrain attributes out at the archetype level. The former approach is preferred over the latter except in certain cases. For instance, if a set of attributes is shared by some though not all specializations, they may *move up* to the parent class and be appropriately constrained out in archetypes. If a specialization differs from its parent by a single attribute, the inclusion of the attribute in the parent class *may* be preferred.
2. CIMI generally favors the definition of explicit attributes in the reference model over the *slicing* of lists in archetype definitions. The *attribute subset pattern* is achieved by defining a multi-cardinality attribute in the reference model and specifying subsets of the list elements in archetypes. For instance, one may specify that the LOCATABLE class, the supertype of all CIMI classes, has an attribute called *participation* of type PARTICIPATION and whose cardinality is 0..\*. In an archetype, one may then constrain the *participation* attribute in the following manner. The first element of the list represents the author. The second element represents the data enterer. The third element represents the location where the authoring activity took place. The fourth element of the list represents the system where the information was recorded. While such subsets are allowed in both UML and ADL, CIMI generally avoids their use and favors the explicit representations of such subsets as full-fledged attributes in the model. For instance, CIMI explicitly adds an attribute for the agent of an activity, the location of an activity, the entity involved in the performance of the activity, and so on. The motivation for this approach stems from the fact that CIMI is a logical model rather than a physical model and favors greater reference model expressivity over physical patterns that enable better economies of structure.
3. CIMI may offer a number of variants for a given attribute. For instance, CIMI defines bodyLocation: AnatomicalLocation and bodyLocationPrecoord: CODED\_TEXT in order to support both a coded and a post-coordinated anatomical location. Similarly Assertion.dueTo:CODED\_TEXT and Assertion.dueTo: ClinicalStatement allow users to link an assertion to another clinical statement or simply to a code.

## The CIMI Core Reference Model

The CIMI Core Reference model module introduces three core reference model hierarchies:

1. The *Any* hierarchy from which all CIMI primitive types are derived.
2. The DATA\_VALUE hierarchy from which all CIMI complex types are derived.
3. The LOCATABLE hierarchy from which all other CIMI classes are derived.

### CIMI primitives

We list below the set of CIMI primitive types and their mappings to UML and FHIR. In CIMI all primitive types derive from the Any abstract class:

|  |  |  |  |
| --- | --- | --- | --- |
| CIMI Primitives | UML Primitives | XML Primitives | FHIR Primitives |
| Boolean | Boolean | -- | boolean |
| Real | Real | -- | decimal |
| Byte | -- | byte | Byte |
| Integer | Integer | -- | integer |
| Character | Character | -- | -- |
| String | String | -- | string |
| URI | -- | anyUri | uri |
| List<T> | -- | -- | -- |
| Array<T> | -- | -- | -- |
| Any (root type) | -- | -- | Element |

### CIMI Complex Types

We list below the CIMI ‘complex’ data types. In CIMI, all complex data types derive from the DATA\_VALUE abstract class:

|  |  |  |
| --- | --- | --- |
| CIMI Data Types | FHIR Primitives | FHIR Complex Types |
| DATE | date | -- |
| TIME | time | -- |
| DATE\_TIME | dateTime | -- |
| COUNT | -- | Count |
| PROPORTION | -- | Ratio<Quantity> |
| QUANTITY | -- | Quantity |
| DURATION | -- | Duration |
| INTERVAL\_VALUE<T> | -- | Period, Range |
| PLAIN\_TEXT | string\* | -- |
| CODED\_TEXT | -- | Coding |
| URI\_VALUE | uri | -- |
| EHR\_URI | uri | -- |
| IDENTIFIER | -- | Identifier (partial match) |
| YESNO | boolean\* | -- |
| PARSABLE | string\* | -- |
| MULTIMEDIA | -- | Attachment (partial match) |
| ORDINAL | -- | -- |
| DATA\_VALUE (root type) | -- | Element |

\* *Extended FHIR primitive types*

CIMI and FHIR differ in the boundary between primitives and complex data types as shown in the table above. Moreover, a number of FHIR types do not have equivalents in CIMI. These include Annotation and SampledData, though both types could be accommodated within the CLUSTER hierarchy.

### The LOCATABLE Hierarchy

All classes in CIMI apart from CIMI primitives and complex types derive from the LOCATABLE class. The following sections will introduce some of the core reference model patterns found in CIMI.

### The PARTICIPATION\_RELATIONSHIP pattern

The CIMI Core Reference Model introduces the participation pattern. It defines PARTY, which may be either a ROLE or an ACTOR and a PARTY\_RELATIONSHIP to represent a relationship between two parties. Each class in the pattern has a type attribute to allow the specification of formal semantics at the archetype level. These classes also serve as the root types for specializations introduced in other reference modules.

## The CIMI Foundational Reference Model

The CIMI Foundational Reference Model introduces three top-level hierarchies:

1. The CLUSTER and VIRTUAL CLUSTER hierarchies
2. The COMPOSITION hierarchies
3. The CONTENT hierarchies

We describe these hierarchies below.

#### The CLUSTER/VIRTUAL\_CLUSTER hierarchies

The CLUSTER abstract class is the starting point for CIMI structures such as addresses, contact information, medications, and devices. Unlike CLUSTER, VIRTUAL\_CLUSTER allows for the grouping of attributes to support model component reuse and consistency but whose containment structure is not essential to the model and can be ignored by tools, editors, and code generation frameworks.

The CLUSTER hierarchy differs from the DATA\_VALUE hierarchy in that specializations of DATA\_VALUE represent a concise set of highly stable complex types (e.g., a quantity or a proportion) while specializations of CLUSTER are used to define the far more numerous and variable structures that are used to compose the reference model patterns (e.g., person names or international address structures).

#### The COMPOSITION hierarchy

At this time, the COMPOSITION hierarchy consists of a single class and can be used to represent clinical reports or patient records for instance. A composition is composed of sections and content entries.

#### The CONTENT hierarchy

The CONTENT hierarchy plays a special role in CIMI because it is the parent hierarchy for ClinicalStatements. The CONTENT class has the following subclasses: SECTION and ENTRY. ENTRY, in turn, has two specializations: COMPOUND\_ENTRY and INDIVISIBLE\_ENTRY. A COMPOUND\_ENTRY can be composed of other COMPOUND\_ENTRYs and INDIVISIBLE\_ENTRYs. ENTRYs represent units of standalone clinical information. An example may be a laboratory panel or a complex orderable.

## The CIMI Clinical Reference Model

The CIMI Clinical Reference Model introduces a number of structural pattern upon which all core archetypes are built.

### The CIMI Clinical Statement Pattern

The CIMI Clinical Statement Pattern forms the core of the CIMI model. The ClinicalStatement class is a specialization of the ENTRY class. It represents a statement about some aspect of a health care process. The CIMI ClinicalStatement pattern is composed of a statement topic and a situational context for that topic. The ClinicalStatement pattern also includes a reference to the subject of information and relevant provenance metadata for the information contained in a clinical statement.

StatementTopic and StatementContext are attribute groups (VIRTUAL\_CLUSTERs) which have the following characteristics:

1. They are reusable components, which can be assembled to form clinical statements. For instance, one can coordinate the Procedure statement topic with the Proposed statement context to represent a ProcedureProposal. The Procedure statement topic may also be paired with the Ordered statement context to create a ProcedureOrder statement.
2. They represent groupings of attributes that are aligned with the SNOMED CT Concept Model. For instance, the Procedure statement topic is aligned with the SNOMED CT Procedure Concept Model. The Performed context aligns with the Situation with Explicit Context Concept (SWEC) Concept Model.
3. They provide for a mechanism to state presence or absence of a finding as well as performance or non-performance of an action. For instance, the pairing of the Procedure topic with the NotPerformed context allows for the expression of a procedure that was not performed.

### The CIMI Attribution/Provenance patterns

In the CIMI model, provenance information is represented by the Attribution class. The Attribution class provides a pattern for the capture of provenance information such as the *what, who, when, where, why*, and *how* associated with a particular activity – e.g., provenance attributes about the verification of a clinical statement.

CIMI proposes three attribution patterns:

1. *Attribution information as a part of the clinical statement* – In this pattern, the ClinicalStatement pattern contains a number of attributes of type *Attribution* (e.g., ClinicalStatement.authored and ClinicalStatement.verified). This pattern provides a consistent way to capture attribution information that extends beyond simply the agent of an activity (e.g., the author). When attribution is part of the ClinicalStatement model, any change to the attribution for an activity will result in a version change.
2. *Attribution information that is external to the clinical statement* - CIMI also allows the capture of provenance information that is external to the clinical statement through the *Provenance* class. The provenance class includes the Attribution class and pointers to one or more clinical statements (the *Provenance.target* attribute). This pattern allows the attachment of provenance information to a clinical statement without impacting its version.

### The CIMI Assertion and EvaluationResult Pattern

[To be provided by Susan]

### The CIMI Procedure pattern

[TBD]

### An Introduction to CIMI archetypes

### Detailed Clinical Models – The Skin and Wound Assessment Archetypes

[To be provided by Jay and Susan]

### CIMI and FHIR

The CIMI logical model and FHIR are complementary models. Unlike FHIR, CIMI does not provide a specification for the representation of instances of clinical data. Rather, it relies on physical models such as FHIR to do so.

In order to achieve this, the CIMI preferred models must first be transformed into FHIR resource profiles. The CIMI logical model provides the underlying specification for such profiles through the translation of both reference model structures and archetype constraints into their equivalent representations in corresponding FHIR structure definitions. Once the set of CIMI-FHIR profiles has been generated, FHIR instances conformant with these CIMI profiles represent the set of conformant CIMI instances.

In order to reduce the transformation burden associated with such translations, CIMI aims to align the granularity of its models with those specified by FHIR. Given their differing requirements, CIMI models and FHIR resources may differ in their structure. For instance, while CIMI clinical statements are compositional structures, FHIR resources tend to be flatter. However, CIMI shall aim to (1) provide declarative and computable transformations from all of its models to FHIR profiles and (2) minimize the cost of such transformations by keeping both models aligned.

At a high level, the transformation of CIMI models into FHIR profile can be achieved as follows:

1. CIMI archetypes must first be aligned with the corresponding FHIR resources
2. Reference model attributes are mapped to attributes in the mapped FHIR resource(s)
3. CIMI attributes that have no equivalent in FHIR are handled as FHIR extensions
4. ADL model constraints are then translated into their corresponding representations in FHIR structure definitions

Archetypes in CIMI are defined hierarchically with more specific archetypes further tightening the constraints of their ancestors in the hierarchy. In order to retain this structure in FHIR, the translation of CIMI archetype hierarchies into FHIR resource profiles will most likely result in the generation of layered resource profiles in FHIR that mimic the structure found in CIMI.

### Future work

While much effort has gone into this phase of the CIMI model development and in the preparation of the January 2017 ballot material, the CIMI Working Group recognizes that much still remains to be done. In the months following the January HL7 Working Group Meeting, the CIMI Team intends to address community comments and refactor the model accordingly. By the May 2017 HL7 Working Group Meeting, our goal is to complete the CIMI Reference Model, CIMI top-level archetypes including archetypes that capture US Core and QI Core requirements, and ballot the model as an Informative Specification. By the September 2017 HL7 Working Group Meeting, we intend to complete the terminology bindings for all proposed archetypes, implement detailed clinical models for selected use cases, and specify formal declarative mappings from CIMI to FHIR using the FHIR Mapping Language. The model will then be balloted as a Standard for Trial Use.

### Request for Comments

We encourage the community to comment on any aspect of the proposed model. In particular, we would like to solicit comments and feedback in the following areas:

* Gaps, corrections, or enhancements to the proposed reference model classes and archetypes including any proposed terminology bindings
* The Clinical Statement Pattern
* The Provenance patterns
* The EvaluationResult/Assertion patterns
* The Procedure Pattern
* The proposed alignment with the SNOMED CT concept model, in particular, alignment with the Situation with Explicit Context Concept Model, the Observable model, the Clinical Finding Concept Model, and the Procedure Concept Model
* Alignment of information model attributes with other ontologies
* Alignment of the CIMI model with FHIR and with the Federal Health Information Model (FHIM)
* The CIMI Style Guide
* The CIMI Modeling Principles

## **Overview**

$Overview

#macro (removeLineBreak $s)

#set ($displayedName = “”)

#foreach ($c in $s.toCharArray())

#if($c.getType($c)==15)

#set($displayedName = $displayedName.concat(“ “))

#else

#set($displayedName = $displayedName.concat($c.toString()))

#end

#end

#end

#set($packages = $array.createArray())

#set($void = $array.addCollection($packages, $Package))

#set($void = $array.addCollection($packages, $SmartPackage))

#foreach ($package in $sorter.sort($packages, “qualifiedName”))

#set($qualifiedName = $package.qualifiedName)

# **Package $qualifiedName**

$package.documentation

#if ($package.hasImage())

$package.image

1. $package.name

#end

|  |  |
| --- | --- |
| **Name** | $package.name |
| **Qualified Name** | $qualifiedName |

#set ($classList = $array.createArray())

#set ($interfaceList = $array.createArray())

#set ($enumerationList = $array.createArray())

#foreach ($element in $report.getOwnedElementsIncludingAdditional($package, true))

#if ($element.elementType == “class”)

#set ($void = $classList.add($element))

#elseif ($element.elementType == “interface”)

#set ($void = $interfaceList.add($element))

#elseif ($element.elementType == “enumeration”)

#set ($void = $enumerationList.add($element))

#end

#end

#foreach ($class in $sorter.sort($classList))

## **Class $bookmark.create($class.ID, $class.name)**

$class.documentation

#if ($class.hasImage())

$class.image

1. $class.name

#end

|  |  |
| --- | --- |
| **Name** | $class.name |
| **Qualified Name** | $class.qualifiedName |
| **Visibility** | $class.visibility |
| **Abstract** | $class.isAbstract |
| **Base Classifier** | #if ($class.hasGeneralization())  #set($genList=$array.createArray())  #foreach ($general in $class.generalization)  #set($tmp=$genList.add($general.general))  #end  #foreach($gen in $sorter.sort($genList))   * $bookmark.open($gen.ID, $gen.name)   #end  #end |
| **Realized Interface** | #if ($class.hasInterfaceRealization())  #set($realList=$array.createArray())  #foreach ($realize in $class.interfaceRealization)  #set($tmp=$realList.add($realize.contract))  #end  #foreach($real in $sorter.sort($realList))   * $bookmark.open($real.ID, $real.name)   #end  #end |

#if ($class.hasOwnedAttribute())

### **Attribute Detail**

#foreach ($attr in $sorter.sort($class.ownedAttribute))

#### $attr.name

$attr.documentation

|  |  |
| --- | --- |
| **Type** | $attr.type.name |
| **Default Value** | #if($attr.defaultValue && $attr.defaultValue.hasText())  $attr.defaultValue.text  #end |
| **Visibility** | $attr.visibility |
| **Multiplicity** | $attr.multiplicity |
| **Definition** | $attr.documentation |

#end

#end

#if ($class.hasOwnedOperation())

### **Operation Detail**

#foreach ($ope in $sorter.sort($class.ownedOperation))

#### $ope.name

$ope.documentation

|  |  |
| --- | --- |
| **Type** | $ope.type.name |
| **Visibility** | $ope.visibility |
| **Is Abstract** | $ope.isAbstract |
| **Parameter** | #set ($paraList = $array.createArray())  #foreach($para in $sorter.sort($ope.ownedParameter))  #if($para.direction!=”return”)  #set ($void = $paraList.add($para))  #end  #end  #foreach($para in $sorter.sort($paraList))  #set($paraName = “$para.direction $para.name : $para.type.name”)  #if($para.multiplicity!=””)  #set($paraName = “$paraName [$para.multiplicity]”)  #end  #if($para.defaultValue && $para.defaultValue.hasText())  #set($paraName = $para.defaultValue.text)  #end   * $paraName   #end |

#end

#end

#set($relationList = $report.getRelationship($class))

$group.init()

#foreach ($relation in $relationList)

#set($void = $group.put($relation.humanType, $relation))

#end

#if (!$relationList.isEmpty())

### **Relation Detail**

#foreach ($key in $sorter.sort($group.groupNames()))

#### $key

#foreach($rel in $sorter.sort($group.get($key)))

|  |  |
| --- | --- |
| **Name** | #if($rel.hasName())  $rel.name#end |
| **Related Element** | #foreach ($re in $sorter.sort($rel.relatedElement))  #if($re != $class)   * $re.name   #end  #end |

#end

#end

#end

#end

#foreach ($interface in $sorter.sort($interfaceList))

## **Interface $bookmark.create($interface.ID, $interface.name)**

$interface.documentation

#if ($interface.hasImage())

$interface.image

1. $interface.name

#end

|  |  |
| --- | --- |
| **Name** | $interface.name |
| **Qualified Name** | $interface.qualifiedName |
| **Visibility** | $interface.visibility |
| **Base Classifier** | #if ($interface.hasGeneralization())  #set($genList=$array.createArray())  #foreach ($general in $interface.generalization)  #set($tmp=$genList.add($general.general))  #end  #foreach($gen in $sorter.sort($genList))   * $bookmark.open($gen.ID, $gen.name)   #end  #end |

#if ($interface.hasOwnedAttribute())

### **Attribute Detail**

#foreach ($attr in $sorter.sort($interface.ownedAttribute))

#### $attr.name

$attr.documentation

|  |  |
| --- | --- |
| **Type** | $attr.type.name |
| **Default Value** | #if($attr.defaultValue && $attr.defaultValue.hasText())  $attr.defaultValue.text  #end |
| **Visibility** | $attr.visibility |
| **Multiplicity** | $attr.multiplicity |

#end

#end

#if ($interface.hasOwnedOperation())

### **Operation Detail**

#foreach ($ope in $sorter.sort($interface.ownedOperation))

#### $ope.name

**Description**

$ope.documentation

|  |  |
| --- | --- |
| **Type** | $ope.type.name |
| **Visibility** | $ope.visibility |
| **Is Abstract** | $ope.isAbstract |
| **Parameter** | #set ($paraList = $array.createArray())  #foreach($para in $sorter.sort($ope.ownedParameter))  #if($para.direction!=”return”)  #set ($void = $paraList.add($para))  #end  #end  #foreach($para in $sorter.sort($paraList))  #set($paraName = “$para.direction $para.name : $para.type.name”)  #if($para.multiplicity!=””)  #set($paraName = “$paraName [$para.multiplicity]”)  #end  #if($para.defaultValue && $para.defaultValue.hasText())  #set($paraName = $paraName.concat(“=”).concat($para.defaultValue.text))  #end   * $paraName   #end |

#end

#end

#set($relationList = $report.getRelationship($interface))

$group.init()

#foreach ($relation in $relationList)$group.put($relation.humanType, $relation)#end

#if (!$relationList.isEmpty())

### **Relation Detail**

#foreach ($key in $sorter.sort($group.groupNames()))

#### $key

#foreach($rel in $sorter.sort($group.get($key)))

|  |  |
| --- | --- |
| **Name** | #if($rel.hasName())  $rel.name#end |
| **Related Element** | #foreach ($re in $sorter.sort($rel.relatedElement))  #if($re != $interface)   * $re.name   #end  #end |

#end

#end

#end

#end

#foreach ($enum in $sorter.sort($enumerationList))

## **Enumeration $bookmark.create($enum.ID, $enum.name)**

$enum.documentation

#if ($enum.hasImage())

$enum.image

1. $enum.name

#end

|  |  |
| --- | --- |
| **Name** | $enum.name |
| **Qualified Name** | $enum.qualifiedName |
| **Visibility** | $enum.visibility |
| **Base Classifier** | #foreach ($classifier in $sorter.sort($enum.baseClassifier))   * $bookmark.open($classifier.ID, $classifier.name)   #end |

#set($relationList = $report.getRelationship($enum))

$group.init()

#foreach ($relation in $relationList)$group.put($relation.humanType, $relation)#end

#if (!$relationList.isEmpty())

### **Relation Detail**

#foreach ($key in $sorter.sort($group.groupNames()))

#### $key

#foreach($rel in $sorter.sort($group.get($key)))

|  |  |
| --- | --- |
| **Name** | #if($rel.hasName())  $rel.name#end |
| **Related Element** | #foreach ($re in $sorter.sort($rel.relatedElement))  #if($re != $enum)   * $re.name   #end  #end |

#end

#end

#end

#end

#end

# **Appendix A: Diagram**

#foreach ($diagram in $sorter.sort($Diagram))

#if ($diagram.diagramType == “Class Diagram”)

#removeLineBreak($diagram.name)

## **$displayedName**

$diagram.documentation

$diagram.image

1. $diagram.name

#end

#end

1. http://www.openehr.org/releases/BASE/latest/docs/bmm/bmm.html [↑](#endnote-ref-1)
2. http://www.openehr.org/releases/AM/latest/docs/ADL2/ADL2.html [↑](#endnote-ref-2)