Core Principles and Properties of HL7 Version 3 Models

HL7 Normative Ballot

HL7 V3 MODELS, R1
HL7 Version 3 Standard: Core Principles and Properties of Version 3 Models, Release 1
Normative Ballot 1 - May 2008

Primary Contributer and Infrastructure & Management Co-Chair	Grahame Grieve Kestral Computing Pty. Ltd.
Primary Contributer and Vocabulary Co-Chair	W. Ted Klein Klein Consulting, Inc.
Primary Contributer and Modeling & Methodology Co-Chair	George Beeler, Jr., PhD. Beeler Consulting LLC
Primary Contributor, Vocabulary WG member	Dr. Jobst Landgrebe, ii4sm
Primary Contributor	Harold Solbrig, Mayo Clinic
Primary Contributor	Alan Honey, ii4sm
Modeling & Methodology Co-Chair	Lloyd McKenzie McKenzie Consulting
Modeling & Methodology Co-Chair	Dale Nelson Zed Logic, Inc.
Modeling & Methodology Co-Chair	Craig Parker, MD RemedyMD, Inc
Modeling & Methodology Co-Chair	Ioana Singureanu U.S. Department of Veterans Affairs

Last Published: XXX

Table of Contents

1.The Rationale for V3: Semantic interoperability

(MnM)

2.Models: RIM and its derivatives.

(MnM, Refer to HDF, RIM)

2.1. Instances

The fundamental notion of V3 is that in order to exchange data, systems exchange serialised streams of data that are an "instance" of a V3 model under a set of rules that describe why and when the data is exchanged.

All V3 models are valid of classes linked by associations. The classes and assocations are defined in the RIM. The classes have a series of named attributes which are assigned a type defined in the datatypes. Some attributes are associated with controlled vocabularies which provide clearly defined semantic meaning to the static models. Together, the structural vocabularies, the data types and the RIM classes constitute the reference model.

All V3 instances are instances of the reference model, and conform with the rules of the reference model. V3 models usually also conform to other additional constraint models that describe how the general reference model is used to describe particular administrative or clinical healthcare information.

Instance of V3 models may have any forms of expression and be used in many contexts, such as a message payload in a message associated with an HL7 defined interaction, a CDA document, or a payload as part of a service interaction, etc.

Instances of V3 models are exchanged in the context of a dynamic model that specifies why and when the data is exchanged.

Dynamic models are discussed further below.

2.1.1. Serialization

In order for systems to exchange the instances of a V3 model, they need some particular form of representation. As a response to industry demand, HL7 offers a defined representation of the V3 instances in XML, known as the XML ITS. Other forms of representation could be imagined, such as XMI, HUTN, ASN.1 and so forth, but there has not been sufficient demand to justify the creation of alternatives to the standard XML form.

The ITS must define not only how the instance is serialised, but also how the links from the instance to the many models that contribute to defining the meaning of the instance are expressed and/or derived from the serialized representation.

2.2. RIM: Reference Information Model

The RIM defines all the classes that are used in V3 instances. The RIM itself is a UML class model; the classes are standard classes in the UML sense, and have associations and attributes as defined in the RIM models. All classes defined in the RIM are specialisations of the base class InfrastructureRoot which defines the attributes that support the core behaviour of V3 models as described in this specification.

The RIM makes extensive use of the two other parts of the V3 reference model, the structured vocabulary and the datatypes. Vocabulary and it's associated concepts are discussed below.

The datatypes define the set of types that may be used to define the value domains and associated semantics of the RIM class attribute. The semantics of the types are defined in the the abstract datatypes (the ITS describes how the datatypes are actually represented when serialized).

Known Issue 02 (§ 2.2)

2.2.1. How the RIM is maintained

Known Issue 03 (§ 2.3)

2.3. Other V3 Static Models

All other V3 Static Models are statements of constraint against the RIM to which instances of the V3 reference model may be required to conform in a particular context of use.

These models are expressed using a modeling formalism and language developed by HL7 for the purpose. This is fully defined in the **HL7 Development Framework.** (HDF) and the **HL7 Model Interchange Format** (MIF). However this is only one possible form of expression. Other forms of expression have been imagined and proposed or are under development.

Static models may be considered or represented as a direct statement of constraint or as a class model (UML) or some form of typing model (i.e. schema) in their own right. The difference in these two is largely an implementation issue; the semantics are always clear: all instances are instances of the reference model, and all other satic models are constraints on the reference model. The degree of success at representing a static model as a typing model depends on the target platform.

2.3.1. DIM : Domain Information Model

The first level of constraint is a domain information model. This provides a solution to the information requirements of a particular problem domain. A DIM may have multiple entry points. As such, a DIM is not a directly implementable model, and is a fairly general statement of an domain with fairly general vocabulary bindings.

Known Issue 05 (§ 2.5)

2.3.2. CIM : Constrained Information Model

CIMs represent a second level of constraint. CIMs must have single entry points, which makes them serialisable. CIMs are therefore suitable for use as implementation constructs on information systems and should be completely specified for this purpose. CIMs are generally focused on narrower problem domain that a DIM.

CIMs are either derived from a DIM directly, or from another CIM. Though technically, a CIM could also be derived from the RIM directly, this is prohibited as a matter of policy to encourage consistent design.

CIM cascades can be as deep as desired, but in most domains HL7 only defines a DIM and one layer of CIMs.

Known Issue 06 (§ 2.6)

2.3.3. LIM : Local Information Model

Like CIMs, LIMs are a constraint model that has a single entry point. However LIMs differ from CIMs:

- LIMs may be derived from the RIM directly as well as DIMs or CIMs (though derivation from the appropriate DIM is recommended)
- LIMs may be defined and published by anyone (including HL7 and it's affiliates).
- LIMs may be incomplete models (refer to the static model definitions for further information about incomplete models).

LIMs are principally intended to be used as templates, but may be used in other fashion by site or realm agreement.

2.4. Dynamic Models

In the past, the HL7 work on dynamic models was an attempt to describe the behavioral semantics of messaging. HL7 now deals with this set of problems in the Services Aware Enterprise Architecture Framework (SAEAF) and the Behavioral Framework (BF), both published and maintained by the Architecture Board.

3. Types as domains for model classes and attributes [chapter title to be adjusted]

(MnM)

3.1. How RIM and datatypes fit together (was 14)

3.2. Datatype Flavors (was 5.2)

For datatypes, the type must be the type from the reference model; the expressed model is always that specified in the abstract datatypes. This policy exists to ensure that implementations of the datatypes are robust for use in all the environments that V3 is used.

Datatypes may also have additional constraints associated with them. These constraints are referred to as datatype flavors. Datatype flavors are very similar to applied models, but only one flavor can be specified.

Reference: The Refinement, Constraint and Localisation should be consulted for further information about datatypes flavors.

3.3. Null Flavor (was 6)

It is common to encounter missing or incomplete information in healthcare. In some circumstances, why, how, or in what way the information is missing or incomplete may have some semantic significance that may make a difference to the workflow or clinical management the depends on the information.

For this reason all datatypes and RIM classes have a property called "nullFlavor" which specifies why the information does not exist, is not known or available, or cannot be expressed in the allowed value domain.

This table summarises the currently accepted values that the nullFlavor property may have it is not null:

Table 3: Domain NullFlavor (OID: 2.16.840.1.113883.11.10609, Source: Internal)

lvl	code	name	Definition
1	NI	no information	The value is exceptional (missing, incomplete, improper). No information as to the reason for being an exceptional value is provided. This is the most general exceptional value. It is also the default exceptional value.

Table 3: Domain NullFlavor (OID: 2.16.840.1.113883.11.10609, Source: Internal)			
lvl	**code**	**name**	**Definition**
2	INV	invalid	The value as represented in the instance is not an element in the constrained value domain of a variable.
3	OTH	other	The actual value is not an element in the constrained value domain of a variable. (e.g., concept not provided by required code system).
4	NINF	negative infinity	Negative infinity of numbers.
4	PINF	positive infinity	Positive infinity of numbers.
3	UNC	unencoded	No attempt has been made to encode the information correctly but the raw source information is represented (usually in originalText).
3	DER	derived	An actual value may exist, but it must be derived from the provided information (usually an expression is provided directly).
2	UNK	unknown	A proper value is applicable, but not known.
3	ASKU	asked but unknown	Information was sought but not found (e.g., patient was asked but didn't know)
4	NAV	temporarily unavailable	Information is not available at this time but it is expected that it will be available later.
3	QS	sufficient quantity	The specific quantity is not known, but is known to be non-zero and is not specified because it makes up the bulk of the material.'Add 10mg of ingredient X, 50mg of ingredient Y, and sufficient quantity of water to 100mL.' The null flavor would be used to express the quantity of water.
3	NASK	not asked	This information has not been sought (e.g., patient was not asked)
3	TRC	trace	The content is greater than zero, but too small to be quantified.
2	MSK	masked	There is information on this item available but it has not been provided by the sender due to security, privacy or other reasons. There may be an alternate mechanism for gaining access to

Table 3: Domain NullFlavor (OID: 2.16.840.1.113883.11.10609, Source: Internal)			
lvl	**code**	**name**	**Definition**
			this information.Note: using this null flavor does provide information that may be a breach of confidentiality, even though no detail data is provided. Its primary purpose is for those circumstances where it is necessary to inform the receiver that the information does exist without providing any detail.
2	NA	not applicable	No proper value is applicable in this context (e.g., last menstrual period for a male).

A datatype or a class is known as a "null" class if it has a value for it's nullFlavor property. Null values are also known as "exceptional values". Null values are improper values that do not conform to the proper or expected value domain as described by the applicable specification (usually any model that the type claims conformance too - see typing below). The information may either be missing or partially present, or even completely present but not valid with respect to the constraints imposed by the models it conforms to. While null values may not conform to the "proper or expected value domain" as described by the specification, they must nevertheless conform to all the rules specified by the specifications to which they conform, null values SHALL only be used as specified by the models, both in regard to where and how they are used.

In this sense, null is used to create a two level conformance strategy. In some cases, a properly acceptable value domain is defined, and only information that completely conforms to the specified value domain may be provided. In other cases, a properly acceptable value domain is defined, and some information must be provided, but it may not conform to the narrow value domain if it explicitly declares that it does not conform. See the conformance section for further details.

3.3.1. Note about the name nullFlavor

The property is named nullFlavor because of the similarities between the concept of a null value and the concept and behaviour of null in implementation technologies, particularly SQL and OCL. As in SQL and OCL, the value null is in the value domain of the all the types, and nulFlavors will generally propagate through operations such as comparison (i.e. the result of a comparison operation between a null value and some other value is null).

However there are some important differences between the implementation of nulls in such technologies and the HL7 nullFlavor. Most notably, in most implementation technologies, a null instance has no further information associated with it (some variation of the concept of a null pointer). This is not true of the HL7 concept of null; if a datatype or class is null, the nullFlavor property is not null, and any of the other properties might not be null.

Note: the nullFlavor property functions in a reverse sense to the data type or class; if the value is not null the nullFlavor will be null, and if the value is null, then the nullFlavor is not null - it will specify an actual nullFlavor that provides more detail as to in what way or why no proper value is supplied.

Note: In OCL, null is an instance of OclVoid which is a super type of all types. nullFlavor is not modelled the same way in HL7: a null value is still a valid instance of a particular type (see types below). If a true null is encountered in an implementation environment (i.e. the class is not represented in the XML when using the XML ITS, or is present with an xsi:nil="true" attribute), it is semantically equivalent to a null-value of NI, and all other properties not related to nullFlavor will also have nullFlavor NI.

3.3.2. Implementation Considerations

When performing operations upon null values, the semantic meaning of the nullFlavor SHALL be considered. This is particularly important for equality. The only case where non-proper (NULL) values may be equal is where both values have a nullFlavor of NA and all other properties equal. In all most other cases, the outcome of comparing NULL values is also null. However, there are exceptions based on the semantic meaning of nullFlavor. For instance, in the datatypes, although direct comparison of two values with nullFlavor PINF is always null (NI), two intervals with the equal low bounds and high bounds of PINF will return true, since they specify the same set. Similarly, comparison of NINF and PINF is always False.

The "actual value" refers to the value of the information itself, rather than the information as represented in the type itself. These two may diverge when the information provided is incomplete, such as when an expression is provided. The null flavor "other" is used whenever the actual value is not in the required value domain: this may occur, for example, when the value exceeds some constraints that are defined in too restrictive a manner. For example, if the value for age is 100 yrs, but the constraining model specifies that the age must be less than 100 years, the age may still be specified, provided that the model does not make the attribute mandatory.

```
<value nullFlavor="OTH" value="120" unit="yr"/>
```

Some of the null flavors are not generally applicable to all circumstances. The nullFlavors NINF, PINF, QS, and TRC SHALL only be used in associated with datatypes that are a specialisation of the QTY type. The nullFlavor UNC SHALL only be used with any data type that has an originalText, and when UNC is used the originalText property SHALL be populated. The nullFlavor "DER" SHALL only be used with the EXPR type, and an expression SHALL be provided.

Note: NULL-flavors are potentially applicable to any class, any data type, and any property of a data value. Where the difference of null flavors is not semantically significant, ITS are not required to represent them. (this is usually appropriate for structural attributes in the RIM classes, and simple properties of the datatypes).

3.4. Identifying classes and data types in instances (was 5)

A type is a class or a datatype.

All HL7 models are constraints on a reference model built from the classes defined in the RIM and the datatypes defined in the abstract datatypes.

This reference model is further constrained by additional constraint models that associate new names for particular constraints on the associations and classes. These constraint models may come from a linear sequence of constraints where each model is an additional constraint on another model (and when the instance conforms to a model it also conforms to the models on which that model is derived), or an instance may conform to multiple different constraints that are not related to each other.

So any given type is an instance of the class or datatype as specified in the reference model, while at the same time conforming to multiple other different design specifications within this cascading hierarchy of models.

Every class and data type SHALL declare conformance to a single master type. This requirement exists to ease the path of implementations in common target technologies. The type as a duple: the name of the model, and the name of the type/constraint definition in the model. Both the name of the model and the name of the type may be defined by some applicable design contract rather than expressed directly as an attribute of the class.

ITSs that describe how to represent V3 models SHALL make clear how the both parts of the type may be determined from examination of the instance, and what other resources are required at design and/or runtime to unambiguously resolve the type of the class or datatype.

Note: The InfrastructureRoot class in the RIM defines the notional attribute typeId to represent the type of the class. ITSs are not required to represent this attribute directly; some other method of representation may be chosen that is more appropriate with the base technology and consistent with the way the ITS specifies that the type information is determined from the instance.

3.4.1. Model Types for Classes

For classes, the type need not be the type from a reference model; the context may specify that the expressed type is a name taken from one of the applicable constraining model. As a consequence, there are three types of models applicable to classes:

3.4.1.1. Expressed Models

The expressed model is the model that contains the type expressed by the class.

Note: The existing XML ITS fixes the expressed model throughout the instance to be the static model associated with the interaction identifier specified in the root element of the interaction (or from "ClinicalDocument" for CDA). The type of a class is not usually represented directly; instead the names of the associations in the expressed model are used, and the type is determined by implication from the association name. For choices, elements of the type name may be pre-coordinated with the association name in the instance.

Note: Only complete, implementable models with one entry point (CIMs and some LIMs) may be used as expressed models.

3.4.1.2. Implied Models

The implied models are specified by the derivations contained in the definition of the expressed model. All expressed models SHALL specify derivations from the RIM. Additional derivations from other models may also be specified.

Note: this means that the RIM is always an expressed or applied model.

Implementation Note: A processor can correlate the instance data against an implied model by reading the full static model for the expressed model and tracing the derivations from the expressed model to the implied model of choice. This can also be done by the developer by hard coding the derivations in the application. HL7 XML ITS schemas also provide a partial link to the RIM level definition. The implied RIM model is of such consequence that a separate pattern for identifying the RIM classes in the instance exists, using structural codes.

3.4.1.3. Applied Models

Are other models to which the class conforms to but are not explicit or implicit in the type the the class conforms to. These models are usually known as templates. The applied model may be invoked explicitly in in the instance, or by specifying it in some form of design contract (e.g. interaction profile). Note that it is not necessary to declare all the constraint models that a class conforms to.

Note: The InfrastructureRoot class defines an attribute called templateId which is used to represent the set of applied models that a class conforms to. Like the typeId attribute, the templateId is notional; ITSs may define alternate methods for representation of the applied models.

ITSs that describe how to represent V3 models SHALL make clear how the applied models may be determined from examination of the instance, and what other resources are required at design and/or runtime to unambiguously resolve the applied models.

Note: If an applied model specifies derivations, then the models specified in the derivations are also implied models.

Reference: The templates specification should be consulted for further template related information.

4. Identifying elements

MnM/Vocab (OIDs, IIs, etc.)

One of the founding principles of V3 models is the importance of the correct identification of classes and objects.

4.1. Referencing Objects

When source and destination systems share sufficient information to permit it, the source system may simply refer to an object rather than providing full details of the object. Rather than updating the object in either snapshot or update mode, the destination system should use the information provided to identify an existing instance of data.

It is not necessary for the destination system to already have information, only for the system or the appropriate users to know how to locate the information that the reference pertains to.

For this reason, object references are used more widely than update mode. Nevertheless, the concept of reference is tightly related to the concept of update mode - an object will either be passed in as a snapshot, an update, or a reference.

Although complex scenarios involving combinations of these modes can be envisaged, HL7 does not support combining modes in order to reduce processing complexity. If an object is passed as a reference, there shall be no expectation that any updates to the object may occur. If an object is represented using update Mode, any information provided as part of the object that has no associated update instructions is ignored.

4.2. Identifying Objects (was 9)

Whether an object is being conveyed using snapshot mode, update mode, or as a reference, the first step for most processing systems is to correctly locate an existing record for the concept that the object represents, if one exists.

In order to accomplish this, the system must correctly identify the object. In most cases, the identification will be implicit or explicit in the contracts that control the system communication. However in some cases it will be necessary for the source system to clearly indicate the attributes that should be used to identify the object.

For example, a source system may wish to indicate which of several identifiers associated with an object should be used to identify the object. The semantic properties of the identifiers - scope and reliability - are generally preferred as the criteria for choosing which identifier should be used, but in some cases it may be necessary to specify a particular attribute.

Another case is where the source system does not know the relevant identifiers for the object, but is able to define some key criterion for identification of the concept. For instance, the source system may know that the patient had an episode of care on a given date, but not the destination system's identifier assigned to the episode of care.

A source system should clearly identify the attributes of an object that it expects to be used to identify the object correctly, if it does not assume that identifier scope and reliability suffice.

The general implication of these rules is that when an object is sent using update mode or as a reference, only the information that is required in order to correctly identify the object is sent, along with any specific updates for update mode, and that all the information provided should be clearly labeled. However it isn't always clear how much information is required to correctly identify the reference, so additional useful information is always allowed. Generally, it would be expected that this additional information would be of use in some human intervention procedure if automated resolution of the reference failed.

Data types are not subject to identification - the full value of the datatype is itself the identity of the value.

In the absence of any explicit agreement or information in the instance, the default method for resolving identity is that all identifiers in the object's ID field must match the corresponding identifiers on the destination object.

4.3. Global Uniqueness (was 12)

Certain identifiers must be globally unique to prevent misidentification.

Globally unique identifiers may be created as either Universally Unique Identifiers (UUIDs—see ISO/IEC 11578:1996) or Object Identifiers (OIDs—see ITU-T X.660 or ISO/IEC 9834-3). UUIDs are globally unique by virtue of the method of their generation. OIDs are globally unique if the OID registration procedures defined by ISO in the 9834 series of standards are followed. A set of local identifiers may be made globally unique by prefixing them with a common global identifier.

The instance identifier (II) type has a root, which must be populated, and an extension, which is optional. Together, the root and extension must be globally unique.

Note that there are specific situations where only a local identifier is available. A typical example is on a point of care device. In these cases, either the context of use assigns a global identifier root, or the identifier is incomplete (some flavor of null).

4.4. OID Registry

For some scenarios, it is not enough that the identifier be globally unique; the identification must also be consistent among a group of systems exchanging V3 instances. Some concepts must be consistently identified within a realm, such as Social Security Numbers in the USA. Other concepts, notably shared standards such as HL7-defined concepts, ISO standards, and ICD-N and SNOMED terminologies, need to be consistently identified by all systems producing and consuming V3 instances.

One way to produce common consistent identification of these various kinds of objects is to maintain a central system where these identification concepts are registered. HL7 maintains an OID registry for this exact purpose. Any identifiers of interest to HL7 implementers may be registered on the HL7 OID registry, which includes

- OIDs issued by HL7 that refer to objects or concepts defined by HL7,
- OIDs issued by HL7 that refer to externally defined objects or concepts, and
- externally issued OIDs that refer to externally defined objects or concepts.

Note that the presence of an OID on the HL7 OID registry does not mean that HL7 claims responsibility for the concept of object identified, only that it is of interest to some HL7 customer. If the OID is in the HL7 OID branch, then HL7 has issued the OID and accepts responsibility for working with the owner of the object or concept to maintain the identification of the concept.

HL7 assigns an OID to each of its code systems, as well as to external standard coding systems that are being used with HL7 and HL7 Affiliate specifications. HL7 also assigns OIDs to public identifier-assigning authorities (e.g., U.S. State driver's license bureaus, U.S. Social Security Administration, HIPAA Provider ID registry, other countries' Social Security Administrations, Citizen ID registries, etc.).

The HL7 registered OIDs should be used for these organizations and namespaces, regardless of whether these organizations have other OIDs assigned by other registrars.

HL7 will also assign OIDs in its branch for HL7 users and vendors upon their request. When this is done, the registration authority (RA) for all OIDs under this assigned OID is delegated to the person or organization so assigned. The understanding is that they will have sole responsibility for further OID assignment under their new root and will perform such assignment consistent with the ISO standards governing OIDs. Any objects that are subsequently assigned by these RA delegates may be registered in the HL7 OID registry. Once this is done, the OID so registered will be used to identify the object in subsequent HL7 messages.

In some cases, technical errors are made during the OID assignment and registration processes, and sometimes an OID that has been registered for some time for HL7 purposes must be decommissioned and replaced. The OID itself is not retired. The retired flag is associated with the OID entry in the registry. This does not mean that the OID has been retired (the OID is merely "used up" for HL7 purposes). In these cases, the erroneous OID entry is identified as "Deprecated," and the OID that replaces it is identified in the OID registry. After a period of 2 years, the deprecated

OID will be set to “Retired,” but both it and its identified replacement remain in the HL7 OID registry.

Known Issue 13 (§ 2.13)

4.5. OID Conflict Resolution

When assigning OIDs to third parties or entities, HL7 investigates whether an OID is already assigned for such entities through other sources. If a preexisting OID is found, HL7 records the OID in the registry, and HL7 does not assign a duplicate OID in the HL7 branch. If no OID is found, HL7 will create one in the HL7 branch. If an appropriate third party can be identified, HL7 will notify the party when an OID is being assigned for that party in the HL7 branch.

Though HL7 exercises due diligence before assigning an OID in the HL7 branch to third parties, it is not possible, given the lack of a global OID registry mechanism, to make absolutely certain that there is no preexisting OID assignment for such third-party entities. Furthermore, external assigning authorities may encounter the same issue, failing to discover that HL7 has assigned an OID and assigning a duplicate. When such cases of duplicate assignment are discovered, HL7 works to resolve this situation via the deprecation process outlined above for technical errors.

4.6. HL7 OID branch

The HL7 root OID is 2.16.840.1.113883. All OIDs that HL7 assigns are issued within the space defined by this OID. This OID has immediate sub-spaces as summarized in this table:

Table 5: Defined Sub-spaces Beneath the HL7 OID Root

Identity	Use
0	HL7 Root OID
1	HL7 registered internal objects (other than published documents and organizational bodies)
2	HL7 organizational bodies and groups
3	External groups that have been issued an HL7 OID root for their own use as Registration Authorities
4	Registered externally maintained identifier systems
5	HL7 Internal Coding Systems

Table 5: Defined Sub-spaces Beneath the HL7 OID Root	
Identity	
Use	
6	Registered external coding systems (with an HL7 issued OID)
7	HL7 published documents
8	HL7 OID registered documentation products and artifacts
9	HL7 Registered conformance profiles
10	HL7 Registered Templates
11	HL7 defined and registered value sets
12	HL7 Version 2.x tables as code systems
13	Externally authored and curated value sets, HL7 registered
19	HL7 Examples Root used for published examples; meaningless identifier, not to be used for any actual entities

4.7. Specifying Identity with Instance Identifiers and Concept Descriptors

Both the Instance Identifier (II) and the Concept Descriptor (CD) data types are used to define how object identities are expressed in HL7 class attributes. Their basic structures are similar: each includes a namespace and an identifier class attribute, with other optional information in the case of the CD. In a CD, the namespace is the codeSystem attribute, and the identifier is the code attribute. In an II, the namespace is the root attribute and the identifier attribute is the extension. Via the linkage to a terminology and its richer set of attributes, the CD class allows the system to make usage of behaviours of terminologies (e.g. synonyms, language specificity, relationships and reasoning logic), while the II data type can only identify a unique object.

Due to their similar basic identification attributes, a guideline is offered to help modellers decide which data type to use for a given entity.

1. When the entity described by the class attribute is a concept (a class of entities) and/or the modeler wants to allow machine behavior to be applied to the concept instance at run time, the data type CD should be chosen (e.g. SNOMED CT code for headache).

2. If the purpose of a class attribute is to uniquely identify an object, and it resolves only to a single object within a class of entities, then the modeler should chose an II (e.g. drivers license number).
3. If, at design time, the purpose does not clearly fall into usages 1 or 2 above (e.g. because two perspectives allow to see one entity as class or as concrete object), a CD or II can be used arbitrarily. This does not impede interoperability, as the data type is defined in the model.

5. Coded Model Elements and Their Vocabularies

In HL7, a static model identifies and describes the information that can be recorded and exchanged in the form of classes and their attributes and associations.

A class attribute includes a description of the characteristic that the attribute represents, a cardinality constraint that identifies whether or not the attribute must always be present in an instance of the class and whether the attribute can appear more than once, and a data type that defines the range of possible values for the attribute. Certain data types employ enumerated lists of values to represent controlled sets of concepts: these are expressed as *coded elements*. In HL7 Version 3, a coded element is represented using one of the following data types: CD, CE, CV, CO, CR, and SC (see *Data Types: Abstract* or *Data Types: Abstract R2*). In HL7, the possible values for these elements and their associated meanings are defined in *code systems*, from which both the representations and the associated meanings are drawn.

The following section *(5.1)* describes the structural properties of concepts, code systems and value sets, while the next section *(5.2)* describes the usage of concepts, code systems and value sets in the business context of HL7 information models and messages.

5.1. Structural Properties of Concepts, Code Systems and Value Sets[1]

5.1.1. Concepts

A *concept* defines a unitary mental representation of a real or abstract thing; an atomic unit of thought. It should be unique in a given code system. A concept may have synonyms in terms of representation and it may be a primitive or compositional term

Concepts serve multiple purposes in the HL7 Version 3 model. Every object (entity, act, role, act_relationship, etc.) is associated with a concept which provides at least part of its intended meaning. Further, object attributes may also be associated with concepts that specify some or all of the attribute's meaning.

[1] *This section is currently under harmonization with the CTS2 information model.*

A *code* is a concept representation published by the author of a code system as part of the code system, and it is an entity of that *code system*. It is intended to be used as the preferred unique identifier for that concept in that code system and used in the code property of an HL7 coded data type.

The meaning of a *code* within a particular *code system* entity is valid only within that code system. For example, each table having enumerated codes in the HL7 Version 2 standard represents a different *code system,* since codes are sometimes used in different tables to have different meanings (e.g., “M” in the gender table means "Male," while “M” in the marital status table means "Married").

An HL7 concept can also be used to define a set of subordinate concepts, each of which is represented by a code. This set is then used to construct a list of possible values for a coded data type, where each code in the list represents a particular individual, process, or characteristic that logically belongs to the set of meanings represented by the parent concept.

HL7 has created a logical model of *concept* that specifies the characteristics used in the HL7 tooling. This model includes

1. An identifier that represents the concept within the context of a *code system* (described below). This identifier, when combined with the name of the code system itself, provides a globally unique name for the particular element. This globally unique name can be used in transactions and data records that span both space and time.
2. One or more designations (terms, appellations, symbols) that signify the concept
3. Additional text, annotations, references and other resources that serve to further identify and clarify what the concept is intended to denote
4. Where appropriate, assertions about relationships that might or must exist between the referenced concept and other concepts

5.1.2. Code Systems

A *code system* is a managed collection of concept identifiers, usually codes, but sometimes more complex sets of rules and references. They are often described as collections of uniquely identifiable concepts with associated representations, designations, associations, and meanings. Examples of code systems include ICD-9 CM, SNOMED CT, LOINC, and CPT. To meet the requirements of a code system as defined by HL7, a given concept representation must resolve to one and only one meaning within the code system. In the terminology model, a code system is represented by the Code System class.

Code systems are often referred to as terminologies, vocabularies, or coding schemes.

At a minimum, Code Systems have the following attributes:

- An identifier ("id") that uniquely identifies the Code System. In HL7, this ID is in the form of an ISO OID.
- A description ("description") that describes the Code System. This may include the code system uses and intent.
- Administrative information proper to the Code System, independent of any specific version of the Code System.

A code system is typically created for a particular purpose, and may include finite collections, such as concepts that represent individual countries, colors, or states. Code systems may also represent broad and complex collections of concepts, e.g., SNOMED-CT, ICD-9-CM, LOINC, and CPT.

A *Code System Entity* is any element or component of a code system which may have property and information specific to it defined in the code system. Concept representations, designations, and associations are all examples of code sytem entities.

Where possible, HL7 modelers faced with a requirement for a coded concept will reference an existing code system. Some of these code systems are replicated within the HL7 standard repository for stability or convenience, while others are documented as references. HL7 will only create a new code system when an appropriate existing code system is not available. Such is the case with Act Codes, which are defined and maintained by the HL7 organization. There are also cases where an otherwise appropriate external resource is not available due to licensing or other restrictions.

Code systems evolve over time. Changes occur because of corrections and clarifications, because the understanding of the entities being modeled evolves (e.g., new genes and proteins are discovered), because the entities being modeled change (e.g., new countries emerge; old countries are absorbed), or because the assessment of the relevance of particular entities within the knowledge resource change (e.g., the addition of ICD-9-CM morbidity codes related to terrorism). Because of this, it is important to be able to know which version of a given code system was used in the creation of an HL7 model or the recording of coded data.

The HL7 model depends, however, on the meaning of a specific concept identifier remaining constant over time, independent of the particular version of the knowledge resource. In cases where the knowledge resource itself doesn't enforce this (e.g. older versions of ICD-9-CM, where codes were retired and subsequently re-used to represent something different), it may be necessary to construct a composite unique identifier that consists of both the code and version.

Code systems may publish unitary concept representations composed of multiple indivisible or unitary concepts. Concept representations such as these are referred to as *pre-coordinated*, and are curated as part of the code system maintenance and publishing process. Some code systems have a mechanism to construct a representation for a new concept making use of formal relationships, published unitary concept representations, and a syntax, permitting the construction of concept representations that do not exist in the knowledge resource as it is published. Such externally created concept representations are called *post-coordinated expressions*. Note also that it is possible for a post-coordinated expression to be constructed for concepts that already have pre-coordinated codes published in the code system. For example, the concept "excision of pituitary gland"[2] may be pre-coordinated with a unique identifier and a designation of "hypohysectomy." An alternative definition for this concept can be obtained by post-coordinating codes for "brain excision" and "pituitary gland," without adding a reusable designation to the source system. Post-coordination allows us to derive complex composite terms by combining the concepts they are derived from, which reduces the number of concepts in the source system. Post-coordination requires the explicit resolution of complex questions of context and semantic precedence (e.g., the preposition "of" in this example).

[2] This example is from Dolin RH et al.: Selective Retrieval of Pre- and Post-coordinated SNOMED Concepts, Proc AMIA Symp. 2002; 210–214

5.1.3. Value Sets and Their Definitions

A *Value Set* represents a uniquely identifiable set of valid concept representations, where any concept representation can be tested to determine whether or not it is a member of the value set. A concept representation may be a single concept code or a post-coordinated combination of codes.

Value sets exist to constrain the permissible content for a coded element in an HL7 static model or data type property. A class attribute expressed as a coded data type must be associated with a list of codes that represent the possible concept designations that can be represented in that attribute. *Value sets* cannot have null content, and must contain at least one concept representation. Any given concept is generally (but not required to be) represented by only a single code within the *Value Set*. Identical codes from different code systems are allowed because they can be disambiguated by identifying the code system they come from.

Ideally, a given *concept* should be represented only by a single *code*. However, in unusual circumstances, a given concept can have more than one code (e.g. where a different case is used to signify the same concept, as 'l' and 'L' in UCUM for 'litre').

Value set complexity may range from a simple flat list of concept codes drawn from a single code system to an unbounded hierarchical set of implicit post-coordinated expressions drawn from multiple code systems.

5.1.3.1.1. Value Set Definition Types

There are two basic approaches that can be used to define the contents of a value set:

- Extensional definition: Explicitly enumerating each of the value set elements.
- Intensional definition: Defining an algorithm that, when executed by a machine (or interpreted by a human being), yields such a set of elements.

An *extensional* definition is an enumeration of all of the concepts within the value set. *Value sets* defined by extension are composed of explicitly enumerated sets of concept representations (with the *code system* in which they are valid). The simplest case is when the value set consists of only one *code*.

An *intensional* value set definition is a set of rules that can be resolved (ideally computationally) to an exact list of concept representations at a particular point in time. While the construction rules can potentially be quite elaborate, HL7 has identified a core set of rules that appear to be useful in most circumstances. For the sake of value set definition interoperability, HL7 strongly recommends that these algorithms be used whenever possible.

Intensional definitions can specify

- All active unique identifiers from a given coding system

- All unique identifiers that participate in a specified relationship with a given concept in a coding system, which may or may not include the specified code itself
- The transitive closure of a specified transitive relationship with a given code, including or excluding the code itself
- A *nested value set* definition in which a value set entry references another value set (a child value set). There is no preset limit to the level of nesting allowed within value sets. Value sets cannot contain themselves, or any of their ancestors (i.e., they cannot be defined recursively). Nested value sets are always intensionally defined in HL7.
- Unions, intersections and exclusions of any of the above

The *intentional definition* must be specific enough that it is always possible at a point in time (within a specific version of the code system) to determine whether a given value (including post coordinated collections of codes) is a member of the value set. For example, an intensional value set definition might be defined as, "All SNOMED CT concepts that are children of the SNOMED CT concept 'Diabetes Mellitus.'"

5.1.3.1.2. Unique Meaning Rule

HL7 recommends that, whenever possible, a value set be drawn from a single code system. This is not always practical, however, and when it happens, it is the responsibility of creator of the value set definition to assure that the set doesn't contain more than one globally unique identifier that could potentially denote the same entity. Care must be taken to ensure that every entity represented in the value set has only one possible globally unique identifier. For example, both CPT and LOINC have codes that represent hematocrit, meaning that there are two possible globally unique identifiers with approximately the same meaning: 2.16.840.1.113883.6.1#4544-3 for LOINC and 2.16.840.1.113883.6.12#85014 for CPT-4. If both of these are possible choices in a value set, data encoded with one code may be missed when searching using the other code, and interoperability in general may suffer. Further, dividing semantic stewardship for the value domain can introduce semantic traps: the different source system owners may define their concepts with differences that escape one user's notice, but have a significant effect on another user's interpretation. In the above example, the CPT code specifies a test order, but does not require that the test be filled by a specific method: it refers to any hematocrit. The LOINC code, on the other hand, specifies the automated count method, specifically excluding packed cell volume tests. Equating these codes may be permissible in some contexts, but would be incorrect in others.

Extra care must be taken to assure that overlapping references do not appear in implicitly defined value sets, especially as there is no easy way to automatically determine when this situation exists. Again, the easiest way to avoid this is to avoid the use of multiple code systems in a single value set where possible.

5.1.3.2. Value Set Resolution

To obtain a set of concept designations, value sets must be resolved. While this is straightforward for extensional value sets, an intensional value set definition must be resolved to an expansion. This process must record the exact set of definitional rules used to resolve the value set in the desired version (see below). This can be done as early as the point of value set definition or as late as run time, but the definition needs to be available to accompany any data that is generated using the value set. Note that the resolution of post-coordinated designations is independent of how a value set is defined.

5.1.3.3. Value Set Definition Versioning

The definition of a value set can change over time. New identifiers may be added to or removed from a value set definition, and the rules used to construct the set may change. When a value set definition changes, it should be done in a way that ensures both the old and new versions are available for comparison.

There are multiple strategies for tracking value set versions. Two of the most common are

1. to increment the version number each time a change is made to the value set
2. to track modification dates for each change to the value set.

In HL7 standards, value set versions are determined by effective date (the date at which the value set version became effective), and not by available date (the date the value set version was made available within an organization) or by a version number. This policy has the following implications:

1. For enumerated value sets maintained by HL7, the activation date and deactivation date for individual codes in the value set must be maintained as part of the value set database.
2. For intensionally defined value sets in the HL7 value set database, the activation date and superseded date must be recorded (tracked) each time the logic of the definition is changed.
3. For externally maintained terminologies that have named or numbered releases, a table must be maintained that shows the modification dates for the named or numbered releases.
4. For externally maintained terminologies that maintain modification dates for each individual code change, no additional information is needed.

A sub-value set is a sub-set of a parent superset. There may be no designations in a sub-value set that are not also contained in the superset. A sub-value set is generally created as part of the successive constraining process of model development. The CTS2 models this via the DesignationValueSetVersionMembership class.

5.2. Binding Vocabulary for Use

When HL7 information models and messages are designed and constrained for use, those attributes requiring coded values are associated with value sets. In HL7, this process is called *vocabulary binding*. Vocabulary bindings are specific to a model or message element's *realm, concept domain* and *usage context*[3].

Binding consists of identifying the coded attributes in a model, defining value sets that must be used in those coded attributes and declaring the circumstances under which those value sets must be used with those coded attributes.

The binding process associates a model element with the identifier of a value set. This identifier refers to the value set definition. Bindings also have an optional version identifier. This is used to identify the particular expression of the definition (if it has changed over time) and to identify the correct version of the code systems involved.

Both the definition of the value set and the binding of the model element to the value set definition are recorded for use when systems use the specification to communicate—whether to select, validate, or interpret a value. The location where definition and bindings are stored may differ from Realm to Realm. Some Realms may use the HL7 repository for both definition and binding; others may use resources stipulated by governmental authorities, regional cooperatives, or specialized institutions. In any case, any system using the HL7 specification requires access to these resources.

An element may be bound to different value sets under different circumstances. The following subsections describe the way HL7 assigns the vocabulary structures described above to model elements.

5.2.1. Realms

In HL7, the broadest binding context is the realm[4]. All model instances must declare a particular realm (or sub-realm) based on the jurisdiction from which they originate, for which they are destined, or for some third jurisdiction by site-specific agreement. The declared realm applies to the entire model or specification artifact: it is not specific to individual elements of that model or artifact.

A *Realm* refers to a named interoperability conformance space, meaning that all static models within a particular Realm share the same conformance bindings. In nontechnical terms, it can be considered a dialect where speakers use the semantics of the language but agree to use certain terms that are specific to their community. A Realm is also commonly referred to as a *Binding Realm*, especially where other types of realms are being discussed. A binding realm has a unique realm code: the binding

[3] The CTS2 information model expresses the relationship the structural value set view and the HL7 usage view via the classes JurisdictionalDomain (which correspond to an HL7 realm), ConceptDomain, UsageContext and ValueSetContextBinding.

[4] A Realm is represented as a JursidictionalDomain in the CTS2 model

realm Germany has a realm code of DE, and the steward is HL7 Germany. In order to enable conformance, the name of the Realm is carried in the model instance.

In the interest of maximizing interoperability, interoperability spaces should be as large as possible: Realms are preferred to be large-grained. A Realm is used to provide and manage the bindings of value sets to reflect rules within a conformance space—e.g., a country.

5.2.1.1. Defined Realms

5.2.1.1.1. Affiliate Realms

Each HL7 International Affiliate owns a Realm bounded by the geographic scope of the Affiliate.

5.2.1.1.2. Sub-Realms

In some circumstances, an International Affiliate might choose to create additional binding realms narrower in scope than the affiliate-wide binding realm. The sub-binding realms might be constructed geographically (e.g., regions, states, provinces, etc.) or by type of implementation (e.g., human vs. veterinarian). Sub-binding realms can only be created by International Affiliates.

Note**:** Because the purpose of binding sub-realms is to allow the use of different code sets for the same message within an affiliate, they can cause interoperability issues within the Affiliate. They should therefore only be introduced after careful consideration of the interoperability consequences.

5.2.1.1.3. Combination Realms

There is a need for Realms that combine more than one country. There is a North America Realm for Cancer Registries for the US and Canada. This Realm has been created as a combination of the US Realm and the Canadian Realm. Any Realms may be combined for such a purpose to make an interoperability space that extends beyond one country. The International Affiliates who agree to do so are the stewards of the combination realm.

5.2.1.2. Generic Realms

Four Generic Realms have been defined that are not specific to an International Affiliate (or to a delegated subset or coordinated alliance of International Affiliates). These generic realms should never appear in model instances: they are only used in the standards creation process.

5.2.1.2.1. Universal Realm

The *Universal Realm* constitutes the core HL7 realm which is by definition invariable and fully inherited by all HL7 models and compliant implementations. If a Universal Realm binding exists for any attribute, all implementations are expected to use the value set associated with that binding. No other bindings may exist. Structural elements (e.g., Act.classCode) and most data types are examples of content in this realm. Content from domain technical committees is rarely included in the Universal Realm and, when introduced, must go through special processes to ensure full international consensus on the universal constraint to a single binding.

5.2.1.2.2. Example Realm

The *Example Realm* is used for bindings to sets of codes that are known to provide incomplete or non-implementable coverage to the associated domain. They are used to fulfill the example requirement of concept domain definition: a concept domain definition (below) must include three examples. They may also be used in the construction of realm-independent example instances. Example realms are not needed when appropriate Representative Realms exist.

5.2.1.2.3. Representative Realm

Representative Realm bindings are intended to be complete and implementable examples. Unlike universal bindings, there is no expectation that all (or any, necessarily) affiliates will choose to adopt the representative realm bindings. Representative realm bindings provide a starting point and a focus for consensus, while recognizing that cultural and political variations between International Affiliates may result in alternative bindings. To qualify for Representative Realm designation, candidate content must be sufficiently comprehensive and internally consistent to be adoptable and implementable by specialized binding realms. A representative realm binding has no official force in an affiliate unless the affiliate chooses to adopt it with an Affiliate-specific binding. When an Affiliate does choose to use a binding from the representative realm, the binding definition is copied into the Affiliate realm: there is no persistent link that may result in unintended cascading changes.

5.2.1.2.4. Unclassified Realm

The *Unclassified Realm* accommodates content that is new and in the process of being refined as well as legacy content that has not yet been promoted to its destination realm. The Unclassified Realm may also serve as a transition point for content contributed from other Realms. The unclassified realm exists for HL7 administrative purposes and has no effect on implementations

5.2.2. Concept Domains and Usage Context

An HL7 *Concept Domain* is a named category of like concepts (a semantic type) that will be bound to one or more attributes in a static model whose data types are coded. *Concept Domains* exist to constrain the intent of the coded element while deferring the association of the element to a specific coded terminology until later in the model development process. Thus, *Concept Domains* are independent of any specific vocabulary or code system.

Concept domains are universal in nature (independent of any realm), so the name for a concept domain should never contain any reference to a specific realm. Concept domains are and registered with HL7: they are proposed as part of the HL7 standards development process and are approved by the RIM harmonization process. Both processes are described in the *HL7 Development Framework* (HDF).

A concept domain is documented by specifying a name and a narrative definition. In addition, at least three concept identifier examples that represent possible values for the attribute are required. The identifiers should represent concepts that characterize the intent and purpose of the concept domain. This can be accomplished in one of the following ways:

1. Including three example concept identifiers as part of the narrative definition;
2. Associating the concept domain with a value set that contains at least three example concept identifiers in the context of the Example Realm; or
3. Associating the domain with a value set that contains a set of concept identifiers whose denotations completely cover the intended meaning of the domain, using either the Universal or the Representative Realm as a context.

5.2.2.1. Examples of Concept Domains

The concept domain *HumanLanguage* carries the description, "Codes for the representation of the names of human languages." The set of concept identifiers that represent different human languages can be drawn from different code systems, depending on which realm or sub-realm is creating the message. For example, the United States Realm may choose to use a value set that includes concept identifiers for various Native American languages, while New Zealand may find such a value set inappropriate.

[This would be the place to address structural codes, if that's where value sets are bound to concept domains rather than model attributes. If value sets are always bound to concept domains and never to model attributes, then 5.2 is incorrect.]

5.2.2.2. Sub-Domains

One concept domain may be defined to be "sub domain" of another. This means that the intended meaning and reference of the sub-domain is intended to be narrower than the meaning of the parent. For example, there is a domain called observation method, with a sub-domain of genetic observation method. This is not intended to be an ontological assertion; its primary purpose is to indicate that all of the coded identifiers in a value set that is associated with the sub-domain should also be valid identifiers for the parent domain, though the reverse may not be true.

5.2.2.3. Constraining Concept Domains on Attributes

The HL7 RIM specifies concept domains for all coded attributes. It does not, however, associate all of these concept domains with value sets. When a specific HL7 static model is produced, the modelers determine how to handle each coded element. They may choose to

1. retain the concept domain that was specified in the RIM,
2. substitute a sub-domain of the original RIM concept domain, restricting the possible values that can go in the associated attribute, or
3. bind the element with a single value set, indicating that the identifiers in the associated value set *must* be used in this particular model.

A static model is considered to be "abstract" as long as it contains at least one coded element that is not associated with a value set. A model cannot be used to create instances until all coded elements have been associated with value sets. The following sections describe different mechanisms by which concept domains are associated with value sets to render a model "concrete," or usable.

5.2.3. Binding Mechanisms and Strategies

The Realm, concept domain, and usage context define the scope for a particular binding between a model element and a value set. This section explains how, given those parameters, the binding is defined.

5.2.3.1. Binding Mechanisms

5.2.3.1.1. Binding Schedule Mechanisms

There are two schedule mechanisms for binding an attribute or data type property to a value set that HL7 has agreed to support.

Model Binding involves binding a coded attribute or data type property in a static model to a specific value set. The contents of the value set must be consistent with the concept domain definition in the model's parent (DMIM, RIM, etc.). Where the corresponding attribute or property in the parent model is a value-set or is a domain which has been bound to a value-set, the value-set bound to the child model attribute or property must be the same or a subset of the parent model value-set.

Context Binding involves binding a coded attribute or data type property to a combination of concept domain, realm, and (optionally) context of use, without knowing the value set at design time. This combination is later used to identify the appropriate value set. This type of binding is used primarily when the value set to be bound is not known at message design time. Because the realm associated with the instance is identified within the instance, it is possible for a receiver who knows the message specification (and thus the concept domain) to determine the appropriate value set to validate against. In this context, it is important to understand that a principle of HL7 is that after an HL7 model has been balloted and is complete it may still not be implementable, as certain coded attributes may not have been constrained beyond concept domains (to be implementable, every coded attribute must be associated with a value set of permissible codes). Context binding is the mechanism whereby an abstract model specification can be made implementable by decoupling value set specification from model development.

This distinction between Model and Context Binding affects the timing of activities, but it is driven by a division of labor. Model Binding is performed by modelers at model design time, and it is used for model elements that should be bound to a specific value set irrespective of Realm, e.g. structural codes. Context Binding is performed by Realm representatives, and it obtains for all elements using the Concept Domain in any model or artifact within the Realm.

The two approaches may require coordination. A Realm may assert a context Binding, which a modeling team may further restrict, but not broaden or contravene, in a particular model. Similarly, if there is already a Model Binding where a given Realm wishes to assert a Context Binding, the Context Binding should identify a superset of the values in the extant model Binding(s). Known Issue.

5.2.3.1.2. Binding Version Mechanisms

There are three version mechanisms for binding vocabulary to coded model elements, each of which may be used with each of the two schedule mechanisms described above: these are Static Binding, the Single Code, and Dynamic Binding.

Static binding is a binding to a specified version of a value set. As a result, the allowed values of the value set do not change automatically as new values are added to a value set, and the expanded list can be included in an implementation guide. A static binding is fully specified when the binding references a specific version (date) as well as the value-set OID/unique name.

Dynamic binding is a binding to a value set without a specified version. As a result, the allowed values for a coded item automatically change as the value set or its underlying code system is maintained over time. This means that for dynamic binding, the binding is to the most current version of the value set at a given point in time. A documented expansion of a dynamically bound value set must always be checked for currency before use.

Dynamic binding is fully specified when the binding references the value-set OID or unique name. It need not specify a version date: it stipulates that the most recent version be used at runtime.

Single Code binding is defined as the binding of a single code to a coded attribute or data type property in a static model. It can be seen as a special case of static binding with a value set of size one.

5.2.3.1.3. Unbound Domains

In some situations, a concept domain referenced in an HL7 static model might not have an applicable binding for the affiliate making use of the model (no universal binding and no Affiliate binding for that affiliate). In that case, the domain is considered to be un-bound. The determination of the set of codes to use remains subject to site-specific negotiation until an applicable binding is created universally or for that affiliate.

5.2.3.2. Binding Methods

For each binding schedule mechanism (model, context), there are three available version mechanisms: dynamic, static and single code. This means that in HL7, six binding methods are available. We will first discuss model, then context binding methods.

The following table outlines the key differences at a high level:

	Model	Context
Static	Realm known at design time Domain known at design time Value set known at design time Value set version known at design time	Realm known at design time Domain known at design time Value set known at intermediate time Value set version known at intermediate time
Dynamic	Realm known at design time Domain known at design time Value set known at design time Value set version known at run time	Realm known at design time Domain known at design time Value set known at intermediate time Value set version known at run time
Single	Code known at design time	Code known at intermediate time

5.2.3.2.1. Methods for Model Binding

5.2.3.2.1.1. Dynamic Model Binding of Value sets

This method is used when binding a value set to a coded attribute or data type property in a static model at design time where the coded content of the value set should change to reflect the most current thinking. Dynamic Model Binding for both extensionally and intensionally defined value sets (native or imported) is accomplished by referencing the OID or the name (or both) of the value set in the binding statement; the effective time of the operation on the value set (such as validation) is the date of the expansion of the value set.

5.2.3.2.1.2. Static Model Binding of Value Set

This method is used when binding a value set to a coded attribute or data type property in a static model at design time, where stability and predictability are more important than up-to-the-minute code system changes. Static Model Binding for both extensionally and intensionally defined value sets is accomplished by referencing the value set OID or name (or both) and the effective date of the value set in the binding statement. The date of the binding statement is the effective date of the expansion of the value set.

5.2.3.2.1.3. Model Binding to a Single Code

This method is used when binding a single code to a coded attribute or data type property in a static model at design time. The binding is accomplished by stating the code, the code system OID or name (or both), and, optionally, the effective date of the code system version.

Known Issue 18 (§ 2.18)

5.2.3.2.2. Methods for Context Binding

5.2.3.2.2.1. Dynamic Context Binding of Value Sets

This method is used when a concept domain is bound to a coded attribute or data type property in a static model and the reference is to be resolved to a dynamic value set at run time. A value set must be assigned to the combination of domain and realm at some time between model design and run time. The following elements must be known in order to resolve the domain name to a specific value set:

1. The identity of the static model
2. The unique identity of the coded attribute or data type property in the static model (ClassName.attributeName[.datatypePropertyName])
3. The concept domain that is bound to the coded attribute or data type propertyKnown Issue 19 (§ 2.19)
4. The binding-realm within which the data exchange is to occur

The first three properties are part of the model binding statement for the model. The last two properties are part of the Context Binding statement contained in the terminology server. The Binding-Realm is passed as part of the context as the message is parsed (RealmCode); the concept domain, the realm, and the value set must be available to the terminology server, and may be included in an implementation guide. Known Issue 20 (§ 2.20)

5.2.3.2.2.2. Static Context Binding of Value Sets

This method is used when a concept domain is bound to a coded attribute or data type property in a static model and the reference is to be resolved to a static value at run or compile time. The following elements must be known to resolve the domain name to a specific value set:

1. The five elements listed above for dynamic context binding
2. The effective date of the value set

The first three properties are part of the model binding statement for the model. The last three properties are part of the Situation Constrained statement. The Binding-Realm is passed in a message instance wrapper (RealmCode); the concept domain, the realm, the value set, and the effective date must be available to the terminology server, and may be included in an implementation guide.

5.2.3.2.2.3. Context Binding to a Single Code

This method is used when a concept domain is bound to a coded attribute or data type property in a static model and the reference is to be resolved to a single code in a code set at runtime. The following elements must be known to resolve the domain name to a specific coded value:

1. The identity of the static model
2. The unique identity of the coded attribute or data type property in the static model (ClassName.attributeName[.datatypePropertyName]+)
3. The concept domain that is bound to the coded attribute or data type propertyKnown Issue 19 (§ 2.19)
4. The binding-realm within which the data exchange is to occur
5. The code, the code system OID or name (or both) and optionally the effective date of the code system versionKnown Issue 19 (§ 2.19)

5.2.3.2.3. Binding Syntax

Binding syntax is determined by the conformance work group. It is documented in [source].

5.2.3.3. Additional Notes on Domains and Value Sets

5.2.3.3.1. Concept Domain and Value Set Naming Conventions

HL7 concept domains and value sets will be named according to the following rules:

- All concept domains and value sets will use “camel back” style names.
- The name will be restricted to the basic 26-character alphabet and the digits 0-9 using ASCII characters. White space (tabs, spaces), punctuation (periods, commas, colons, semicolons, parentheses, quotes, etc.), underscores, hyphens or other separators are not allowed in the name.
- The leading character must be upper-case alpha
- Concept domain and concept sub-domain names should be accurate labels for the concept spaces that they designate. Concept domain and concept sub-domain names should never include realm or code-system specific information. The concept domain name should also be independent of the RIM attribute where possible, so that the concept domain can be re-used with different attributes. For example, a concept domain should be called “HumanGender” rather than “PatientGender” so that the same domain could be bound to the “GuarantorGender” attribute.
- Value sets may be named by combining the name of the concept domain with other contextual information that will uniquely identify the value set; this is very helpful when a value set is appropriate for only a single sub-domain (which is most often the case). If a value set is expected to be used in more than one concept domain, then a more general name that clearly identifies the usage of the value set should be created. For example, the following would be appropriate names:

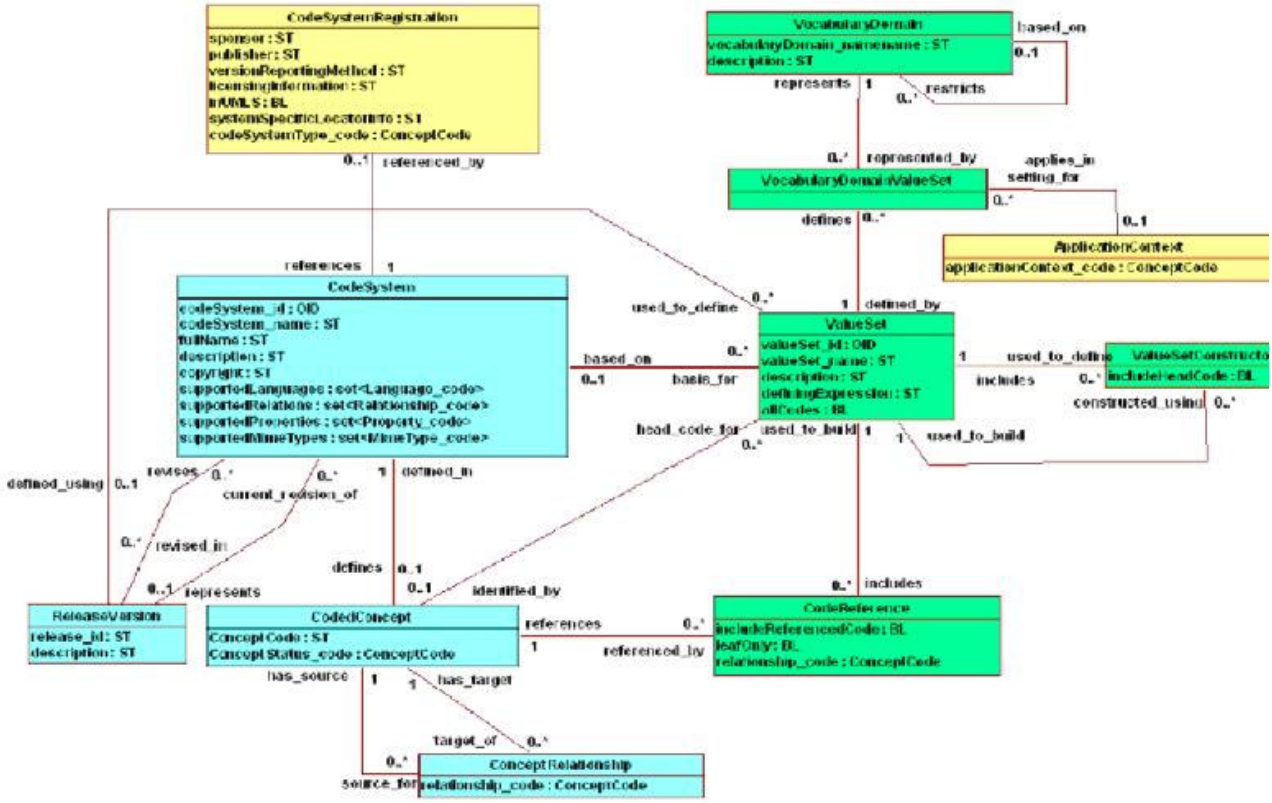
- Concept Domain: HumanGender
- Value Set: HumanGenderUSRealm
- Concept Domain: Country
- Value Set: CountryFIPS

5.2.3.3.2. Root Concepts

A Value set may be referenced as *abstract* or *specializable*. If a value set is referenced as abstract, the “navigational concept" - the root concept of which all other concepts in the value set are specializations - is not selectable. If specializable, the root concept (head code) is selectable, meaning that highest level concept can be selected without further refinement.

The terms “abstract” and “specializable” may be thought of as referring to the root concept as an object-oriented class, which may be concrete (usable) and specializable, or abstract (not usable except via specialization).

NOTE: Being *abstract* or *specializable* is not a property of value set itself, but is an indication that for any specified context, the value set should be referenced as either abstract or specializable.Known Issue 16 (§ 2.16)



Value Sets

5.2.3.3.3. X-Domain (X-Value Set) [Deprecated]

In HL7, there are value sets called “X-domains.” “X-Domain” is a misnomer: a more proper name would be “x-value sets,” since they are really HL7-defined value sets or sub-value sets. X-Domains came into existence to address a muddling between the code system hierarchy and value sets. Earlier versions of the vocabulary maintenance tools didn’t distinguish between a value set that included concepts X, Y and Z and a concept code with subtypes X, Y and Z. The prefix “X-” was added to value sets that were intended to represent simple collections, not conceptual hierarchies. A rule was established that new concept codes couldn’t be introduced within an X-domain. They first had to be entered elsewhere in the coding scheme hierarchy and then added separately to the X-domain. As time permits, all X-Domains will be replaced by value sets.

Known Issue 17 (§ 2.17)

6. Accountability and Updates (was 11)

MnM: include a conformance review referring back to appropriate sections

In addition to using update Mode to describe the changes that have happened or should happen, instances can also carry accountability information relating to the information in the message, both associations and attributes. The accountability information can include the time range during which the information was or is valid, and a link to the control act associated with the value. The control act can describe who made the change, when the change was made, what application made the change, and some context for the change in the overall dynamic model.

Known Issue 12 (§ 2.12)

Generally, this form of accountability history is used in registry-type systems where there is a strong need for the receiver to establish the authority on which a particular piece of data is being changed. Understanding the details can be important in helping a receiver make the determination whether they wish to adopt the change.

Accountability information will be handled by using the HXIT generic type extension. This extension will be applicable to both attributes and to associations. To provide support for accountability information in addition to a time stamps, the HXIT extension will be modified to allow for the presence of either a simple time stamps or a ControlAct.id reference. The reference will allow the changes to an individual attribute or association to be associated with the ControlAct that changed it. The ControlAct can be used to convey such information as event time, author, authoring organization, data-enterer, reason, and any other accountability information deemed to be important.

When working with interactions triggered by a state-transition notification, a state-transition request or a state-transition fulfillment request, the individual ControlAct

classes associated with the changes to each attribute or association will be sent as 'Components' of the ControlAct in the ControlAct wrapper. When working with query response interactions, the ControlAct classes will be attached to the focal class of the query response via a subject association.

Multiple associations and attributes may reference a single ControlAct, or each may reference a separate one.

Committees must explicitly enable exposing the Accountability History link for a given attribute or association.

6.1. Update control (was 7)

HL7 Static models are used to represent information about the real world when it is exchanged between systems. The objects in the instance represent real world concepts about which a certain amount of information is known.

Snapshot: A methodology in which the sending system includes all the data it has into the message with no specific indications of which data items were added, replaced, or removed. The term was chosen because the source system sends a "snapshot" of the objects as it knows them.

Snapshot is typically used when information is exchanged between systems where the destination system is not known, or where it is not clear how much information the destination system already has about the real world concept.

When a receiving application processes an object that is represented using a snap shot, and it already has information about the real world concept that matches this object, the application should match objects in the instance with the information it already has, and then appropriately process the information from the message to the information it has on file (for instance, in some cases it would make sense to merge all the attributes and associations of the objects).

Potential Advantages:

- Can be easier for senders to implement
- Many sending systems implement Version 2 messages in this fashion

Potential Disadvantages

- Typically more complicated for receivers to process appropriately
- Easier for relevant data to be deleted

Update mode: A methodology in which the message designer specifies the allowable update mode values for items within the message and the message sender specifies the specific update mode value for items for items within the message.

However in some contexts, the destination system is well known and there is an implicit or explicit contract between the source and destination systems that ensures

the information the destination system holds is well known to the source system. In such contexts, it is possible to only send the changes that have occurred on the source system or should occur on the destination system. These changes may be additions, deletions, and revisions to existing data. This practice is known as "update" mode.

Another use for update mode is where the source application includes all the same data items in the message specification as it would for snapshot node, but marks each value for each data item in the message specification that indicates whether it is added, replaces another item, or has not changed.

Where update mode can be used, it offers several advantages. Potential Advantages (depending how it is used):

- reduced instance size
- The receiver does not need to compare data to determine what changes the sender has made
- Where the receiver gathers data from multiple sources, it does not need to store ‘images’ of data received from a particular sender to ensure that it can adequately compare to the previously sent data when determining changes
- reduced processing time
- simpler implementation decision making
- Conveys important information for how the sending system has processed the information
- Query responses are able to document accountability information in terms of what changes were performed (see accountability below).

Potential Disadvantages:

- update mode offers the opportunity for two systems to get information out of sync, so modellers and implementors should always be careful.
- Typically requires for effort for the source system

The normal mode for V3 instances is snapshot; update mode is only allowed when the [[constraining model]] design specifically allows update mode.

Update Control interpretation depends on the context of the message type:

1. When used in a message driven by a state-transition notification or a state-transition fulfillment request trigger event (where the focal class is an object owned by the sending system), the update control represents the change that occurred on the sending system as a result of the state change associated with the trigger event. The recipient is not bound to make the same changes as those done on the sending system.
2. When used in a message driven by a state-transition request trigger event (where the focal class is an object owned by the receiving system), the update control represents the change that is desired by the sending system as a result of that trigger event. If the recipient accepts the request, they must make the requested changes.
3. When used in a query response message, the update control represents the most recent change that has occurred to the sender’s object within back to a

specified time. The committee may allow the time from which changes are reported to be specified by a query parameter or fixed by the query definition. If not otherwise specified, the start time is the first time the system became aware of the object.

6.2. Update Mode

HL7 provides a single property called updateMode to support the concepts defined in Update Control, Referencing Objects, and Identifying Objects.

Note: a more appropriate name might be useCode, but the property name is updateMode for backwards compatibility reasons.

Note: The updateMode property actually applies to associations and attributes, not to classes and datatypes, though it is formally defined on the types.

The value of the updateMode property identifies how the attribute or association contributes to the processing of the instance. HL7 models strictly control the use of the updateMode attribute; it may only be populated with a value that the [[constraining model (internal reference)]] allows. If there is no value, then the constraining model should be consulted for guidance on how the instance should be processed.

The updateMode property can have one of the following values:

Table 4: Table of Update Mode Values

Code	Name	Description
A	Add	The item was (or is to be) added, having not been present immediately before. *(If it is already present, this may be treated as an error condition.)*
D	Delete	The item was (or is to be) removed (sometimes referred to as deleted). *If the item is part of a collection, delete any matching items.*
R	Replace	The item existed previously and has (or is to be) revised. *(If an item does not already exist, this may be treated as an error condition.)*
AR	Add or Replace	The item was (or is to be) either added or replaced. -- *[Delete: (This option is included to support one specific case, discussed below. Its general use is discouraged, the preferred metdodology is to use the combination of the individual Add and Replace values.)]--*
N	No Change	There was (or is to be) no change to the item. This is primarily used when this element has not changed, but other attributes in the instance have changed.

Table 4: Table of Update Mode Values		
Code	**Name**	**Description**
U	Unknown	It is not specified whether or what kind of change has occurred to The item, or whether The item is present as a reference or identifying property. *(replaces: It's not specified whether the item was (or is to be) added, revised, or not changed.)*
REF	*reference*	*This item provides enough information to allow a processing system to locate the full applicable record by identifying the object.*
K	*Key*	*This item is part of the identifying information for this object.*

Notes:

1. Portions requiring harmonisation proposals in italics
2. R and AR may not be applied to multiple attribute values within a DSET, BAG or LIST. If a single attribute value is marked with a R is used to update a collection, the single value replaces all the items in the collection
3. REF may only be applied to associations, not attributes.
4. U is semantically equivalent to a nullFlavor of NI. However due to some methodological issues in V3, a specific code is required to in some circumstances.
5. If an item is deleted from a collection, all matching items should be deleted from the collection

7. Model Designer Guidance (was 10)

This section is intended for people designing static models, typically HL7 domain committees.

When designing a model, a committee may allow UpdateMode to be used on attributes and associations identified by the committee. To enable UpdateMode, the committee must select the set of permitted updateMode values.

In addition to identifying the allowed set of values, the committee may also choose to identify a 'default' updateMode for the attribute or association. This is the updateMode that will be assumed by the receiver if none is specified in the instance.

updateMode of "Replace" is not permitted on Entity.id, Role.id, Participation.id and Act.id attribute. If an identifier was captured erroneously, the incorrect submission should be nullified and the record resubmitted with the correct identifier. If a new

identifier has been issued, replacing the old identifier, this should be handled as a supersedes or replaces relationship between the class with the old identifier and the class with the new identifier.

If no UpdateMode set is enabled for an attribute or association, it is the same as if the UpdateMode were set to 'Unknown'. The effective behavior is that of 'Snapshot'. I.e. the current element value is specified with no indication of whether it was changed or not.

The allowed UpdateMode set available for RIM attributes is empty by default. This means that committees must specifically enable UpdateMode by declaring an allowed set of Update Modes within their design for each attribute or association in their DIM where they want them to be used. Once an UpdateMode set has been defined in the DIM, any derived models (CIM, serialized static models or serialized message models). I.e. Update Modes may be removed from the allowed set, but never added.

If a committee defines update modes for a particular attribute or association, implementers must support the allowed update mode set to be conformant. (Failure to support the complete set defined by the committee may result in interoperability problems.) Implementers should be able to document what update modes they support in their conformance profile, but failure to support those identified by the committee that defined the artifact is considered non-conformant.

The committee does not need to define a default update mode, and may define a default at any derived model. Once a default is defined, it may not be removed or changed in any subsequently derived models. I.e. if a default is defined in an R-MIM, it may not be changed or removed in serialized static models or Message Types derived from that R-MIM. Because of this restriction, committees are discouraged from defining a default UpdateMode at the DIM level.

Update modes should not be specified in templates, as they are intended to be used across multiple different static models that make their own rules about use of updateMode.

Notes:

1. UpdateMode is not a concept that should appear in all, or even in most models developed by committees. It should be treated as an 'advanced modeling concept', and only employed in models where the facilitator is certain that the concept is needed to adequately reflect the needs identified by their committee. Furthermore it should only be enabled on those attributes or associations where there is an identified need. When a facilitator has identified a perceived requirement for UpdateMode in their model, they are encouraged to bring the requirement to the Modeling and Methodology Technical Committee for review.
2. UpdateMode will primarily be used for trigger events where the state transition is "revise" and for query responses; however, it may be appropriate in other circumstances. Committees are encouraged to discuss additional patterns for usage so that they may be reflected in this document.
3. UpdateMode should not be enabled in Transmission or ControlAct wrappers.

4. There is no way to Remove a single element from a BAG where there are multiple matching elements because there is no means to indicate which occurrence within the bag is to be removed.
5. Id attributes should never be sent with an UpdateMode of Replace. If such a use-case arises, it will addressed as a future methodology change.
6. Classes that do not carry an id attribute cannot be identified at all.

Appendix A: Code system types

A *vocabulary* in the HL7 sense is a terminology, as defined below.
A *terminology* is a set of concepts designated by terms belonging to a special domain of knowledge, or subject field . A terminology is not an arbitrary collection of terms, but a collection of designations attributed to concepts making up the knowledge structure of a subject field. The concepts of a well-structured terminology should constitute a coherent concept system based on the relations established between concepts. The meaning of each concept within a system can be determined by the intension, i.e. the unique set of characteristics constituting the concept, or its extension, i.e. the enumeration of the subordinate concepts of a concept.
Most terminologies can be classified as either *reference terminologies* or *interface terminologies,* though there are other types, e.g. indexing terminologies (like UMLS).
A *reference terminology* is a terminology in which every concept designation has a formal, machine-usable definition supporting data aggregation and retrieval. *Interface terminologies* are used to mediate between a user's colloquial conceptualizations of concept descriptions and an underlying reference terminology.
A terminology can be seen as a kind of *ontology*. An ontology has all the characteristics of a terminology, but it also uses a methodology for the description of the relationships between concepts (e.g., Description Logic) which allows humans and (depending on the methodology) machines to reason about the properties of that subject domain and to deduce knowledge from the way the concepts relate to each other. Ontologies allow different types of concept relationships to capture their richness. Hierarchical terminologies can be seen as rudimentary ontologies as they describe the relationships of concepts to each-other but have only one type of relationship.
A *taxonomy* (synonym: classification) is a somewhat less formal system. Taxonomies often have less granular concepts than terminologies and may lack concept codes. To be used as vocabularies in HL7, taxonomies need to have concept codes and must contain concepts at an adequate level of granularity. Like hierarchical terminologies, however, taxonomies often specify hierarchical parent-child relationships