HL7 Common Terminology Services

Editor/Principal Contributor	Harold Solbrig solbrig@mayo.edu Mayo Clinic/Foundation
Principal Contributor	Tony Weida tweida@apelon.com Apelon, Inc.
Principal Contributor	Larry Streepy streepy@healthlanguage.com Health Language, Inc.
Contributor	David Markwell david@clincial-info.co.uk Clinical Information Consultancy
Contributor	Keith Campbell
Vocabulary Co-Chair/Contributor	Stanley Huff, M.D. stan.huff@ihc.com Intermountain Health Care
Vocabulary Co-Chair	Christopher Chute, M.D., Dr P.H. chute@mayo.edu Mayo Clinic/Foundation
Vocabulary Co-Chair	Ted Klein kci@tklein.com Klein Consulting, Inc.
Vocabulary Co-Chair	Paul Frosdick paul.frosdick@nhsia.nhs.uk NHS Information Authority

Table of Contents

1 Introduction

The Health Level Seven (HL7) Version 3 standards are based on a Reference Information Model (RIM) which is flexible and general in structure. Representation of information within this model is dependent on the availability of terminological resources which can be used to populate the properties of the model with appropriate semantic content. Whenever possible, the HL7 Version 3 standard references existing terminological resources instead of attempting to create a new resource within the standard itself.

These external terminological resources can vary considerably in both content and structure. The HL7 standard needs to be able to identify a minimum set of characteristics that any terminological resource must posses if it is to be used in a HL7 messaging environment. One approach to this task would be to specify a common data structure which all terminological resources would have to fit. This approach, however, is not without drawbacks. First, a common data structure would have to represent a 'least common denominator', which could mask more advanced content and functional characteristics that might be particular to a specific terminology. Another drawback is that this approach puts much of the responsibility for maintaining and updating the content on the HL7 standards body rather than the individual terminology developers.

The Common Terminology Services (CTS) specification was developed as an alternative to a common data structure. Instead of specifying what an external terminology must look like, HL7 has chosen to identify the common functional characteristics that an external terminology must be able to provide. As an example, an HL7 compliant terminology service will need to be able to determine whether a given concept code is valid within the particular resource. Instead of describing a table keyed by the resource identifier and concept code, the CTS specification describes an Application Programming Interface (API) call that takes a resource identifier and concept code as input and returns a true/false value. Each terminology developer is free to implement this API call in whatever way is most appropriate for them.

This document describes a set of API calls that represent the core functionality that will be needed by basic HL7 Version 3 applications.

1.1 Revision History

- **06/20/2005** - Added note that first supportedLanguage is the default
- **06/20/2005** - Added description of default behavior for lookupDesignation if language is blank or null
- **04/12/2005** - Minor typo corrections
- **11/28/2004** - Incorporated results of San Antonio ballot (Spring, 2004)
- **11/26/2004** - Minor typos, etc. as submitted by Tony Weida (Apelon)
- **03/12/2004**- Incorporated results of San Diego ballot (Winter, 2003)
- **11/01/2003** - Final committee draft - released for ballot.

2 Scope and Design of the HL7 Common Terminology Server

2.1 Outline

The HL7 Common Terminology Services (HL7 CTS) defines an Application Programming Interface (API) that can be used by HL7 Version 3 software when accessing terminological content. Before proceeding, we need to first state some things that the CTS specification is not designed to do.

- The current version of CTS is not intended to be a complete terminology service. The scope of CTS is restricted to the functionality needed to design, implement and deploy an HL7 Version 3 compliant software package. In much the same spirit as the XML/SGML relationship, the HL7 CTS is meant to represent a proper subset of functionality that may be provided by more sophisticated APIs such as that represented by the OMG TQS specification.
- CTS is not intended to be a general purpose query language. It is intended to specify only the specific services needed in the HL7 implementation.
- CTS does not specify how the service is to be implemented. It is intentionally silent when it comes to service advertising and discovery, establishing and maintaining connections, and the delivery and routing of messages. It is presumed that a CTS implementation will use the underlying architecture that is most appropriate for the given implementation circumstances.

2.2 Design Principals

The following general principles were used while developing the HL7 CTS specification:

1. It shall be easy to write programs that use HL7 CTS.
2. The intent of the HL7 CTS is to specify core services only.
3. The design of HL7 CTS shall be formal and concise.
4. The initial implementation technology for HL7 CTS will include XML.
5. HL7 CTS shall be compatible with the nomenclature, model and approach expressed in the HL7 Vocabulary document, the Version 3 RIM and its derivative structures.
6. Whenever possible, the HL7 CTS shall remain a consistent subset of the Object Management Group (OMG) Terminology Query Services (TQS) provided that the TQS terminology model does not conflict with other HL7 CTS design principles. If it is discovered that the TQS model is conflicting with HL7 CTS design principles or is incomplete, or incorrect, good faith efforts should be made to notify the appropriate revision task force.
7. HL7 CTS should limit the assumptions about the form and structure of a terminology to those necessary to support HL7 implementations.

2.3 Other Relevant Work

There is no generally accepted standard for terminology services but there are several sources of material on this topic.

- **The OMG Terminology Query Services (TQS) specification**
  TQS specifies a full terminology service, but is not widely implemented and vendor support is minimal. The specification is considered by some to be too "heavyweight" and also assumes a particular technical solution (CORBA). Since reliance on the TQS standard is not an assumption of the HL7 standard, a more general approach to terminology services is needed - at least to cover those areas in which HL7 is terminology dependent.
- **The DAML + OIL and OWL Web Ontology Language**
  While this is not strictly a terminology server specification it contains elements from the representation of ontological aspects that are relevant to some of the large scale terminologies such as SNOMED Clinical Terms, NHS Clinical Terms Version 3 and GALEN. However, this web based proposal is also a heavy weight specification that is unlikely to facilitate early widespread implementation.
- **API specifications specific to individual terminologies**
  An example of this is the "Read Code - Version 3 API" specified for the NHS Clinical Terms in 1996 and revised in 1998. Work is proceeding to develop a SNOMED CT API based on similar principles. There is an informal understanding that this specific API will converge with or utlize elements of the CTS where tese are appropriate. A COM implementation of this is supported by at least one freeware coding engine (CLUE). Specifications of this type identify many of the general functions needed to access a terminology. However, they are inevitably specific to the needs of a particular terminology. Explicit support for a single defined terminology model allows efficient implementation within an operational environment at the expense of the flexibility to access other terminologies.
- **General purpose RDBS interfaces including SQL and ODBC**
  For "simple" code lists, a simple SQL query may be the most efficient way to look up a code. In addition to the code-value / code-meaning pairs, however, many coding schemes have other relevant properties that may only be accessed through a secondary service. This does not prevent the use of SQL but it requires a common model against which queries can be run and an efficient means of returning the properties required. These additional properties apply to the scheme as a whole as well as to individual entries in the terminology.
- **Terminology Query Language (TQL)**
  TQL, formerly based on an SQL-like syntax, today implements as a URI-based language specific to terminology servers designed by Michael Hogarth and colleagues from the University of California. TQL provides a rich mechanism that deals specifically with common properties and relationships in terminological models. A working Java-based implementation of TQL can be downloaded for free from the web.

2.4 Revision History and Versions

The version of the CTS API described in this document does very little when it comes to handling revisions and revision history. Much of this effort has been deliberately postponed until a subsequent release. The approach that we are taking is to get a functional API out and in place and then enhance and revise it as needed.

2.5 Stateful and Session Extensions

The CTS specification has been designed in a way that allows stateless/session independent operations. While this allows a wider variety of implementations, it does have the potential to negatively impact the performance of systems that could take advantage of previously fetched references to value sets, concepts, etc. It is not the intent of this specification to preclude or prohibit such implementations. Vendors who extend the CTS API to allow stateful and/or session-based implementations are strongly encouraged to submit the extensions to the HL7 Vocabulary Technical Committee for inclusion in a subsequent version of this specification.

3 CTS Modules

3.1 The Message and Vocabulary API

There are two distinct layers between HL7 Version 3 message processing applications and the target vocabularies. The upper layer, the **Message API**, communicates with the messaging software, and does so in terms of vocabulary domains, contexts, value sets, coded attributes and other artifacts of the HL7 message model. The lower layer, the **Vocabulary API**, communicates with the terminology service software, and does so in terms of code systems, concept codes, designations, relationships and other terminology specific entities.

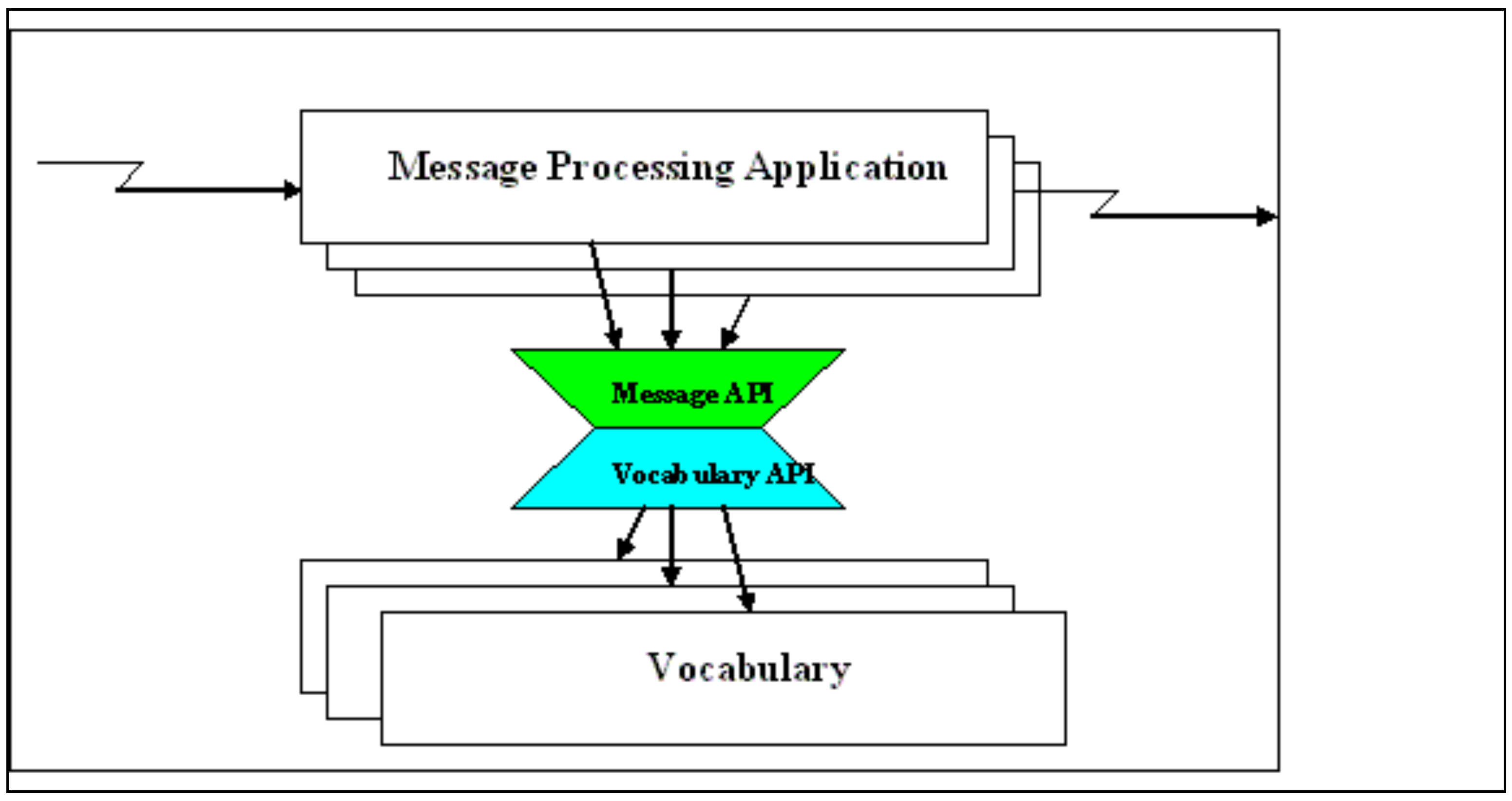
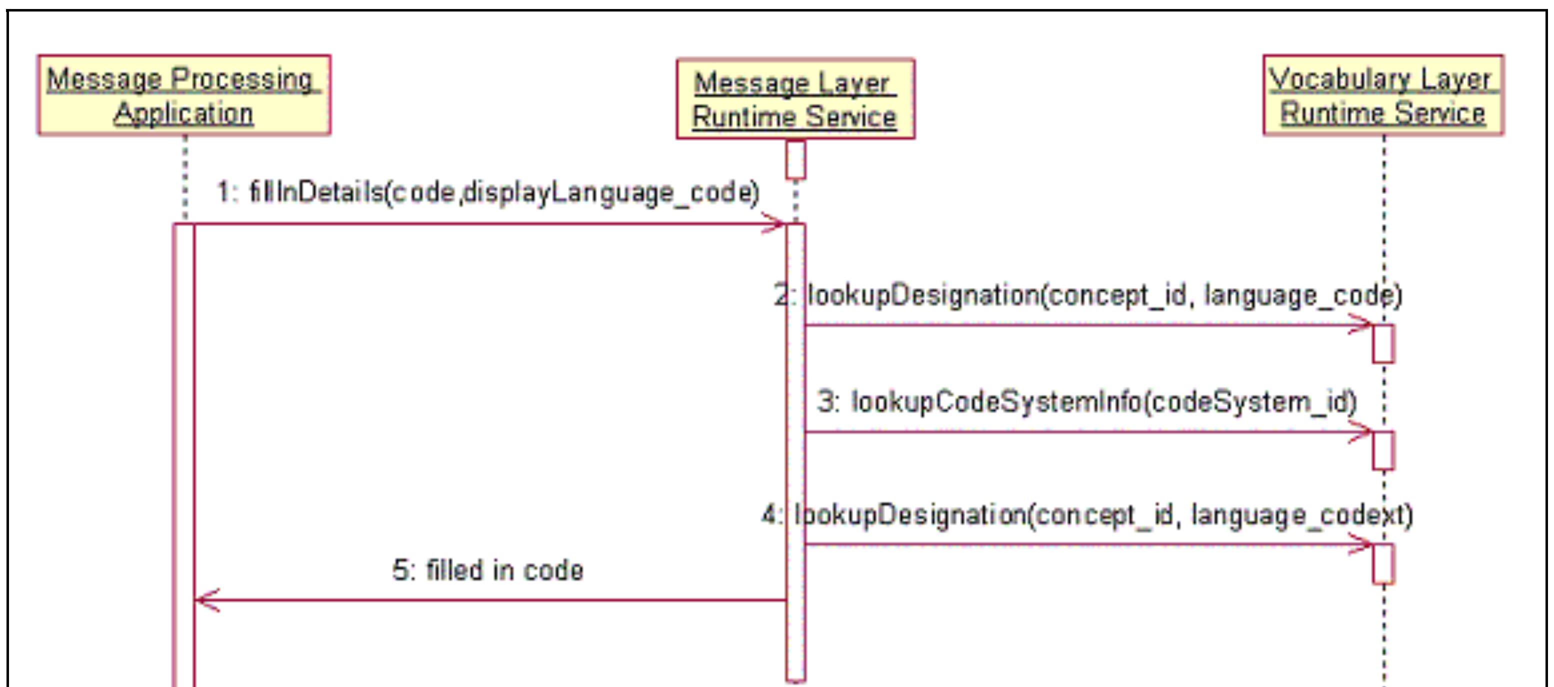


Figure 1: Message and Vocabulary APIs

The Message API is specific to HL7. Its primary purpose is to allow a wide variety of message processing applications to create, validate and translate CD-derived data types in a consistent and reproducible fashion.

The Vocabulary API is intended to be generic[1]. It allows applications to query different terminologies[2] in a consistent, well-defined fashion. The Message API utilizes the Vocabulary API. The following figure shows an example of a Message/Vocabulary interaction sequence. In this example, an application requests that the Message Runtime Service fill out the details of a CD-derived attribute. The service, in turn, makes multiple calls to the Vocabulary API service in order to get concept code designations, code system names, release versions, etc.


Message Processing Application
Message Layer Runtime Service
Vocabulary Layer Runtime Service
1: fillInDetails(code,displayLanguage_code)
2: lookupDesignation(concept_id, language_code)
3: lookupCodeSystemInfo(codeSystem_id)
4: lookupDesignation(concept_id, language_codext)
5: filled in code

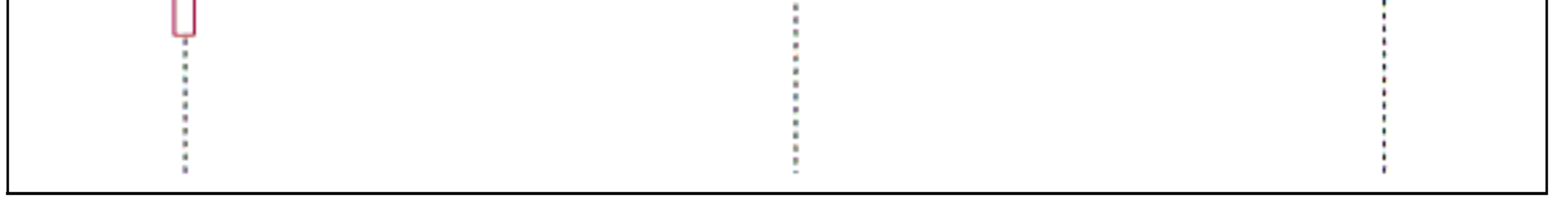


Figure 2: Sample Message/Vocabulary API interaction

3.2 Runtime and Browsing Functions

3.2.1 General HL7 CTS User (Actor) Classes

The classes of actors that are anticipated to be users of the HL7 CTS API include:

- **Message Creation Software** – Software that is involved in the creation of HL7 messages. From a vocabulary perspective, this process involves the translation of internal messages and data into the syntax and semantics of the HL7 Version 3 standard.
- **Message Processing Software** – Software that receives, decodes and acts on the content of standard HL7 Version 3 messages. This process may include validation, translation and inferencing steps.
- **RIM Modelers** – The combination of people and tools that create and define HL7 Message content.
- **Software Developers** – The people who build the software that creates, validates and processes HL7 Version 3 messages.
- **Vocabulary Translators** – A combination of tools and people that translate the abstract HL7 Version 3 specification into the structure and terms of actual data processing applications.

3.2.2 The Divided Requirements Profile

The first two actor classes – message creation and message processing – have a different requirements profile than do the succeeding three:

- Performance – High throughput and scalability are paramount for message creation and processing. Modeling and mapping is considerably more tolerant of variable and potentially sub-optimal response times.
- Reliability – Message creation and processing software requires highly reliable software, while the modeling and authoring environment can tolerate some degree of unreliability and occasional failures.
- Functionality – Once solidified, the functional requirements of message creation and processing software will remain fixed. Any additional capabilities beyond those requirements will not be used. Authoring and modeling, however, will continue to create new and different browsing and viewing demands, and any (useful) functional capabilities may be drawn upon at random times.

Throughout the rest of this document, the message creation and processing profile will be referred to as the Runtime profile and the authoring and browsing as the Browser profile.

3.3 Translation

One additional functional area still needs to be specified – concept code translation between different code systems. The ability to translate between code sets for different realms is an integral part of the messaging API. It is specified as a separate interface at the vocabulary layer because translation is not specific to a single vocabulary. Translation services have the potential to be developed independently from one or more of the terminologies included in the translation process.

3.4 Individual Specification Components

The message/vocabulary split can be combined with the two requirements profiles to yield five separate modules:

	Runtime	**Browser**
Message API	Message Runtime	Message Browser
Vocabulary API	Vocabulary Runtime	Vocabulary Browser
Translation	Vocabulary Translation	

Table 1: Specification Components

This specification is written in a fashion that should make it possible for each of these areas to be developed independently and still interoperate. Some terminology developers may wish to focus exclusively on the Vocabulary API, while some developers may only implement the Runtime API. Theoretically, the Message API would only need to be implemented once, as the underlying HL7 structure is public and available to everyone. In practice, however, it may be necessary to more tightly couple the Message Runtime with the Vocabulary Runtime to achieve the desired performance.

4 Synopsis of Module Functions

The following sections contain a brief synopsis of the functionality of each of the five modules described above. Some parameters and utility functions have been omitted for the sake of clarity. This is strictly a high level overview and all of the functions below will be described in considerably more detail later in this document.

4.1 Message Layer Runtime Functions

The Message Layer Runtime Module describes the services used by the message creation, processing and translation software.

Function	**Inputs**	**Outputs**	**Description**
getServiceName		Service name	Return the name that was assigned to this service by the service provider.
		Version	Return the current

getServiceVersion		Version identifier	version of the service software.
getServiceDescription		Service description	Return a description of the service function, authors, copyrights, etc.
getHL7ReleaseVersion		Version identifier	Return the HL7 release version that is currently supported by this service.
getCTSVersion		Major and minor version number	Return the CTS version that this service implements.
getSupportedMatchAlgorithms		List of match algorithms	Return a list of string match algorithms implemented by this service.
getSupportedVocabularyDomains	Match text and algorithm, time limit and size limit	List of vocabulary domain names	Return a list of the vocabulary domains matching the supplied match text that are recognized by this service.
validateCode	Name of the vocabulary domain, code to be validated, application context(realm), flag indicating whether to validate active concepts only and flag indicating whether to check both errors and warnings or just errors	List of errors and warnings.	Validate the coded attribute for the supplied vocabulary domain and context.
validateTranslation	Name of the vocabulary domain, coded attribute containing translation(s) to be validated, application context(realm), flag indicating whether to validate active concepts only and flag indicating whether to check both errors and warnings or just errors	List of errors and warnings.	Validate the CD translations, if any, for the supplied vocabulary domain and context.
translateCode	Name of the vocabulary domain, coded attribute to be translated, target coding system, and target application context(realm)	Translation of the coded attribute.	Translate the supplied coded attribute into a form that uses the target code system or uses whatever code system is appropriate for the supplied context
fillInDetails	Coded attribute and target language code	Coded attribute value with details supplied.	Fill in the optional parts of the coded attribute such as the concept display name, the code system name and code system version.
subsumes	Parent coded attribute, child coded attribute	True / False	Determine whether the parent coded attribute subsumes (or implies) the child.
areEquivalent	First coded attribute, second coded attribute	True / False	Determine whether the two coded attributes are 'equivalent'.
lookupValueSetExpansion	Name of the vocabulary domain, application context(realm), language for expansion text, flag indicating whether to do a complete expansion of just one level, time limit and size limit	Hierarchical expansion of the value set associated with the domain in the supplied context	Return a hierarchical list of selectable concepts for the supplied vocabulary domain and context.
expandValueSetExpansionContext	Opaque expansion context returned from previous lookupValueSetExpansion or expandValueSetExpansionContext call	Further hierarchical expansion of the value set associated with the domain in the supplied context	Return further expansion on nested value set contents.

Table 2: Message Layer Runtime Functions

4.2 Vocabulary Layer Runtime Functions

This set of functions is used by the Message Layer Runtime and Message Layer Browser services as well as the Vocabulary Layer Browser service. These functions can also be used in a stand-alone fashion.

service. These functions can also be used in a stand-alone fashion.

Function	Input	Output	Description
getServiceName		Service name	Return the name that was assigned to this service by the service provider.
getServiceVersion		Version identifier	Return the current version of the service software.
getServiceDescription		Service description	Return a description of the service function, authors, copyrights, etc.
getCTSVersion		Major and minor version number	Return the CTS version that this service implements.
getSupportedCodeSystems	Time limit and size limit	List of code systems and versions supported by the service implementation.	Return the identifier, name and release versions of all code systems that are supported by the service.
lookupCodeSystemInfo	Code system name or identifier	Description of the code system including name, id, description, version, supported languages, supported relations, supported properties, etc.	Return detailed information about a specific code system.
isConceptIdValid	Code system identifier, concept code and flag indicating whether inactive concepts are considered valid	True / False	Determine whether concept code is currently valid in the specified code system
lookupDesignation	Code system identifier, concept code and target language code	Designation text	Return the preferred designation for the concept code in the supplied language
areCodesRelated	Code system identifier, source concept code, target concept code, relationship code, relationship qualifiers. and flag indicating whether to use only directly related codes or the transitive closure of the relationship	True/False	Determine whether the named relationship exists between the source and target codes.

Table 3: Vocabulary Layer Runtime Functions

4.3 Code Mapping Functions

Function	Input	Output	Description
getServiceName		Service name	Return the name that was assigned to this service by the service provider.
getServiceVersion		Version identifier	Return the current version of the service software.
getServiceDescription		Service description	Return a description of the service function, authors, copyrights, etc.
getCTSVersion		Major and minor version number	Return the CTS version that this service implements.
getSupportedMaps		List of named sets consisting of from code system id, name and version, to code system id, name, and version and a mapping description	Return a list of mappings that are supported by this service.
mapConceptCode	Source code system identifier and concept code, target code system identifier and name of mapping resource	Corresponding concept code in target system and quality indicator	Return the mapping of the supplied concept code from the source code system to the target code system using the named mapping resource.

Table 4: Code Mapping Functions

4.4 Message Layer Browsing Functions

Function	Input	Output	Description
getServiceName		Service name	Return the name that was assigned to this service by the service provider.
getServiceVersion		Version identifier	Return the current version of the service software.
getServiceDescription		Service description	Return a description of the service function, authors, copyrights, etc.
getHL7ReleaseVersion		Version identifier	Return the HL7 release version that is currently supported by this service.
getCTSVersion		Major and minor version number	Return the CTS version that this service implements.
getSupportedMatchAlgorithms		List of string match algorithms implemented by the browser service	
getSupportedAttributes	Match text and algorithm, time limit and size limit	List of RIM attributes known to the browser	Returns a list of RIM attributes whose name matches the supplied match text that are known to the browser.
getSupportedVocabularyDomains	Match text and algorithm, time limit and size limit	List of vocabulary domains known to the browser	Returns a list of vocabulary domains whose name matches the supplied match text that are known to the browser.
getSupportedValueSets	Match text and algorithm, time limit and size limit	List of value sets known to the browser	Returns a list of value sets whose name matches the supplied match text that are known to the browser.
getSupportedCodeSystems	Match text and algorithm, time limit and size limit	List of code systems known to the browser	Returns a list of code systems whose name matches the supplied match text that are known to the browser.
lookupVocabularyDomain	Name of vocabulary domain	Domain name, description, domains restricted by this domain, list of RIM attributes that use this domain, and list of value sets that represent this domain	Look up all of the information known about the supplied vocabulary domain
lookupValueSet	Value set name or identifier	Detailed value set description, including name, identifier, description, list of value sets used to construct the set, value sets that this set helps define, list of concept codes the value set references, etc.	Look up detailed information on a value set (including vocabulary domains, constructors, etc).
lookupCodeSystem	Code system name or identifier	Name, id, copyright, release and registration information	Look up details on a code system
lookupValueSetForDomain	Name of vocabulary domain and application context(realm)	Name and id of the value set used for this vocabulary domain	Return the identifier of the value set that would be used for the vocabulary in the supplied context (if any).
isCodeInValueSet	Value set name or identifier, code system identifier and concept code, and indicator whether to include the "head code" as part of	True/False	Determine whether the supplied concept code is a valid value in the supplied value set

	the value set		

Table 5: Message Layer Browsing Functions

4.5 Vocabulary Layer Browsing Functions

Function	Input	Output	Description
getServiceName		Service name	Return the name that was assigned to this service by the service provider.
getServiceVersion		Version identifier	Return the current version of the service software.
getServiceDescription		Service description	Return a description of the service function, authors, copyrights, etc.
getCTSVersion		Major and minor version number	Return the CTS version that this service implements.
getSupportedMatchAlgorithms		List of string match algorithms implemented by the browser service	
getSupportedCodeSystems	Time limit and size limit	List of supported code systems and their descriptions	
lookupConceptCodesByDesignation	Code system identifier, match text and algorithm, target language code, flag indicating whether non-active concepts should be retrieved, time limit and size limit	List of code system identifiers and concept codes.	Return a list of concept codes that have designations that match the supplied match string in the supplied language, if any.
lookupConceptCodesByProperty	Code system identifier, match text and algorithm, target language code, flag indicating whether non-active concepts whould be retrieved, optional list of property mime types, time limit and size limit	List of code system id / concept codes.	Return a list of concept codes that have properties that meet the supplied criteria
lookupCompleteCodedConcept	Code system identifier and concept code	Everything that is known about the concept (designations, properties, relationships, etc.)	Return a complete description of the supplied concept code
lookupDesignations	Code system id and concept code, match text and algorithm, target language	List of designations	Return all designations for the supplied concept code that match the supplied criteria.
lookupProperties	Code system id and concept code, match text and algorithm, list of property codes to search, list of mime types to match and target language code	List of concept properties (property code, value, language, mime type)	Return the properties of the given code system id / concept code that match the supplied criteria.
lookupCodeExpansion	Code system id and concept code, relationship code, relationship direction indicator, target languag code, size limit and time limit	Hierarchical code expansion list	Recursively list the concept codes that are related to the supplied concept, including the preferred designation for the codes.

Table 6: Vocabulary Layer Browsing Functions

5 The CTS Message API Model

5.1 Introduction

This section describes the model that underlies the *CTS Message API*. It describes the relationship between HL7 coded attributes and vocabulary. It is based on the HL7 V3 Meta-Model Version 1.16 and, with the exception of the coloring scheme, attempts to remain consistent with its notation and data types. This document does not provide a complete or in-depth model of all the logical entities that might comprise a coding system[3]. Its primary purpose is to describe the classes and relationships that have a direct bearing on the contents of HL7 coded attributes from the perspective of a meta-model.

The Vocabulary API model describes the CTS model from vocabulary perspective and provides more detail about concept codes, designations, relationships, etc.

5.1.1 Notation

In the diagrams that follow, the classes whose instances are solely the responsibility of the HL7 Modeling Groups are colored green and classes that represent content that is managed by either HL7 or a third party terminology provider are colored blue. Existing meta-model classes will be colored a pale yellow. The authors of this document have little or no control over the content and structure of these existing classes – they are in the model solely as reference points.

The descriptions that follow the model use a semi-formal notation based on the model content :

- Class names will be represented in **boldface** (e.g. **VocabularyDomain**, **ValueSet**)
- Attribute names will appear in *italics* (e.g. *vocabularyDomain_name*, *valueSet_id*)
- Relationships will be underlined (e.g. a **VocabularyDomain** is represented by zero or more **VocabularyDomainValueSets**)

Where appropriate, relationship cardinality will be expressed by the forms below or something similar:

- 1..1 "**Class 1** must relationship with exactly one **Class 2** "
- 0..1 "**Class 1** may be relationship by at most one **Class 2** "
- 1..* "**Class 1** must be relationship by one or more **Class 2** "
- 0..* "**Class 1** may be relationship by zero or more **Class 2** "

5.2 Vocabulary Domain

A vocabulary domain serves as the link between an HL7 coded attribute and the set(s) of valid concept codes for that attribute. A vocabulary domain represents an abstract conceptual space such as "countries of the world", "the gender of a person used for administrative purposes", etc.

The figure below below shows the relationship between vocabulary domains and HL7 attributes.

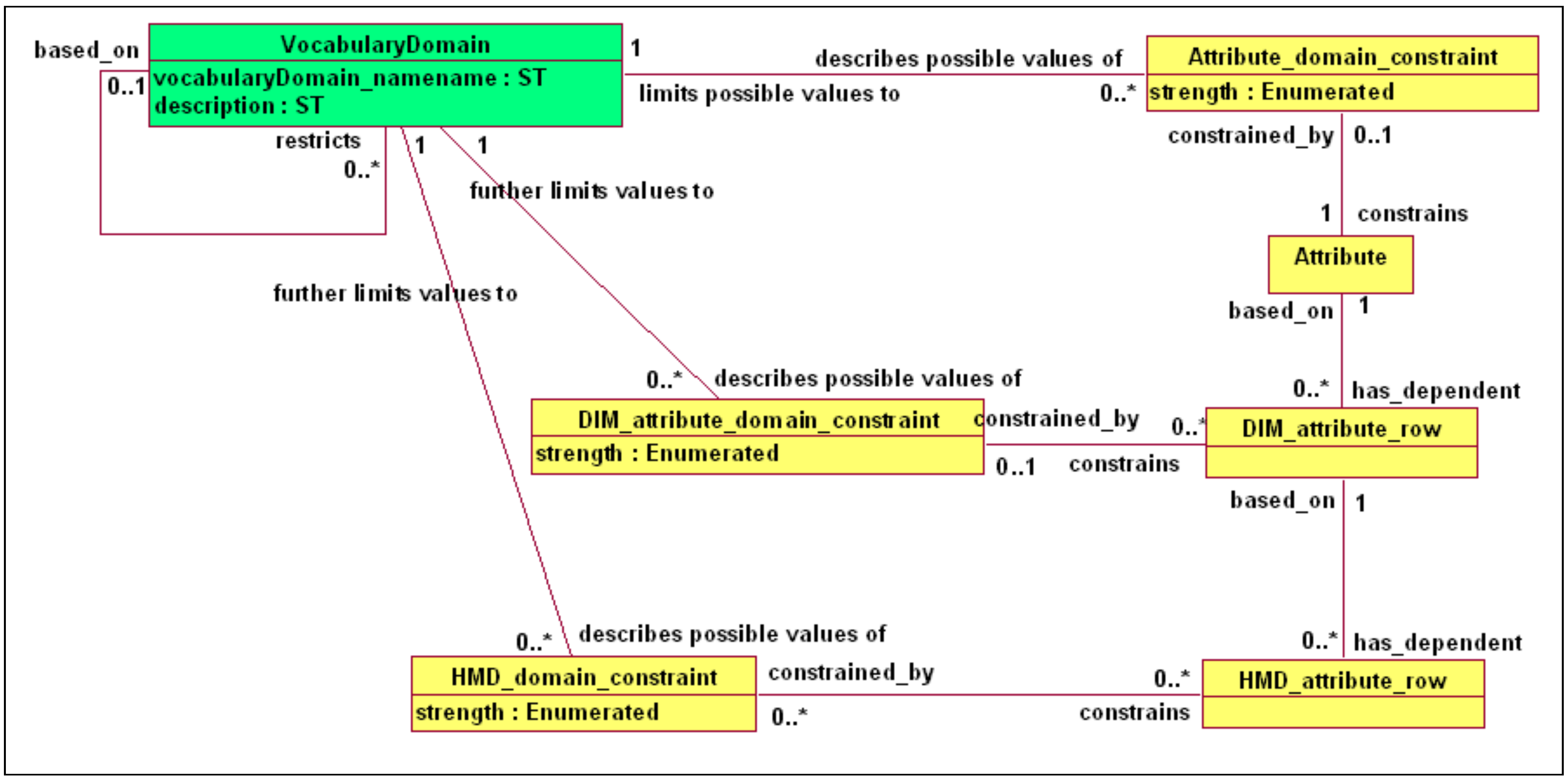


Figure 3: RIM Attributes and Vocabulary Domains

An **Attribute_domain_constraint** must constrain the possible values of exactly one RIM **attribute** by limiting the possible values to those described by exactly one **VocabularyDomain**. A **VocabularyDomain** may describe the possible values of zero or more

Attribute_domain_constraints. A **DIM_attribute_row** must be based on exactly one **Attribute** from the HL7 model, and serves to express the presence of the attribute in a specific design information model (DIM). **DIM_attribute_rows** inherit all of the constraints of the **Attribute** that they are based on, and **DIM_attribute_row** may be constrained by zero or more **DIM_attribute_domain_constraints**.

A hierarchical message description (HMD) completely defines the structure of a set of messages. **HMD_attribute_rows** are a part of a HMD, and each HMD_attribute_row is based on exactly one **DIM_attribute_row** . An **HMD_attribute_row** inherits all of the constraints of the corresponding **DIM_attribute_row** and may be yet further constrained by at most one **HMD_domain_constraint**.

Each **VocabularyDomain** has a unique *name* along with a *description* of the conceptual space that it represents. **VocabularyDomains** that describe the possible values of **DIM_attribute_domain_constraints** or **HMD_domain_constraints** restrict the conceptual space of the corresponding **DIM_attribute_domain** or **Attribute_domain_constraint** upon which the domain is based.

The example below shows how vocabulary domains might be constrained between the attribute, the DIM attribute row and the HMD attribute row.

Attribute	Vocabulary Domain	Description	Restricts Domain
sourceCountry	Country	A country of the world	-
sourceCountry(DIM)	HL7MemberCountry	Any country that is an official member of HL7	Country
sourceCountry(HMD)	EUHL7MemberCountry	Any EU country that is an official member of HL7	HL7MemberCountry

Table 7: Sample Vocabulary Domain Restriction

5.2.1 Additional Constraints [4]

- An **Attribute** must be constrained by exactly one **Attribute_domain_constraint** if and only if the *attribute type* is "coded".
- Any **VocabularyDomain** that describes the possible values of a DIM_attribute_domain_constraint must be based on the **VocabularyDomain** of the corresponding **Attribute**.
- Any **VocabularyDomain** that describes the possible values of a **RIM_attribute_domain_constraint** must be based on the **VocabularyDomain** of the corresponding **DIM_attribute_row** if one exists and on the **VocabularyDomain** of the corresponding Attribute if one doesn't.

5.3 Value Set

A vocabulary domain describes a 'conceptual space' from which the values of an attribute can be drawn. Before an attribute can be used in a message, however, the actual list of concept codes needs to be defined. A list of valid concept codes is referred to as a value set.

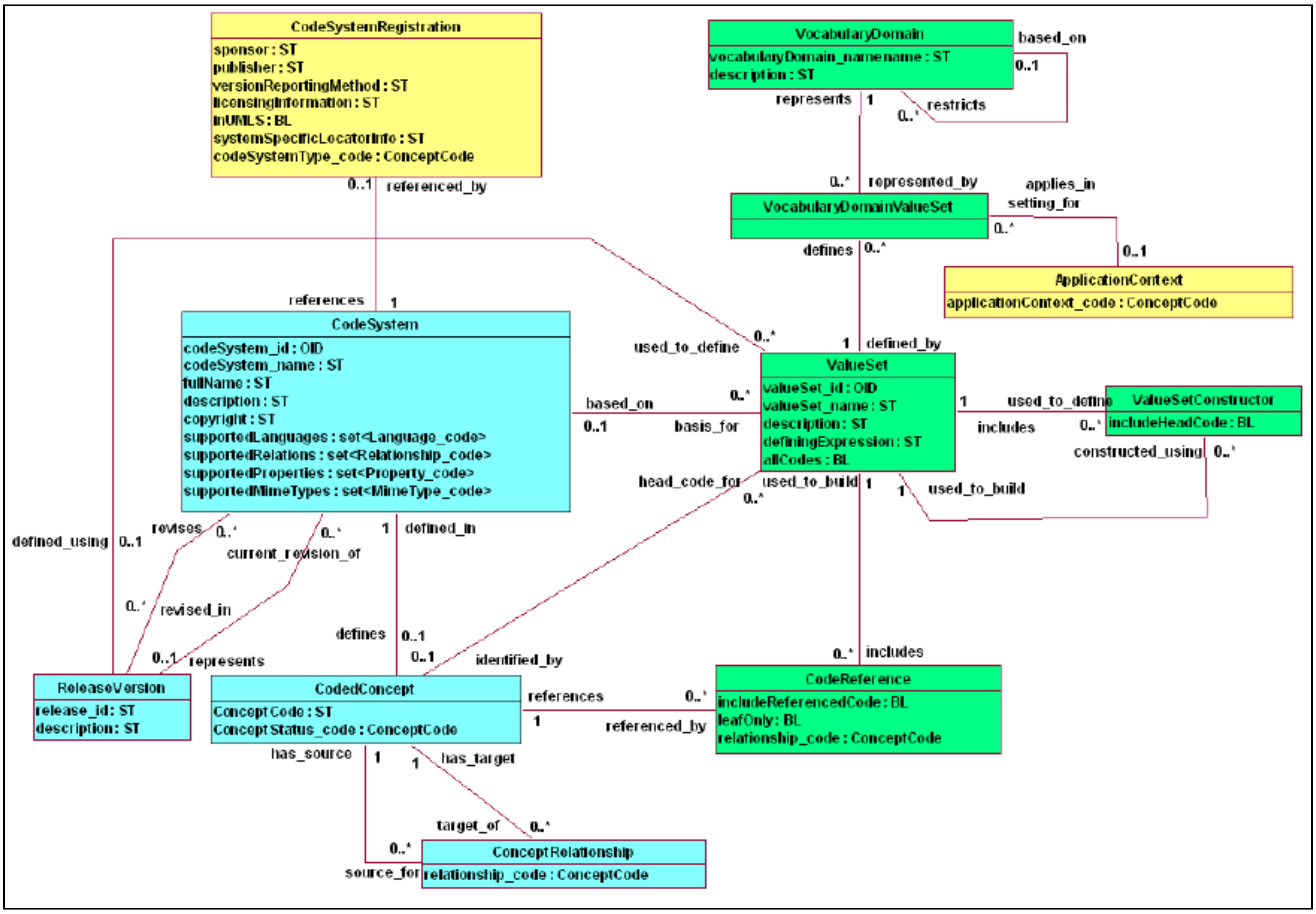


Figure 4: Value Sets

A **VocabularyDomain** may be represented by zero or more **ValueSets**. While it is possible for abstract RIM and DIM attributes to not be represented by any **ValueSets** at all, **VocabularyDomains** that describe the possible values of Coded **Attributes** used in an actual messages must be represented by at least one **Value_set**.

5.3.1 Linking Vocabulary Domains to Value sets

A **VocabularyDomainValueSet** represents an association between exactly one **VocabularyDomain** and exactly one **ValueSet**. Each association between a **VocabularyDomain** and a **ValueSet** may apply in zero or one **ApplicationContexts**. An **ApplicationContext** names a specific geopolitical entity (e.g. EU, Canada) and/or practice setting (e.g. veterinary medicine, public health), etc. and may be a setting for zero or more **VocabularyDomainValueSet** associations.

5.3.2 Defining Value sets

A **ValueSet** may include a list of zero or more **CodedConcepts** drawn from a single **CodeSystem**. A **ValueSet** can represent:

- All of the **CodedConcepts** defined in exactly one **CodeSystem**
- A specified list of **CodedConcepts** that are defined in exactly one **CodeSystem**
- The set of **CodedConcepts** represented by another **ValueSet**.

The details about each of these forms will be described in the next section. First we describe the characteristics that all value sets have in common.

It is anticipated that there will be far too many value sets to be able to assign a unique mnemonic or meaningful name to all of them. The primary identifier for a ValueSet is a meaningless numeric identifier, the valueSet_id[5]. The valueSet_name can also contain a unique 'meaningful' identifier where appropriate. The valueSet_name is designed for communication between carbon-based life forms.

A **ValueSet** has a *description* that describes the intent and purpose of the set. It also has a *definingExpression*, which is intended to carry a formal machine readable expression that can be used to construct the **ValueSet**. The meaning and interpretation of *definingExpression* is not part of this specification. Both *description* and *definingExpression* are optional. The **ValueSet** attribute, *allCodes*, is described below.

5.3.3 Defining Value Set Content

A **ValueSet** must either be constructed using at least one other **ValueSet** or it must be based on exactly one **CodeSystem** or both[6]. A **CodeSystem** defines a set of zero or more **CodedConcepts** which, in turn, signify relevant classes or entities within a particular domain of interest. **CodeSystems** can range from a simple table of genders to classification systems such as ICD-9 to rich, description-logic based systems such as OpenGalen or SNOMED-CT. As many **CodeSystems** undergo periodic revision, it is useful to record the **ReleaseVersion** that was used to define a given **ValueSet** at a particular point in time. A **ValueSet** may be defined using at most one **ReleaseVersion**. A **ReleaseVersion** may be used to define zero or more **ValueSets**. The defined using relationship is strictly informative, and services may use a later revision of the code system to represent the **ValueSet** content.[7]Code systems will be discussed in considerably more detail in a later section describing the CTS Vocabulary API.

5.3.3.1 Representing the entire contents of a code system

A **ValueSet** can be defined to include all of the **CodedConcepts** defined in the **CodeSystem** upon which the **ValueSet** is based by setting the *allCodes* attribute to TRUE. A **ValueSet** with *allCodes* set to TRUE may not include any **CodeReferences**.

valueSet_id	valueSet_name	description	all Codes	CodeSystem	Release Version
2.16.840.1.113883.1.11.1	AdministrativeGender	The gender of a person used for administrative purposes (as opposed to clinical gender)	True	2.16.840.1.113883.5.1	5

Table 8: A value set representing an entire code system

5.3.3.2 Representing Selected Parts of a Code System

A **ValueSet** may include zero or more **CodedConcepts** defined in the **CodeSystem** on which the **ValueSet** is based. This is accomplished by including one or more **CodeReferences**. A **CodeReference** ties a **CodedConcept** to a **ValueSet**. It must reference exactly one **CodedConcept**. The *relation_code* attribute, however, implicitly references all of the **CodedConcepts** that are the target of the **ConceptRelationship** with the included**CodedConcept**. A **CodeReference** can include or exclude the referenced **CodedConcept** itself. It can include all of the target codes or only the leaf nodes. These various possibilities are reflected in the *includeReferencedCode*, *relationship_code* and *leafOnly* attributes described in the following table:

allCodes	***includeReferencedCode***	***relationship_code***	***leafOnly***	**Description**
True	-	-	-	Include all of the codes in the code system.
False	True	-	-	Include only the referenced concept code
False	True	hasSubtype	False	Include the referenced code and all of its subtypes
False	False	hasSubtype	False	Include all of the subtypes of the referenced code but not the referenced code itself.
False	False	hasSubtype	True	Include only the leaf subtypes of the referenced code

Table 9: Code Reference Attribute Options

Attribute combinations other than those described above are not valid. *relation_code* should be drawn from the HL7 RelationshipCode code system if possible.

5.3.4 Including other value sets in a value set

A **ValueSet** may also be constructed using zero or more additional **ValueSets**. Including a **ValueSet** means that the **CodedConcepts** represented by the included set are to be included as part of the containing set. **ValueSetConstructors** serve two purposes:

1. They allow **ValueSets** to be included by reference, meaning that changes in the contained set are automatically included in the container.
2. They make it possible to include **CodedConcepts** from more than one **CodeSystem** in a single **ValueSet**.

A **ValueSetConstructor** connects two **ValueSets** – the included set and the set being constructed. *includeHeadCode* determines whether the head code of the included set (if any) is to be included as part of the containing value set.

The table below shows examples of value set constructors. The value set "HL7ConformanceInclusion" includes the contents of the value set, "InclusionNotMandatory" which, in turn includes the set "InclusionNotRequired".

usedToBuildSet	**(name)**	**includedSet**	**(name)**	**Include Head Code**
2.16.840.1.113883.1.11.10010	HL7ConformanceInclusion	2.16.840.1.113883.1.11.10012	InclusionNotMandatory	False
2.16.840.1.113883.1.11.10012	InclusionNotMandatory	2.16.840.1.113883.1.11.10015	InclusionNotRequired	True

Table 10: Example Value Set Constructors

5.3.5 Head codes

A **ValueSet** references a set of **CodedConcepts**. Frequently, the association between a **ValueSet** and a collection of **CodedConcepts** implies a subsumption, partitive or other hierarchical relationship where the value set itself represents the 'whole'(parent) and the concept codes the individual 'parts' (children). When this is the case, there may also be a corresponding concept code in the code system itself that represents the same 'whole' or 'parent' concept. We refer to this corresponding code as the 'head code' of the associated value set. Many coded attributes in the HL7 RIM can have varying degrees of granularity. Using the example above, the InclusionNotRequired value set has a head code, "NR", and two subsidiary codes "Excluded (X) " and "Required may be empty (RE)". The second assignment line above indicates that the valid values for this value set may be "X" for excluded, "RE" for required, but may be empty and "NR" when the inclusion is not required for a non-specific reason.

When a value set is used to construct another value set, the head code may or may not be part of the included set. This provides the ability to represent what the HL7 community refers to as 'abstract' and 'specializable' value sets. A **ValueSet** may be identified by at most one **CodedConcept**, which is known as the "head code". A **CodedConcept** may be the head code for zero or more **ValueSets**.

5.4 Registered Code System

Many of the code systems used within HL7 will be supplied by outside parties. The **CodeSystem** class, defined later in this document, represents the characteristics common to all code systems used within the HL7 environment. HL7 also maintains an internal registry (metadata) about code systems themselves – a code system registry. This registry is intended to function as a central repository of metadata about any code system that may appear in an HL7 message, be it internal to a site or system or commonly used and sanctioned between systems.

Registration does not constitute 'sanctioning' from an HL7 standpoint. It simply records a reference. The registration process also assigns a code system identifier (OID) for a code system if one doesn't already exist.

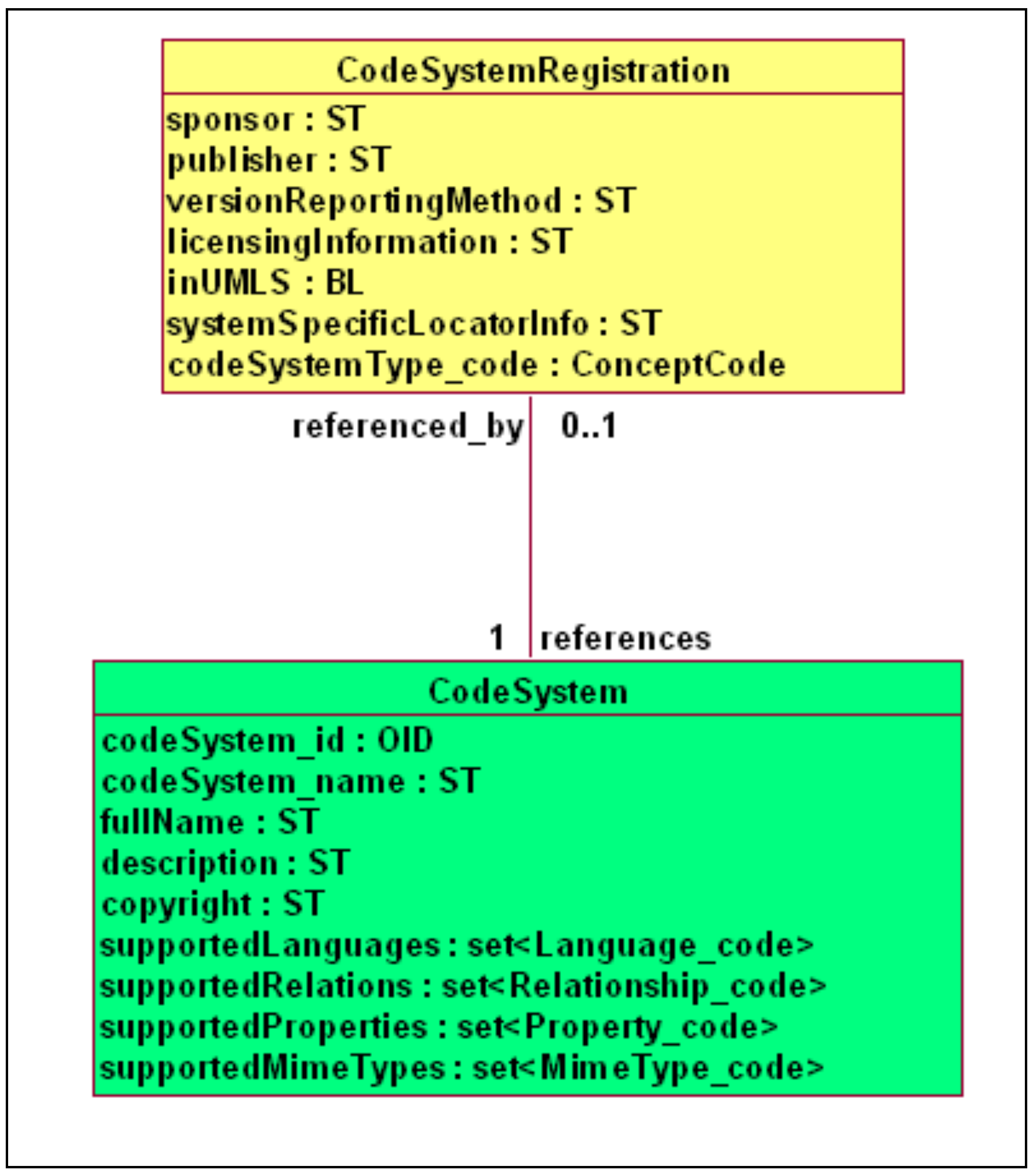

CodeSystemRegistration
sponsor : ST
publisher : ST
versionReportingMethod : ST
licensingInformation : ST
inUMLS : BL
systemSpecificLocatorInfo : ST
codeSystemType_code : ConceptCode
referenced_by
0..1
1
references
CodeSystem
codeSystem_id : OID
codeSystem_name : ST
fullName : ST
description : ST
copyright : ST
supportedLanguages : set<Language_code>
supportedRelations : set<Relationship_code>
supportedProperties : set<Property_code>
supportedMimeTypes : set<MimeType_code>

Figure 5: Registered Code System

A **CodeSystemRegistration** has the following attributes:

- *sponsor* – the name, address and the like of the person or organization that officially sponsors this code system within HL7
- *publisher* – the name, address and the like of the organization responsible for creating and maintaining the code system
- *versionReportingMethod* – a description of how and how often new versions are created and distributed
- *licensingInformation* – a description of the required licenses, costs and how they are acquired
- *inUMLS* – 'true' means that the coded terms defined in this code system are included in the Unified Medical Language System (UMLS)
- *systemSpecificLocatorInfo* – information specific to the code system publisher that serves to define or identify the specific coding system. Within HL7, systemSpecificLocatorInfo may sometimes used to identify the corresponding Version 2.x HL7 table where appropriate.
- *codeSystemType_code* – A code that identifies whether the code system is maintained and distributed internally by HL7 (I), is maintained and distributed by a third party(E) or is maintained by a third party, but is distributed by HL7 (EI).

6 The CTS Message API Specification

6.1 Introduction

The CTS Message API (CTSMAPI) is divided into two sections – the *runtime* section, which defines the set of functions that are necessary for everyday operation of HL7 Version 3 message software and the *browsing* section, which is used for authoring and construction of HL7 messages. Software using the browsing section is presumed to have access to a corresponding runtime section, whereas software in the runtime section may not have access to a browsing section.

6.2 Common CTS Message Elements

6.2.1 Basic Data Elements

The following section describes the basic types that are used in the CTS Message API. Types with the prefix "types::" are all based on the HL7 Version 3 Data Types and are not described further here.

```
struct CTSVersionId {
        types::INT      major;
        types::INT      minor;
};

typedef types::UID      CodeSystemId;

typedef types::ST       CodeSystemName;

typedef types::ST       ConceptCode;

typedef types::ST       VocabularyDomainName;
typedef sequence<VocabularyDomainName> VocabularyDomainNameList;

typedef types::UID      ValueSetId;

typedef types::ST       ValueSetName;

struct ConceptId {
        CodeSystemId    codeSystem_id;
        ConceptCode     concept_code;
};

typedef types::ST ReleaseVersionId;
typedef sequence<ReleaseVersionId> ReleaseVersionIdList;

typedef types::bin_value ExpansionContext;
```

Listing 1: Basic CTS Message API Types

- **CTSVersionId** - a structure that contains the major and minor version of the CTS Specification. The current version of the specification would be major:1, minor:0 (1.0)
- **CodeSystemId** - A unique code system identifier. In the HL7 context, this should be the ISO Object Identifier (OID) assigned by HL7 if one exists. Other identifiers such as the DCE UUID, etc. may be used as code system identifiers in non-HL7 settings. In these situations is the responsibility of the implementor to reconcile any namespace conflicts that may arise between OID's and the other identifiers..
- **CodeSystemName** - the name of a code system. Both code system id and code system names are unique within the HL7 Version 3 namespace, but this is not guaranteed to be universally true.
- **ConceptCode** - a code that uniquely represents a class or concept within a given code system.
- **VocabularyDomainName** - the unique name of an HL7 vocabulary domain
- **ReleaseVersionId** - an identifier that uniquely names a release / version within the context of a code system.
- **ValueSetId** - an ISO Object Identifier (OID) that uniquely identifies a HL7 value set.
- **ValueSetName** - the unique name or mnemonic for a value set. Not all value sets will have both ids and names, but when they do, both must be unique.
- **ConceptId** - the combination of a code system identifier and a concept code which together provide a globally unique name for a concept
- **ReleaseVersionId** - the unique identifier of a version or release of one or more code systems
- **ExpansionContext** – an opaque blob used to pass contextual information between the server and client

6.2.2 Concept Codes

The Message API uses a number of coded attributes which are described below.

```
typedef ConceptCode LanguageCode;

typedef ConceptCode RelationshipCode;

typedef ConceptCode ApplicationContextCode;

typedef ConceptCode DataTypeCode;

typedef ConceptCode CodingStrengthCode;

typedef ConceptCode ValueSetNodeTypeCode;

typedef ConceptCode CodeSystemTypeCode;

typedef ConceptCode MatchAlgorithmCode;
typedef sequence<MatchAlgorithmCode> MatchAlgorithmCodeList;
```

Listing 2: Coded Concepts used in the Message API

- **LanguageCode** – a code for a spoken or written language that follows the rules described in IETF RFC 3066 – Tag for Identification of Languages. This language code consists of multiple subtags separated by hyphens (‘-‘). The first subtag identifies the major language code. It must be drawn from ISO 639.2 -Codes for the representation of names of languages--Part 1: Alpha-2 Code whenever possible. If no two character code is available, it may be drawn from ISO 639.2 - Codes for the representation of names of languages--Part 2: Alpha-3 Code. There are also additional escape mechanisms that aren’t described further here.
  The second subtag is optional. If present, it must be 2-8 characters in length. If two characters in length, it should contain a country code drawn from ISO 3166-1 Two character country codes. If the subtag is 3-8 characters long, it must come from the IANA language tag registry. Additional subtags serve to further refine the language.
- **RelationshipCode** - a concept code that uniquely identifies a particular relation as it occurs in a code system. Relationship codes must be drawn from the HL7 ConceptRelationship code system whenever possible.
- **ApplicationContextCode** - a code that identifies a context or realm such as a geopolitical domain, profession or other setting.
- **DataTypeCode** - a code that identifies the data type of a RIM attribute (e.g. CD, CE, CS, BIN, ST, etc).
- **CodingStrengthCode** - a code that identifies how non-codeable elements should be treated within an HL7 attribute (CWE - coded with allowed exceptions, CNE - coded, no exceptions)
- **ValueSetNodeTypeCode** - a code that defines the type of a value set that is returned as a nested list. Types are "A" – abstract, meaning that the value set is not selectable, "S" – specializable, meaning that the values set may be selected but also has further refinements and "L" – leaf, meaning that the node represents a selectable concept code that has no further subdivisions.
- **CodeSystemTypeCode** - a code that identifies a code system as internally maintained by HL7 (I), externally maintained (E) or externally maintained but HL7 carries an internal copy (EI).
- **MatchAlgorithmCode** - a code that identifies a string matching algorithm. Match algorithm codes are used in internal search functions

The following table lists the code systems and OIDS that are used in the CTS MAPI messages.

MAPI Data Element	Code System OID	Code System Name
LanguageCode	2.16.840.1.113883.6.99	ISO 639-1 Two character Alpha Language Codes
LanguageCode	2.16.840.1.113883.6.100	ISO 639-2 Three character Alpha Language Codes
RelationshipCode	2.16.840.1.113883.5.1088	ConceptCodeRelationship
ApplicationContextCode	2.16.840.1.113883.5.147	VocabularyDomainQualifier.RealmOfUse
DataTypeCode	2.16.840.1.113883.5.1007	DataType
CodingStrengthCode	2.16.840.1.113883.5.147	VocabularyDomainQualifier
ValueSetNodeTypeCode	2.16.840.1.113883.5.24	ConceptGenerality
CodeSystemType	2.16.840.1.113883.5.1085	CodeSystemType
MatchAlgorithmCode	2.16.840.1.113883.5.1094	MatchAlgorithm

Table 11: Coded Data Element OIDS and Names

6.2.2.1 Text Match Algorithms

Several of the functions in the sections that follow allow a string of text to be passed as a search criteria. This text is accompanied by a "match algorithm code" that determines how the text will be applied. The table below lists a set of pre-determined match algorithms. If the "required" column is TRUE, all service implementations must support this algorithm in order to be considered compliant. If the "required" column if FALSE, it isn't necessary for a service to implement the algorithm, but if it does, it should use the supplied code to represent the algorithm. Note that the match algorithm list is not exhaustive. It is permissible for service implementations to extend the list below with additional, custom match algorithms as appropriate, although implementers are strongly encouraged to register the algorithm code with HL7 to enable future interoperability.

Match Algorithm Code	Description	Required
IdenticalIgnoreCase	The lower case representation of the target text must match the lower case representation matchText exacty.	TRUE
Identical	The target text must match the matchText exactly.	FALSE
StartsWithIgnoreCase	The lower case representation of target text must begin with the lower case representation of matchText.	TRUE
StartsWith	The target text must begin with the matchText.	FALSE
EndsWithIgnoreCase	The lower case representation of the target text must end with the lower case representation of matchText.	TRUE
EndsWith	The target text must end with the matchText.	FALSE
ContainsPhraseIgnoreCase	The lower case representation of the target text must contain the lower case representation of the matchText.	TRUE
ContainsPhrase	The target text must contain the matchText.	FALSE
WordsAnyOrderIgnoreCase	The target text must contain all of the words in the match text, but in any order.	FALSE
WildCardsIgnoreCase	The match text may contain zero or more 'wild cards', designated by an asterisk (*). Wild cards match 0 of more characters in the target string. The escape character is a backslash('\') meaning that the matchText "a*b*' would match any string that begins with the string "a*b".	FALSE
RegularExpression	The match text may contain regular expressions, as defined in XML Schema Part 2: Datatypes.	FALSE
NYSIIS	New York State Identification and Intelligence System phonetic encoding	FALSE

Table 12: Match Algorithm Codes

6.2.3 Service Identification Section

The message runtime and browsing API both inherit a common identification interface.

```
interface Identification {
        types::ST       getServiceName() raises (UnexpectedError);
        types::ST       getServiceVersion() raises (UnexpectedError);
        types::ST       getServiceDescription() raises (UnexpectedError);
        types::ST       getHL7ReleaseVersion() raises (UnexpectedError);
        CTSVersionId    getCTSVersion() raises (UnexpectedError);
};
```

Listing 3: Common Identification Interface

- **getServiceName** - returns the name assigned to the service by the service provider
- **getServiceVersion** - returns an implementation-specific service version identifier
- **getServiceDescription** - returns a description of the service, author, purpose, etc.
- **getHL7ReleaseVersion** - returns the latest HL7 vocabulary (not RIM) release represented by the service.
- **getCTSVersion** - returns the specific CTS version that the service implements (e.g. 1.0)

6.2.4 Exceptions

The following table contains the exceptions that can be raised by one or more of the methods described in this chapter. An exception is an abnormal condition that prevents a method invocation from completing.

The exceptions described below assume that the baseline exception includes a text field where the specific details of the exception can be spelled out. The additional attributes below provide further information beyond the basic text.

```
exception UnexpectedError {
        types::ST possible_cause;
};

exception UnknownVocabularyDomain {
        VocabularyDomainName vocabularyDomain_name;
};

exception UnknownApplicationContextCode {
        ApplicationContextCode  applicationContext_code;
};

exception UnknownValueSet {
        ValueSetDescriptor valueSet;
};

exception ValueSetNameIdMismatch {
        ValueSetId       valueSet_id;
        ValueSetName     valueSet_name;
};

exception UnknownConceptCode {
        ConceptId concept_id;
};

exception SubsumptionNotSupported {
        CodeSystemId codeSystem_id;
};

exception UnrecognizedQualifier {
        types::CR qualifier;
};

exception UnableToTranslate {
};

exception UnknownCodeSystem {
        CodeSystemId codeSystem_id;
};

exception UnknownLanguage {
        LanguageCode language_code;
};

exception InvalidExpansionContext {
};

exception QualifiersNotSupported {
};

exception NoApplicableValueSet {
        VocabularyDomainName    vocabularyDomain_name;
        ApplicationContextCode  applicationContext_code;
};

exception CodeSystemNameIdMismatch {
        CodeSystemId            codeSystem_id;
        CodeSystemName          codeSystem_name;
};

exception TimeoutError {
};

exception BadlyFormedMatchText {
        types::ST       matchText;
};

exception UnknownMatchAlgorithm {
        MatchAlgorithmCode      matchAlgorithm_code;
};

exception NoApplicableDesignationFound {
        types::CD       codeToFillIn;
        types::ST       displayLanguage_code;
};
```

Listing 4: Message API Exceptions

- **UnexpectedError** - An error that could not be anticipated by the specification has occurred. This includes things like memory faults, I/O errors, database access errors and any other sort of unexpected event that prevents successful completion of the API call. possible_cause can carry a more detailed description of what actually occurred.
- **UnknownVocabularyDomain** - The vocabularyDomain_name isn't recognized by the service
- **UnknownApplicationContextCode** - The applicationContext_code isn't recognized by the service
- **UnknownValueSet**
    1. A value set id was supplied, but it isn't recognized by the service or
    2. Only a value set name was supplied the name isn't recognized by the service
    3. Neither a name or an id was supplied
- **ValueSetNameIdMismatch** - Both valueSet_id and valueSet_name were supplied in a method call. valueSet_id identified a valid value set but it either wasn't the same set as that identified by valueSet_name. valueSet_name may identify a different value set or no valid value set at all.
- **UnknownConceptCode** - concept_id.concept_code isn't recognized as a valid concept code in the code system identified by concept_id.codeSystem_id
- **SubsumptionNotSupported** - The service doesn't support subsumption testing for the code system named by codeSystem_id.
- **UnrecognizedQualifier** - The supplied relationship qualifier isn't recognized by the service.
- **UnableToTranslate** - the service wasn't able to perform the requested translation
- **UnknownCodeSystem**
    1. A code system_id was supplied and wasn't recognized by the service
    2. A code system name was supplied, but not the id and the name wasn't recognized by the service
    3. Neither the code system name nor the code system id were supplied
- **InvalidExpansionContext** - The expansion context that was supplied by the client is invalid. This can occur because the context was somehow corrupted or because a time limit has expired and the context is no longer valid.
- **QualifiersNotSupported** - The service doesn't support subsumption testing on post-coordinated expressions
- **NoApplicableValueSet** - The service was unable to determine which value set should be used for vocabularyDomain_name and applicationContext_code.
- **CodeSystemNameIdMismatch** - codeSystem_name doesn't identify the same code system as codeSystem_id, or it doesn't name a code system at all.
- **TimeoutError** - The time limit specified for the function call has expired.
- **BadlyFormedMatchText** - The supplied match text was syntactically incorrect for the specified match algorithm.
- **UnknownMatchAlgorithm** - The supplied match algorithm isn't supported by the service.

6.3 The CTS Message Runtime API

6.3.1 Message Runtime Identification Section

The following sections describe the interface methods of the runtime portion of the message API.

```
interface Runtime : Identification {

        MatchAlgorithmCodeList  getSupportedMatchAlgorithms()
                 raises (UnexpectedError);

        VocabularyDomainNameList getSupportedVocabularyDomains(
                in types::ST            matchText,
                in MatchAlgorithmCode   matchAlgorithm_code,
                in long                 timeout,
                in long                 sizeLimit
                )
                raises (BadlyFormedMatchText,
                        UnknownMatchAlgorithm,
                        TimeoutError,
                        UnexpectedError
                );
                        ...
};
```

Listing 5: Message Runtime Identification Section

The Runtime section inherits the identification information from the Identification section.

getSupportedMatchAlgorithms - return a list of match algorithms that are implemented in the service.

getSupportedVocabularyDomains - return a list of vocabulary domains that are known to the service.

Parameters:

- **matchText** - If present and non-null, only vocabulary domains having names that match the text. If matchtext absent or null, all vocabulary domains will be returned.
- **matchAlgorithm_code** - If the matchText is present and non-null, the matchAlgorithm_code determines how the match text is applied. See Text Match Algorithms (§ 6.2.2.1) for details.
- **timeout** - The time, in milliseconds, that the client is willing to wait for this operation to be completed. A timeout value of 0 means that there is no time limit on the operation.
- **sizeLimit** - The maximum number of elements that the service may return. If exactly sizeLimit items are returned, the client assume that there were additional itemes that weren't returned. A sizeLimit of 0 indicates that the number of items that may be returned is unlimited.

Exceptions:

- **BadlyFormedMatchText**
- **UnknownMatchAlgorithm**
- **TimeoutError**
- **UnexpectedError**

6.3.2 Validate a Coded Attribute

```
struct ValidationDetail {
        types::CD codeInError;
        types::BL isError;
        types::ST error_id;
        types::ST errorText;
};
typedef sequence<ValidationDetail> ValidationDetailList;

struct ValidateCodeReturn {
        types::INT nErrors;
        types::INT nWarnings;
        ValidationDetailList detail;
};
```

Listing 6: ValidateCodeReturn structures

A ValidationDetail entry contains a detailed description of an error or warning.

- **codeInError** - the concept code that the actual error or warning occurred on. If the error was detected in a concept qualifier or a translation, this value contains the qualifier name, qualifier value or translation value that was actually in error.
- **isError** - TRUE means that the detail entry describes an error. FALSE means that it is a warning.
- **error_id** - the error identifier (see: Table 13 - ValidateCode and ValidateTranslation return codes)
- **errorText** - a textual explanation of the error or warning

ValidateCodeReturn returns the summary of validateCode call along with the details

- **nErrors** - the number of errors encountered. If greater than zero, the code or translation is not valid.
- **nWarnings** - the number of warnings returned. If a code or translation only has warnings, it can still be transmitted and processed even though some of the fields may not be correct.
- **detail** - a list of the individual errors and/or warnings

```
ValidateCodeReturn validateCode(
        in VocabularyDomainName         vocabularyDomain_name,
        in types::CD                    codeToValidate,
        in ApplicationContextCode       applicationContext_code,
        in types::BL                    activeConceptsOnly,
        in types::BL                    errorCheckOnly
        )
        raises (UnknownVocabularyDomain,
                UnexpectedError,
                UnknownApplicationContextCode,
                NoApplicableValueSet);
```

Listing 7: validateCode

validateCode determines whether the coded attribute value contains a valid concept representation for the supplied vocabulary domain and application context. If errorCheckOnly is false, it also validates the code system name and displayName attributes if they are present. It then recursively validates the qualifiers using the same criteria. Concept codes that have originalText and no code are always considered invalid. validateCode doesn’t validate translations.

Parameters:

- **vocabularyDomain_name** - vocabulary domain for the supplied coded attribute
- **codeToValidate** - the HL7 concept code to be validated. At bare minimum code must contain the code and code system attributes.
- **applicationContext_code** - the context or "realm" in which the code is to be used
- **activeConceptsOnly** - if true (default), only concept codes that are currently active are considered valid. If false, the code will be considered valid as long as it is present in the code system.
- **errorCheckOnly** - if true, the concept code will only be checked for errors. No warnings will be returned

Exceptions:

- **UnknownVocabularyDomain**
- **UnknownApplicationContextCode**
- **UnexpectedError**

6.3.3 Validate the Translation(s) of a Coded Attribute

```
ValidateCodeReturn validateTranslation(
       in VocabularyDomainName         vocabularyDomain_name,
       in types::CD                    codeToValidate,
       in ApplicationContextCode       applicationContext_code,
       in types::BL                    activeConceptsOnly,
       in types::BL                    errorCheckOnly
       )
       raises (UnknownVocabularyDomain,
               UnknownApplicationContextCode,
               UnexpectedError);
```

Listing 8: ValidateTranslation

validateTranslation validates the primary code and any translations of an HL7 coded attribute. Any errors or warnings are returned in the ValidateCodeReturn structure.

Note: the details of what constitutes a valid translation are not covered in this document. It is presumed that an outside entity has defined the rules of translation and that this function provides a standardized way to access those rules. Refer to Code Mapping Model (§ 9) for the underlying translation model.

- **vocabularyDomain_name** - vocabulary domain for the supplied coded attribute
- **codeToValidate** - the HL7 concept code to be validated. At bare minimum code must contain the code and code system attributes.
- **applicationContext_code** - the context or "realm" in which the code is to be used
- **activeConceptsOnly** - if true (default), only concept codes that are currently active are considered valid. If false, the code will be considered valid as long as it is present in the code system.
- **errorCheckOnly** - if true, the concept code will only be checked for errors. No warnings will be returned

Exceptions:

- **UnknownVocabularyDomain**
- **UnknownApplicationContextCode**
- **UnexpectedError**

6.3.3.1 ValidateCode and ValidateTranslation Return codes

The following table lists the validation error identifiers and associated text that can be returned from ValidateCode and/or ValidateTranslation.

Table 13: ValidateCode and ValidateTranslation return codes

ID	Type	Text	Description
E001	E	Unknown code system	The code system isn’t recognized by the service.
E002	E	Invalid concept code for code system	The concept code isn’t valid for the supplied (or implied in the case of CS data types) code system.
E003	E	Code system not valid for vocabulary domain	The code system is recognized by the service, but it isn’t valid for the vocabulary domain and application context.
E004	E	Concept code is not active	The concept code is valid for the domain, but it is no longer active. This error occurs only when activeConceptsOnly is true.
E005	E	Concept code is not valid for vocabulary domain	The concept code is valid for the code system, but is not allowed in the vocabulary domain and application context.
E008	E	Invalid role name for vocabulary domain	The code system and concept code of the qualifier role are legal, but are not valid for the vocabulary domain.
E009	E	Role name must be supplied for vocabulary domain	A qualifier was present that didn’t have a name component and it is required for this vocabulary domain.
E010	E	Invalid value for qualifier	The qualifier value is a valid concept code, but is not valid in the context of the qualifier.
E011	E	Invalid translation	The translation code system and concept code are valid, but it is not a valid translation of the containing concept code.
E013	E	Missing concept code	The concept code field is empty.
E014	E	Unknown coding rationale	The coding rationale code is not recognized.
		Code system name doesn’t	

W002	W	match code system	The code system name does not match the code system id.
W003	W	Unknown code system version	The version is not recognized for the supplied code system.
W004	W	Display name incorrect for concept code	The display name is not correct for the supplied concept code.
W005	W	No HL7 translation present	None of the translations have a SH (both Source and HL7) or HL7 coding rationale.
W006	W	Concept code is not active	The supplied concept code is not active and activeConceptsOnly is TRUE.

6.3.4 Translate a Coded Attribute

```
types::CD translateCode(
       in VocabularyDomainName         vocabularyDomain_name,
       in types::CD                    fromCode,
       in CodeSystemId                 toCodeSystemId,
       in ApplicationContextCode       toApplicationContext_code
       )
       raises (UnknownVocabularyDomain,
               UnknownCodeSystem,
               UnknownApplicationContextCode,
               UnableToTranslate,
               UnexpectedError);
```

Listing 9: Translate Code

translateCode translates fromCode into the concept code, if any, that is valid for the vocabulary domain in the target code system or application context. It returns a complete copy of fromCode with the new translation (if any) appended to the end of the CD.translation sequence. If fromCode already contains a valid translation for the target code system or application context, the return copy of fromCode will match the original.

- **vocabularyDomain_name** - the vocabulary domain of the coded attribute to be translated
- **fromCode**– the code to be translated
- **toCodeSystem_id** - (optional) If supplied, translate the code into a concept code (or codes) drawn from this code system rather than determining the code system from the target context.
- **toApplicationContext_code** - (optional) The application context of the translated code. Used to determine the target value set which, in turn, determines the target code system. Only one of toCodeSystem_id or targetContext may be specified. If neither or both are specified an UnableToTranslate exception will be thrown.

translateCode returns an HL7 coded attribute that has been translated into the terms of the target coding system. The target code system can either be supplied in the call or can be determined from the targetContext.

Exceptions:

- **UnknownVocabularyDomain**
- **UnknownCodeSystem**
- **UnknownApplicationContextCode**
- **UnableToTranslate**
- **UnexpectedError**

6.3.5 Fill in the Details of a Coded Attribute

```
 types::CD fillInDetails(
       in types::CD                    codeToFillIn,
       in LanguageCode                 displayLanguage_code
       )
       raises (UnknownCodeSystem,
               UnknownConceptCode,
               UnknownLanguage,
               UnexpectedError,
               NoApplicableDesignationFound);
```

Listing 10: fillInDetails

fillInDetails returns a complete copy of codeToFillIn with codeSystemName, codeSystemVersion and displayName filled out in the base code and its qualifiers, if any. Qualifiers are filled out recursively - if qualifiers are nested or have other qualifiers, the details are filled in here as well. fillInDetails does not change the code translations.

Parameters:

- **codeToFillIn** - coded attribute to be completed
- **displayLanguage_code**- language to use for the display name(s). If blank or null, the default language code for the code system is used.

Exceptions:

- **UnknownCodeSystem**
- **UnknownConceptCode**
- **UnknownLanguage**
- **UnexpectedError**

6.3.6 Determine Whether One Coded Attribute Subsumes a Second

```
types::BL subsumes(
       in types::CD  parentCode,
       in types::CD  childCode
       )
       raises (UnknownCodeSystem,
               UnknownConceptCode,
               SubsumptionNotSupported,
               UnrecognizedQualifier,
               QualifiersNotSupported,
                UnexpectedError);
```

Listing 11: subsumes

subsumes tests whether the parent coded attribute subsumes (is implied by) the child. If neither parentCode nor childCode have any qualifiers and both are drawn from the same code system, subsumes returns true if and only if childCode can be determined to belong to the transitive closure of the *hasSubtype* relationship graph headed by parentCode. This document makes no further assertions of the semantics of subsumption beyond this one case.

If the service supports subsumption involving qualifiers and/or subsumption tests across multiple code systems, it must define the appropriate translation semantics. If the service doesn't support qualifier subsumption, it should raise the QualifiersNotSupported exception if presented with a parentCode or childCode containing qualifiers. Similarly, if the service doesn't support cross-code system subsumption testing, it should raise SubsumptionNotSupported when supplied with codes with different code systems.

Parameters:

- **parentCode** - the parent coded attribute to test
- **childCode**- the child coded attribute to test

subsumes will return 'true' if the child code can be determined to be a subtype of the parent. Subsumes will also return 'true' if the child and parent codes are equivalent. Translations are ignored in this method.

Exceptions:

- **UnknownCodeSystem**
- **UnknownCode**
- **SubsumptionNotSupported**
- **UnrecognizedQualifier**
- **QualifiersNotSupported**
- **UnexpectedError**

6.3.7 Test Whether Two Coded Attributes are Equivalent

```
types::BL areEquivalent (
        in types::CD    code1,
        in types::CD    code2
        )
        raises (UnknownCodeSystem,
                UnknownConceptCode,
                SubsumptionNotSupported,
                UnrecognizedQualifier,
                QualifiersNotSupported,
                UnexpectedError);
```

Listing 12: areEquivalent

areEquivalent determines whether the two supplied codes represent an equivalent concept from the perspective of the service. It is possible for one or both of the supplied codes to include qualifiers or be drawn from different code systems. code1 and code2 are considered equivalent if and only if code1 subsumes code2 and code2 subsumes code1. The semantics of subsumption are defined in the previous section.

Parameters:

- **code1** - first code to test for equivalence
- **code2** - second code to test

Exceptions:

- **UnknownCodeSystem**
- **UnknownCode**
- **SubsumptionNotSupported**
- **UnrecognizedQualifier**
- **QualifiersNotSupported**
- **UnexpectedError**

6.3.8 Expand a Vocabulary Domain

```
struct ValueSetDescriptor {
        ValueSetId      valueSet_id;
        ValueSetName    valueSet_name;
};
typedef sequence<ValueSetDescriptor> ValueSetDescriptorList;
```

Listing 13: ValueSetDescriptor structure

Every value set has a unique identifier. In addition, some value sets may also have an optional mnemonic or name, which, if present, must also be unique.

- **ValueSet_id** - the unique value set identifier.
- **ValueSet_name** - the unique value set name (optional)

```
struct ValueSetExpansion {
        types::INT              pathLength;
        ValueSetNodeTypeCode    nodeType_code;
        ValueSetDescriptor      valueSet;
        ConceptId               concept_id;
        types::ST               displayName;
        types::BL               isExpandable;
        ExpansionContext        expansionContext;
};
typedef sequence<ValueSetExpansion> ValueSetExpansionList;
```

Listing 14: ValueSetExpansion node

- **pathLength** - an integer that defines the distance from the root value set. The root value set will always have a path length of 0
- **nodeType_code** - a concept code drawn from the HL7 ConceptGenerality (2.16.840.1.113883.5.24) code system. One of:

    - "S" - a specializable node. The concept_id in this node can be selected, but the node can also be further expanded.
    - "A" - an abstract node. The concept_id (if any) in this node cannot be selected. The node must be further expanded. To make a selection
    - "L" - a leaf node. The concept_id in this node may be selected, but no further expansion is possible.
- **valueSet** - id and name (if any) of the value set associated with this node
- **concept_id** - the code system and concept code associated with this node, if any
- **displayName** - the display name used to represent this node.
- **isExpandable** - TRUE means that this node can be further expanded using the expandValueSetExpansionContext function. canExpand will only be TRUE for specializable and abstract nodes, and then only when the original lookupValueSetExpansion call was made with expandAll set to FALSE.
- **expansionContext** - if isExpandable is TRUE, expansionContext contains whatever is required by the specific service implementation to further expand the node in a subsequent call. It is opaque to the client software.

The various cases below describe how different sorts of value set definitions would be expanded.

Case 1: Value set A references all of the codes in a non-hierarchical code system, which contains concept codes 1,2, ..N

pathLength	nodeType_code	Concept_id	displayName
0	A (abstract)	-	(A).valueSet_name
1	L (leaf)	1	(1).preferredName
1	L (leaf)	2	(2).preferredName
1	L (leaf)	...	...
1	L (leaf)	N	(N).preferredName

Table 14: ValueSetExpansion Case 1

Case 2: Value set B references all of the codes in a hierarchical code system. The concept code hierarchy is reflected in concept id (1 is root, 1.1 is first child of root, 1.2 second child, 1.2.1 first child of first child, etc.)

pathLength	nodeType_code	Concept_id	displayName
0	A (abstract)	-	(B).valueSet_name
1	S (specializable)	1	(1).preferredName
2	S (specializable)	1.1	(1.1).preferredName
...	S (specializable)	...	...
n	L (leaf)	1.1.1..n	(1.1.1.n).preferredName
1	S (specializable)	2	(2).preferredName
2	S (specializable)	2.1	(2.1) preferredName
...	S (specializable)	...	...
M	L (leaf)	2.1..m	(2.1..m).preferredName

Table 15: ValueSetExpansion Case 2

Case 3: Value set C specifically references concept codes 1,2,3 in a code system. No references include a relationship code and there is no head code.

pathLength	nodeType_code	Concept_id	displayName
0	A (abstract)	-	(C).valueSetName
1	L (leaf)	1	(1).preferredName
1	L (leaf)	2	(2).preferredName
1	L (leaf)	3	(3).preferredName

Table 16: ValueSetExpansion Case 3

Case 4: Value set D references concept codes 1,2,3,4 in a code system. No references include a relationship code, but concept code 4 is defined as the head code.

pathLength	nodeType_code	Concept_id	displayName
0	A (abstract)	4	(4).preferredName
1	L (leaf)	1	(1).preferredName
1	L (leaf)	2	(2).preferredName
1	L (leaf)	3	(3).preferredName

Table 17: ValueSetExpansion Case 4

Case 5: Value set E references value set D above, with includeHeadCode set to TRUE

pathLength	nodeType_code	Concept_id	displayName
0	A (abstract)	-	(E).valueSet_name
1	S (specializable)	4	(4).preferredName
2	L (leaf)	1	(1).preferredName
2	L (leaf)	2	(2).preferredName

2	L (leaf)	3	(3).preferredName

Table 18: ValueSetExpansion Case 5

Case 6: Value set F references value set D above, with includeHeadCode set to FALSE

pathLength	nodeType_code	Concept_id	displayName
0	A (abstract)	-	(F).valueSet_name
1	A (abstract)	4	(4).preferredName
2	L (leaf)	1	(1).preferredName
2	L (leaf)	2	(2).preferredName
2	L (leaf)	3	(3).preferredName

Table 19: ValueSetExpansion Case 6

Case 7: Value set G references concept codes 1.1, 2.1 and 3.1 in a hierarchical code system. All three references use the hasSubtype relationship code. The 1.1 reference includes the referenced code. The 2.1 reference excludes the referenced code. The 3.1 reference has leafOnly set to true. G has no head code.

pathLength	nodeType_code	Concept_id	displayName
0	A (abstract)	-	(G).valueSet_name
1	S (specializable)	1.1	(1.1).preferredName
2	S (specializable)	1.1.1	(1.1.1).preferredName
…	S (specializable)	1.1….	…
n	L (leaf)	1.1…..n	(1.1….n).preferredName
1	A (abstract)	1.2	(1.2) preferredName
2	S (specializable)	1.2.1	(1.2.1)preferredName
…	S (specializable)	1.2.1…	…
m	L (leaf)	1.2.1…m	(1.2…m)preferredName
1	A (abstract)	1.3	(1.3) preferredName
2	A (abstract)	1.3.1	(1.3.1) preferredName
…	A (abstract)	1.3.1…	…
k	L(leaf)	1.3.1…k	(1.3.1..k)preferredName

Table 20: ValueSetExpansion Case 7

```
ValueSetExpansionList lookupValueSetExpansion(
        in VocabularyDomainName         vocabularyDomain_name,
        in ApplicationContextCode       applicationContext_code,
        in LanguageCode                 language_code,
        in types::BL                    expandAll,
        in long                         timeout,
        in long                         sizeLimit
        )
        raises (UnknownVocabularyDomain,
                UnknownApplicationContextCode,
                UnknownLanguage,
                NoApplicableValueSet,
                TimeoutError,
                UnexpectedError);
```

Listing 15: lookupValueSetExpansion

lookupValueSetExpansion returns an expansion of the value set associated with the supplied vocabulary domain and optional context. The entire vocabulary domain can be expanded all at once (expandAll = TRUE) or can be expanded one level at a time (expandAll = FALSE). If expanding one level at a time, every node that can be further expanded will have isExpandable set to TRUE, meaning that the expansionContext can be passed to expandValueSetExpansionContext for further expansion.

Parameters:

- **vocabularyDomain_name** - vocabulary domain to expand
- **applicationContext_code**– context in which the vocabulary domain is to be expanded (optional)
- **language_code** - language to be used for the returned display names
- **expandAll** - TRUE means expand all nodes to their entire depth, FALSE means expand one level (default: TRUE)
- **timeout** - The time, in milliseconds, that the client is willing to wait for this operation to be completed. A timeout value of 0 means that there is no time limit on the operation.
- **sizeLimit** - The maximum number of elements that the service may return. If exactly sizeLimit items are returned, the client assume that there were additional itemes that weren't returned. A sizeLimit of 0 indicates that the number of items that may be returned is unlimited.

Exceptions:

- **UnknownVocabularyDomain**
- **UnknownApplicationContextCode**
- **UnknownLanguage**
- **TimeoutError**
- **UnexpectedError**

```
ValueSetExpansionList expandValueSetExpansionContext(
        in ExpansionContext     expansionContext
        )
        raises (InvalidExpansionContext,
                TimeoutError,
                UnexpectedError);
```

Listing 16: expandValueSetExpansionContext

expandValueSetExpansionContext takes an opaque expansionContext that was previously returned in a ValueSetExpansion entry and further expands the corresponding node. An additional level. The return is identical to that of lookupValueSetExpansion, and all of the initial constraints, including the timeout value still apply.

Parameters:

- **expansionContext**- context returned by a previous lookupValueSetExpansion or expandValueSetExpansionContext call

Exceptions:

- **InvalidExpansionContext**
- **TimeoutError**
- **UnexpectedError**

6.4 CTS Message Browsing API

The second part of the CTS Message API contains a set of functions that can be used to examine and browse attributes, vocabulary domains and value sets.

6.4.1 Message Browser Identification Information

```
interface Browser : Identification {

                        ...
};
```

Listing 17: Message Browser Identification

The message browser inherits the identification information from the interface described in Service Identification Section (§ 6.2.3).

6.4.2 Message Browser Description Section

This section describes several "building block" structures that are used by the service description and then describes how the information is accessed via the service.

6.4.2.1 Browser Description Building Blocks

```
struct RIMAttributeId {
        types::ST   model_id;
        types::ST   class_name;
        types::ST   attribute_name;
};

struct RIMCodedAttribute {
        RIMAttributeId          RIMAttribute_id;
        DataTypeCode            dataType_code;
        CodingStrengthCode      codingStrength_code;
        VocabularyDomainName    vocabularyDomain_name;
};
typedef sequence<RIMCodedAttribute> RIMCodedAttributeList;
```

Listing 18: RIMCodedAttribute structure

RIMAttributeId uniquely identifies a coded attribute in the RIM.

- **model_id** - the RIM model identifier
- **class_name** - the name of the RIM Class
- **attribute_name** - the name of the attribute within the class

The RIMCodedAttribute describes a RIM attribute.

- **RIMAttribute_id** - the unique attribute identifier
- **dataType_code** - a code that identifies the attribute data type, drawn from the HL7 DataType code system
- **codingStrength_code**- the attribute strength. Drawn from the HL7 VocabularyDomainQualifier code system
- **vocabularyDomain_name**- the name of the vocabulary domain associated with the attribute.

```
struct CodeSystemDescriptor {
        CodeSystemId            codeSystem_id;
        CodeSystemName          codeSystem_name;
        types::ST               copyright;
        ReleaseVersionIdList    availableReleases;
};
typedef sequence<CodeSystemDescriptor> CodeSystemDescriptorList;
```

Listing 19: CodeSystemDescriptor structure

A CodeSystemDescriptor consists of a code sytem identifier, its name, any copyright information and a list of available versions.

- **codeSystem_id** - the ISO Object Identifier (OID) of the code system
- **codeSystem_name** - the name of the code system. Unique within the HL7 Version 3 context.
- **copyright** - copyright notice for the code system, if any. If present, the copyright should be displayed whenever code system is accessed or used.
- **availableReleases**- the releases of the code system that are currently supported by the service. In the current version of this specification, at most one release may be queried for any code system. Future versions will allow more than one however.

6.4.2.2 Browser Description

```
MatchAlgorithmCodeList  getSupportedMatchAlgorithms() raises (UnexpectedError);
```

```
RIMCodedAttributeList   getSupportedAttributes(
        in types::ST            matchText,
        in MatchAlgorithmCode   matchAlgorithm_code,
        in long                 timeout,
        in long                 sizeLimit
        )
        raises (BadlyFormedMatchText,
                UnknownMatchAlgorithm,
                TimeoutError,
                UnexpectedError);

VocabularyDomainNameList getSupportedVocabularyDomains(
        in types::ST            matchText,
        in MatchAlgorithmCode   matchAlgorithm_code,
        in long                 timeout,
        in long                 sizeLimit
        )
        raises (BadlyFormedMatchText,
                UnknownMatchAlgorithm,
                TimeoutError,
                UnexpectedError);

ValueSetDescriptorList  getSupportedValueSets(
        in types::ST            matchText,
        in MatchAlgorithmCode   matchAlgorithm_code,
        in long                 timeout,
        in long                 sizeLimit
        )
        raises (BadlyFormedMatchText,
                UnknownMatchAlgorithm,
                TimeoutError,
                UnexpectedError);

CodeSystemDescriptorList getSupportedCodeSystems(
        in types::ST            matchText,
        in MatchAlgorithmCode   matchAlgorithm_code,
        in long                 timeout,
        in long                 sizeLimit
        )
        raises (BadlyFormedMatchText,
                UnknownMatchAlgorithm,
                TimeoutError,
                UnexpectedError);
```

Listing 20: Message Browser Description

The message browser description lists all of the various entities that are supported by the particular service.

- **getSupportedAttributes** - return a list of the RIM attributes that are known to the service and match the supplied criteria.

  Parameters:

  - **matchText** - If present and non-null, only RIM attributes having names that match the text will be returned. If matchtext absent or null, all RIM attributes will be returned.
  - **matchAlgorithm_code** - If the matchText is present and non-null, the matchAlgorithm_code determines how the match text is applied. See Text Match Algorithms (§ 6.2.2.1) for details.
  - **timeout** - The time, in milliseconds, that the client is willing to wait for this operation to be completed. A timeout value of 0 means that there is no time limit on the operation.
  - **sizeLimit** - The maximum number of elements that the service may return. If exactly sizeLimit items are returned, the client assume that there were additional itemes that weren't returned. A sizeLimit of 0 indicates that the number of items that may be returned is unlimited.

  Exceptions:

  - **BadlyFormedMatchText**
  - **UnknownMatchAlgorithm**
  - **TimeoutError**
  - **UnexpectedError**

- **getSupportedVocabularyDomains** - return a list of the vocabulary domains that are known to the service and match the supplied criteria.

  Parameters:

  - **matchText** - If present and non-null, only vocabulary domains having names that match the text will be returned. If matchtext absent or null, all vocabulary domains will be returned.
  - **matchAlgorithm_code** - If the matchText is present and non-null, the matchAlgorithm_code determines how the match text is applied. See Text Match Algorithms (§ 6.2.2.1) for details.
  - **timeout** - The time, in milliseconds, that the client is willing to wait for this operation to be completed. A timeout value of 0 means that there is no time limit on the operation.
  - **sizeLimit** - The maximum number of elements that the service may return. If exactly sizeLimit items are returned, the client assume that there were additional itemes that weren't returned. A sizeLimit of 0 indicates that the number of items that may be returned is unlimited.

  Exceptions:

  - **BadlyFormedMatchText**
  - **UnknownMatchAlgorithm**
  - **TimeoutError**
  - **UnexpectedError**

- **getSupportedValueSets** - return a list of the value sets that are known to the service and match the supplied criteria.

  Parameters:

  - **matchText** - If present and non-null, only value sets having names that match the text will be returned. If matchtext absent or null, all value sets will be returned.

- **matchAlgorithm_code** - If the matchText is present and non-null, the matchAlgorithm_code determines how the match text is applied. See Text Match Algorithms (§ 6.2.2.1) for details.
- **timeout** - The time, in milliseconds, that the client is willing to wait for this operation to be completed. A timeout value of 0 means that there is no time limit on the operation.
- **sizeLimit** - The maximum number of elements that the service may return. If exactly sizeLimit items are returned, the client assume that there were additional itemes that weren't returned. A sizeLimit of 0 indicates that the number of items that may be returned is unlimited.

Exceptions:

- **BadlyFormedMatchText**
- **UnknownMatchAlgorithm**
- **TimeoutError**
- **UnexpectedError**

- **getSupportedCodeSystems** - return a list of the code systems that are known to the service and match the supplied criteria.

  Parameters:

  - **matchText** - If present and non-null, only code system names names that match the text will be returned. If matchtext absent or null, all code systems will be returned.
  - **matchAlgorithm_code** - If the matchText is present and non-null, the matchAlgorithm_code determines how the match text is applied. See Text Match Algorithms (§ 6.2.2.1) for details.
  - **timeout** - The time, in milliseconds, that the client is willing to wait for this operation to be completed. A timeout value of 0 means that there is no time limit on the operation.
  - **sizeLimit** - The maximum number of elements that the service may return. If exactly sizeLimit items are returned, the client assume that there were additional itemes that weren't returned. A sizeLimit of 0 indicates that the number of items that may be returned is unlimited.

  Exceptions:

  - **BadlyFormedMatchText**
  - **UnknownMatchAlgorithm**
  - **TimeoutError**
  - **UnexpectedError**

6.4.3 Look Up a Vocabulary Domain

```
struct VocabularyDomainValueSet {
        ValueSetDescriptor              definedByValueSet;
        ApplicationContextCode          applicationContext_code;
};
typedef sequence<VocabularyDomainValueSet> VocabularyDomainValueSetList;

struct VocabularyDomainDescription {
        VocabularyDomainName            vocabularyDomain_name;
        types::ST                       description;
        VocabularyDomainName            restrictsDomain_name;
        sequence<VocabularyDomainName>  basisOfDomains;
        sequence<RIMCodedAttribute>     constrainsAttributes;
        VocabularyDomainValueSetList    representedByValueSets;
};
```

Listing 21: VocabularyDomainDescription structure

The VocabularyDomainValueSet structure contains the description of a value set (see: Expand a Vocabulary Domain (§ 6.3.8) for details) and the context code, if any, in which the value set applies

- **definedByValueSet** - the value set that defines a vocabulary domain in the specific application context
- **applicationContext_code** - the context in which the value set applies

VocabularyDomainDescription is the structure returned by the lookupVocabularyDomain function.

- **vocabularyDomain_name** - the unique name of the vocabulary domain
- **description** - a description of the domain
- **restrictsDomain_name** - the vocabulary domain that this domain restricts, if any
- **basisOfDomains** - a list of the domains that are restrictions of this domain, if any
- **constrainsAttributes** - a list of the RIM attributes that use this vocabulary domain, if any
- **representedByValueSets** - a list of the value sets and context that can represent this vocabulary domain, if any

```
VocabularyDomainDescription lookupVocabularyDomain(
        in VocabularyDomainName vocabularyDomain_name
        )
        raises (UnknownVocabularyDomain,
                UnexpectedError);
```

Listing 22: lookupVocabularyDomain

lookupVocabularyDomain returns detailed information about a vocabulary domain.

- **vocabularyDomain_name** - the name of the vocabulary domain to look up

Exceptions:

- **UnknownVocabularyDomain**
- **UnexpectedError**

6.4.4 Look Up a Value Set

```
struct ValueSetConstructor {
        ValueSetDescriptor      includedValueSet;
        types::BL               includeHeadCode;
};
```

```
typedef sequence<ValueSetConstructor> ValueSetConstructorList;

struct ValueSetCodeReference {
        ConceptCode             referenced_code;
        RelationshipCode        relationship_code;
        types::BL               includeReferencedCode;
        types::BL               leafOnly;

};
typedef sequence<ValueSetCodeReference> ValueSetCodeReferenceList;

struct ValueSetDescription {
        ValueSetDescriptor      idAndName;
        types::ST               description;
        types::ST               definingExpression;
        CodeSystemDescriptor    basedOnCodeSystem;
        types::BL               allCodes;
        ConceptCode             head_code;
};

struct FullValueSetDescription {
        ValueSetDescription             description;
        ValueSetConstructorList         constructedUsingValueSets;
        ValueSetDescriptorList          usedToDefine;
        ValueSetCodeReferenceList       referencesCodes;
};
```

Listing 23: FullValueSetDescriptionstructure

FullValueSetDescription includes a complete description of the value set including the list of included concept codes and other value sets.

ValueSetConstructor

- **includedValueSet** - the identifier and name, if any of a value set that is considered to be part of the containing set
- **includeHeadCode** - TRUE means that the head code, if any, of the included value set is considered to be part of the including set. FALSE means that it isn't.

ValueSetCodeReference

- **referenced_code** - a concept code that is referenced by the value set
- **relationship_code**- A relationship code. If present, all of the descendants of the referenced code are consider part of the value set as well, subject to the leafOnly constraint below. If the relationship is transitive, all descendants are included in the value set. If the relationship is not transitive, only the direct targets of the referenced code are considered.
- **includeReferencedCode**- TRUE means that the referenced code itself is part of the value set. FALSE means that only the children or descendants are included. includeReferencedCode will always be TRUE when a relationship code is not supplied.
- **leafOnly** - TRUE means only include the leaf nodes of the relationship. FALSE means include all descendant nodes except, perhaps, the referenced code itself. leafOnly must always be FALSE if no relationship_code is present.

ValueSetDescription

- **idAndName-**the value set identifier and name, if any
- **description** - a textual description of the value set and its use
- **definingExpression**- an expression that defines the set contents (if any)
- **basedOnCodeSystem** - the code system id, name and version used to construct the value set
- **allCodes** - TRUE means that all of the codes in the code system are included in the value set. If TRUE, the value set mustn't reference any additional code
- **head_code**- the concept code that represents the entire value set (if any)

FullValueSetDescription

- **description** - The id, name, description, etc. of the value set
- **constructedUsingValueSets**- a list of additional value sets and head code inclusions, if any, that are used to construct this set
- **usedToDefine** - a list of any other value sets that have been constructed using this set
- **referencesCodes** - a list of concept codes that this value set explicitly references. This will be empty if allCodes is true and will not include any codes that are implicitly included from other value sets and/or concept code relationships.

```
FullValueSetDescription lookupValueSet(
        in ValueSetId           valueSet_id,
        in ValueSetName         valueSet_name
        )
raises (UnknownValueSet,
        ValueSetNameIdMismatch,
        UnexpectedError);
```

Listing 24: lookupValueSet

lookupValueSet retrieves a complete description of a value set given either a value set identifier or a value set name.

Parameters:

- **valueSet_id** - the identifier of the value set to look up
- **valueSet_name** - the name of the value set to look up

Either the value set id, the value set name or both may be supplied. If both are supplied, the name must correctly match the id or an error will be raised.

Exceptions:

- **UnknownValueSet**
- **ValueSetNameIdMismatch**
- **UnexpectedError**

6.4.5 Look up Detailed Information About a Code System

```
struct CodeSystemRegistration {
        types::ST               sponsor;
        types::ST               publisher;
        types::ST               versionReportingMethod;
        types::ST                licensingInformation;
        types::BL                inUMLS;
        types::ST                systemSpecificLocatorInfo;
        CodeSystemTypeCode       codeSystemType_code;
};

struct CodeSystemInfo {
        CodeSystemDescriptor     description;
        CodeSystemRegistration   registrationInfo;
};
```

Listing 25: lookupCodeSystem return structure

CodeSystemRegistration:

- **sponsor** - the HL7 member or organization that sponsers the registration
- **publisher** - the name of the official publisher of the code system
- **versionReportingMethod** - how new versions are created and distributed
- **licensingInformation** - licensing requirements for the code system
- **inUMLS** - TRUE means that the code system is contained in the Unified Medical Language System (UMLS)
- **systemSpecificLocatorInfo**- the meaning of this field depends upon the specific publisher. It serves to further identify the code system information and is intended to contain things like the HL7 Version 2 table name, etc.
- **codeSystemType_code** - A concept code that specifies whether the code system is internally created and maintained by HL7 (I), externally created and maintained (E) or externally created but HL7 maintains an internal image for convenience (EI)

Note: The registration elements above are intended to be descriptive only. While many of the fields (sponsor, publisher, versionReportingMethod, etc.) could have additional structure, it wasn't considered necessary within the scope of the present document

CodeSystemInfo

- **description** - basic description of the code system - see: Table CTSCodeSystemDescriptor structure
- **registrationInfo** - registration information for the code system (if any)

```
CodeSystemInfo lookupCodeSystem (
        in CodeSystemId          codeSystem_id,
        in CodeSystemName        codeSystem_name
        )
        raises (UnknownCodeSystem,
                CodeSystemNameIdMismatch,
                UnexpectedError);
```

Listing 26: lookupCodeSystem

lookupCodeSystem returns detailed information for a code system given either a code system identifier (OID) or name.

- **codeSystem_Id** - the unique code system identifier, which is usually the ISO OID
- **codeSystem_name** - the name of the code system

Either the code system id, the code system name or both may be supplied. If both the id and name are supplied, the name must match the id or an error will be raised.

Exceptions:

- **UnknownCodeSystem**
- **CodeSystemNameIdMismatch**
- **UnexpectedError**

6.4.6 Look up the Value Set for a Vocabulary Domain and Context

```
 ValueSetDescriptor lookupValueSetForDomain(
        in VocabularyDomainName         vocabularyDomain_name,
        in ApplicationContextCode       applicationContext_code
        )
        raises (UnknownVocabularyDomain,
                UnknownApplicationContextCode,
                NoApplicableValueSet,
                UnexpectedError);
```

Listing 27: lookupValueSetForDomain

lookupValueSetForDomain returns the descriptor (id and name if any) of the value set that would be used for the supplied domain in the application context (if any).

Parameters:

- **vocabularyDomain_name** - name of the vocabulary domain to look up
- **applicationContext_code** - application context (optional)

Exceptions:

- **UnknownVocabularyDomain**
- **UnknownApplicationContextCode**
- **NoApplicableValueSet**
- **UnexpectedError**

6.4.7 Determine Whether a Concept Code is in a Value Set

```
 types::BL isCodeInValueSet(
        in ValueSetId            valueSet_id,
        in ValueSetName          valueSet_name,
        in types::BL             includeHeadCode,
        in ConceptId             codeToValidate
        )
        raises (UnknownValueSet,
                ValueSetNameIdMismatch,
                UnknownConceptCode,
                UnknownCodeSystem,
                UnexpectedError);
```

Listing 28: isCodeInValueSet

isCodeInValueSet returns true if the supplied concept identifier is included in and can be selected from the value set, false otherwise.

Parameters:

- **valueSet_id** - the identifier of the value set to look up
- **valueSet_name** - the name of the value set to look up
- **includeHeadCode** - TRUE means that the head code (if there is any) is considered to be part of the set. FALSE means that the head code, if any, is excluded.
- **codeToValidate** - the code system id and concept code to test

Either the value set id, the value set name or both may be supplied. If both are supplied, the name must correctly match the id or an error will be raised.

Exceptions:

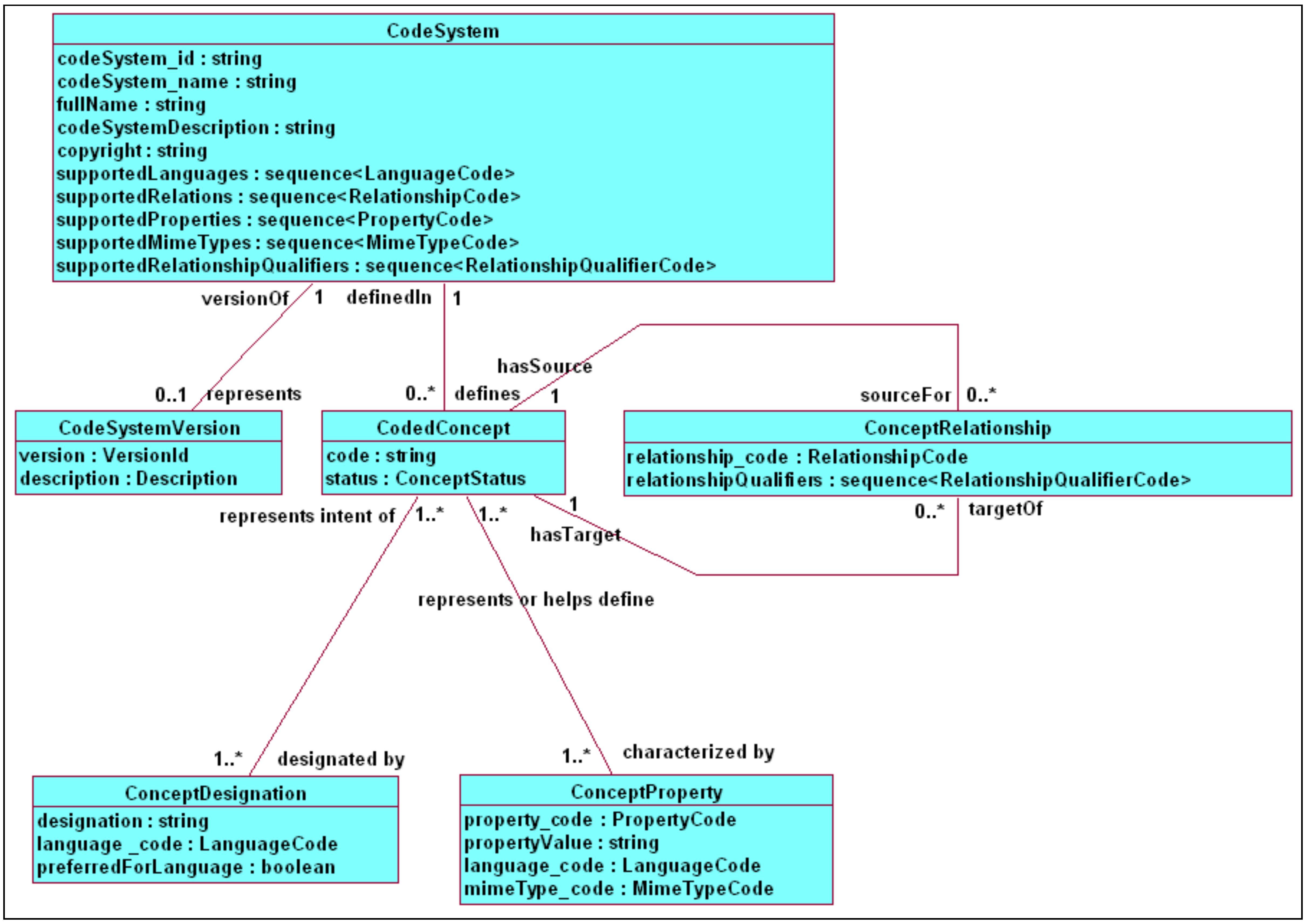
- **UnknownValueSet**
- **ValueSetNameIdMismatch**
- **UnknownConceptCode**
- **UnknownCodeSystem**
- **UnexpectedError**

7 Vocabulary API Model

7.1 Introduction

The following sections describe the model underlying the CTS Vocabulary API. Note that this document does not provide a complete or in-depth model of all the possible entities that might comprise a coding system[8] - it only describes the classes and relationships that have a direct bearing on the contents of HL7 coded attributes from the vocabulary perspective.

7.2 CodeSystem



CodeSystem
codeSystem_id : string
codeSystem_name : string
fullName : string
codeSystemDescription : string
copyright : string
supportedLanguages : sequence<LanguageCode>
supportedRelations : sequence<RelationshipCode>
supportedProperties : sequence<PropertyCode>
supportedMimeTypes : sequence<MimeTypeCode>
supportedRelationshipQualifiers : sequence<RelationshipQualifierCode>
versionOf
1
definedIn
1
hasSource
0..1
represents
0..*
defines
1
sourceFor
0..*
CodeSystemVersion
version : VersionId
description : Description
CodedConcept
code : string
status : ConceptStatus
ConceptRelationship
relationship_code : RelationshipCode
relationshipQualifiers : sequence<RelationshipQualifierCode>
represents intent of
1..*
1..*
1
hasTarget
0..*
targetOf
represents or helps define
1..*
designated by
1..*
characterized by
ConceptDesignation
designation : string
language _code : LanguageCode
preferredForLanguage : boolean
ConceptProperty
property_code : PropertyCode
propertyValue : string
language_code : LanguageCode
mimeType_code : MimeTypeCode

Figure 6: Code Systems

A **CodeSystem** may define zero or more **CodedConcepts**. A **coded concept** represents a class or concept within a particular domain of discourse. Every **CodedConcept** must be defined in exactly one **CodeSystem**. Once defined, the meaning of a coded concept may not change. Existing coded concepts may be retired and new coded concepts may be added, but once defined, the meaning of a coded concept must remain static.

A **CodeSystem** has the following attributes:

- *codeSystem_id* - a globally unique identifier for the code system. Within the context of HL7, a *codeSystemId* should take the form of an ISO Object Identifier (OID). HL7 maintains a registry of code system OIDS, and users are strongly encouraged to register any OID used in these services in the registry.
- *codeSystem_name* - a short token that uniquely identifies the code system within the context of the HL7 RIM. Within HL7, code system names are recorded in a 'code system of code systems' associated with the **CodeSystem** vocabulary domain. The *name* is used strictly for communication between carbon-based life forms, and *codeSystemId* should be used for all computer to computer communication.
- *fullName* - The official name of the **CodeSystem**. For systems that are registered in the HL7 **CodeSystem** vocabulary domain, *fullName* is the preferred English **ConceptDesignation** for the **CodedConcept** matching the **CodeSystem** *name*.
- *codeSystemDescription* - A description of the purpose and content of the **CodeSystem**. For systems that are registered in the HL7 CodeSystem vocabulary domain, *description* is the English **ConceptDescription** of the **CodedConcept** matching the **CodeSystem** *name*.
- *copyright* - An optional copyright notice that, if present, should be displayed whenever the code system is accessed or used.

The following attributes carry additional metadata about a code system:

- *supportedLanguages* - The list of all of the languages that are fully or partially supported byr the code system. A "supported" language is recognized by the code system and at least some of the concept designations or properties are available in that language. All code systems must support at least one language. While the service must list all of the primary language subtags that it supports, it is optional whether it lists secondary languages. (i.e., if it supports "en-UK", it must list "en" but may or may not list "en-UK"). The first language in the supportedLanguages list is treated as the default language for the code system.
- *supportedRelations* - The relations (roles) represented in the code system. Note that the subtype relation is treated as a first-class relationship in this model (code: hasSubtype). A relation code has both a code system identifier and a concept code, so it is possible to draw relation codes from many sources. For the purposes of interoperability, the relation codes should be drawn from the HL7 ConceptRelationship code system - OID 2.16.840.1.113883.5.1088 - whenever possible.
- *supportedProperties* - The property codes supported by the code system. Whenever possible, however, property codes should use the HL7 ConceptProperty code system (OID 2.16.840.1.113883.5.1087).
- *supportedMimeTypes* - A list of the MIME types used in designations, descriptions or properties in the code system. These codes must be drawn from the officially designated HL7 Media Type code system, which is currently OID 2.16.840.1.113883.5.79 - MediaType. The text/plain MIME type must be supported by all code systems.
- supportedRelationshipQualifiers - A list of the relationship qualifiers that are recognized by the code system.

A **CodeSystem** may represent zero or one **CodeSystemVersion** at any given point in time. [9]

7.3 CodedConcept

A **CodedConcept** is unique within the **CodeSystem** that defines it. A **CodedConcept** must be designatedBy at least one **ConceptDesignation**. A **ConceptDesignation** must represent the intent of one or more **CodedConcepts.** [10]**CodedConcepts** may be characterizedBy zero or more **ConceptProperties**. A **ConceptProperty** may represent or define one or more **CodedConcepts**.

A **CodedConcept** may be the sourceFor and/or the targetOf zero or more **ConceptRelationships**. Relationships are described in more detail in a following section.

A **CodedConcept** has the following attributes:

- *code* - an identifier that uniquely names the class or "concept" within the context of the defining **CodeSystem**. Once assigned, the 'meaning' of a concept code should never change. This document makes no assumptions one way or the other regarding more than one **CodedConcept** being assigned the same 'meaning' within a **CodeSystem**.
- *status* - represents the current status of the **CodedConcept** within the **CodeSystem**. Concept status values are drawn from the HL7 ConceptStatus (OID: 2.16.840.1.113883.5.1086) code system. Possible status values include 'proposed', 'active', 'deleted' and 'retired'.

This specification only recognizes the distinction between 'active' or 'not active' at the moment. Concept status and versioning will be more completely addressed in a subsequent version of this specification.

7.4 ConceptDesignation

A **ConceptDesignation** is a name or other textual symbol that represents the intent of zero or more **CodedConcepts**. **ConceptDesignations** are language dependent.

ConceptDesignation has the following attributes:

- *designation* - A string of text that provides an external representation of a CodedConcept.
- *LanguageCode* – a language code that follows the rules described in IETF RFC 3066 – Tag for Identification of Languages. This language code consists of multiple subtags separated by hyphens ('-'). The first subtag identifies the major language code. It must be drawn from ISO 639.2 -Codes for the representation of names of languages--Part 1: Alpha-2 Code whenever possible. If no two character code is available, it may be drawn from ISO 639.2 - Codes for the representation of names of languages--Part 2: Alpha-3 Code. There are also additional escape mechanisms that aren't described further here.
  The second subtag is optional. If present, it must be 2-8 characters in length. If two characters in length, it should contain a country code drawn from ISO 3166-1 Two character country codes. If the subtag is 3-8 characters long, it must come from the IANA language tag registry. Additional subtags serve to further refine the language.
- *preferredForLanguage* - TRUE means that this designation should be used to *represent the intent of* the **CodedConcept** in the specified language when no other contextual information is present. Only one designation may be identified as "preferred" for any given language.

7.5 ConceptProperty

A **ConceptProperty** is an "attribute", "facet" or some other characteristic that may represent or help definethe intended meaning of zero or more **CodedConcepts**. **ConceptProperty** has the following attributes:

- *propertyCode* - A combination of a code system id and concept code that identifies the type of property. Whenever possible, property codes should use the HL7 ConceptProperty code system (OID 2.16.840.1.113883.5.1087).
- *propertyValue* - The textual value of the associated property.
- *language_code* - A language code drawn from the code system officially designated by HL7 for such purposes, which is currently OID 2.16.840.1.113883.6.84 - IETF RFC 3066 – Tag for Identification of Languages. Not all **ConceptProperties** have language codes and, if omitted, it is presumed that the property is of a nature that no language applies.
- *mimeType_code* - A code drawn from the officially designated HL7 Media Type code system, which is currently OID 2.16.840.1.113883.5.79 - MediaType. Not all **ConceptProperties** will have an associated MIME type, text/plain is the default.

7.6 ConceptRelationship

The **ConceptRelationship** class represents binary relationships over the set of **CodedConcepts** defined in a single **CodeSystem**.[11]Each **ConceptRelationship** must have exactly one **CodedConcept** as a source and exactly one **CodedConcept** as a target. The *relationship_code* attribute identifies a directed relation. The HL7 ConceptCodeRelationship code system (OID 2.16.840.1.113883.5.1088) defines the relations that are used within the HL7 Version 3 Vocabulary. The definitions within this code system determine which concept code is the source and which the target, the transitivity, symmetry and reflexivity characteristics of the relation and the term used to represent the inverse of the relationship.
This list of relationship codes include:

Concept Code	IDescription	Trans	Reflexive	Symmetric	Inverse
hasSubtype	An otherwise unspecified subtype relationship.	Yes	No	No	isSubtypeOf
hasPart	An otherwise unspecified partitive relationship.	Yes	Yes	No	isPartOf
smallerThan	A generic ordering relationship.	Yes	No	No	greaterThan

Table 21: Basic Relationship Codes

8 Vocabulary API Specification

8.1 Introduction

The CTS Vocabulary API (CTSVAPI) is divided into two sections - the runtime section, which defines the set of functions that are needed for everyday operation of HL7 Version 3 message software and the browsing section, which is used in the process of defining and creating HL7 vocabulary and values set/vocabulary domain content. The browsing section is presumed to have access to the runtime API, so the functionality of the API is not reproduced in the browsing section. Both sections share the same set of basic type definitions described below.

8.2 Basic Types

The following table lists the basic data types that are used in the CTS Vocabulary Runtime, Vocabulary Browser and Translation APIs.

```
typedef string  OID;

typedef string  Description;

struct CTSVersionId {
        short   major;
        short   minor;
};

typedef OID     CodeSystemId;
typedef sequence<CodeSystemId> CodeSystemIdList;

typedef string  CodeSystemName;

typedef string  ConceptCode;
typedef sequence<ConceptCode> ConceptCodeList;

struct ConceptId {
        CodeSystemId codeSystem_id;
        ConceptCode  concept_code;
};
typedef sequence<ConceptId> ConceptIdList;

typedef string  VersionId;

typedef VersionId       CodeSystemVersion;
typedef sequence<VersionId> CodeSystemVersionList;

typedef sequence<octet> ExpansionContext;
```

Listing 29: Basic Types

- **OID** - An ISO Object Identifier (OID).
- **Description** - A textual description of an object or entity. Some descriptions can be quite lengthy and impelementations will need to take this fact into account
- **CTSVersionId** - The version identifier of a Common Terminology Services (CTS) implementation. Version 1.0 would be major: 1 minor: 0
- **CodeSystemId** - A unique code system identifier. In the HL7 context, this should be the ISO Object Identifier (OID) assigned by HL7, if one exists. This does not preclude, however, the use of other identifiers such as a DCE UUID in a non-HL7 contexts. In this case, however, it is the responsibility of the implementor to reconcile any namespace conflicts.
- **CodeSystemName** - A short mnemonic or name for a code system. CodeSystemName is unique within the HL7 Version 3 namespace, but this cannot be counted on in situations outside of HL7 control.
- **ConceptCode** - A code that uniquely identifies a class or concept within the context of a code system.
- **ConceptId** - the combination of a CodeSystemId and a ConceptCode that is used to provide a globally unique identifier for a concept
- **VersionId** - a unique version identifier. Typically, version identifiers have some sort of ordering that allows people and/or software to determine which order identifiers come in.
- **CodeSystemVersion**- a version identifier for a code system.
- **ExpansionContext** - an opaque sequence of bytes used to transfer context between the service and the client.

8.2.1 Coded Data Elements

This section describes the coded data elements used in the vocabulary API

```
typedef ConceptCode     LanguageCode;
typedef sequence<LanguageCode>  LanguageCodeList;

typedef ConceptCode     MimeTypeCode;
typedef sequence<MimeTypeCode> MimeTypeCodeList;

typedef ConceptCode     ConceptStatusCode;

typedef ConceptCode       PropertyCode;
typedef sequence<PropertyCode> PropertyCodeList;

typedef ConceptCode       RelationshipCode;
typedef sequence<RelationshipCode> RelationshipCodeList;

typedef ConceptCode     MapQualityCode;

typedef ConceptCode     RelationQualifierCode;
typedef sequence<RelationQualifierCode> RelationQualifierCodeList;

typedef ConceptCode     MatchAlgorithmCode;
typedef sequence<MatchAlgorithmCode> MatchAlgorithmCodeList;
```

Listing 30: Vocabulary Coded Elements

- **LanguageCode** – a code for a spoken or written language that follows the rules described in IETF RFC 3066 – Tag for Identification of Languages. This language code consists of multiple subtags separated by hyphens (‘-‘). The first subtag identifies the major language code. It must be drawn from ISO 639.2 -Codes for the representation of names of languages--Part 1: Alpha-2 Code whenever possible. If no two character code is available, it may be drawn from ISO 639.2 - Codes for the representation of names of languages--Part 2: Alpha-3 Code. There are also additional escape mechanisms that aren’t described further here.
  The second subtag is optional. If present, it must be 2-8 characters in length. If two characters in length, it should contain a country code drawn from ISO 3166-1 Two character country codes. If the subtag is 3-8 characters long, it must come from the IANA language tag registry. Additional subtags serve to further refine the language.
- **MimeTypeCode** - A mime type drawn from IETF RFC2045 - Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. HL7 also maintains a subset of the Mime type codes in the Media Type code system, OID 2.16.840.1.113883.5.79 - MediaType.
- **ConceptStatusCode**- the status of a concept within a code system (active, retired, etc.)
- **PropertyCode** - a property that may be associated with concepts in a code system.
- **RelationshipCode** - identifies a particular relation as it occurs in a code system. Relationship codes must be drawn from the HL7 ConceptRelationship code system whenever possible.
- **MapQualityCode** - the general "quality" of a concept mapping (exact, broader, narrower, etc.)
- **RelationQualifierCode** - qualifier for a concept relationship.
- **MatchAlgorithmCode -**a code for an algorithm used for string matching and comparison.

The following table lists the code systems and OIDS that are used in the CTS VAPI messages.

Table 22: Coded Data Element OIDS and Names

VAPI Data Element	Code System OID	Code System Name
LanguageCode	2.16.840.1.113883.6.99	ISO639-1
LanguageCode	2.16.840.1.113883.6.100	ISO639-2
MimeTypeCode	2.16.840.1.113883.5.79	MediaType
ConceptStatusCode	2.16.840.1.113883.5.1086	ConceptStatusCode

PropertyCode	2.16.840.1.113883.5.1087	ConceptProperty
RelationshipCode	2.16.840.1.113883.5.1088	ConceptCodeRelationship
MapQualityCode	2.16.840.1.113883.5.1093	TranslationQuality
RelationQualifierCode	-	-
MatchAlgorithmCode	2.16.840.1.113883.5.1094	MatchAlgorithm

8.2.2 Service Identification Section

The vocabulary runtime and browsing API both inherit a common identification interface.

```
interface Identification {
        string          getServiceName() raises (UnexpectedError);

        string          getServiceVersion() raises (UnexpectedError);

        string          getServiceDescription() raises (UnexpectedError);

        CTSVersionId    getCTSVersion() raises (UnexpectedError);
};
```

Listing 31: Common Identification Interface

The service identification describes the service implementation - its name, version, description and which CTS version it is implementing. It is possible for any identification access call to raise an **UnexpectedError** exception.

- **getServiceName** - returns the name given to the service by the service provider.
- **getServiceVersion**- returns a version identifier specific to the service implementation
- **getServiceDescription** - a description of the service, author, purpose, etc.
- **getCTSVersion** - the specific CTS version that the service implements (e.g. 1.0)

8.2.3 Exceptions

The following table contains the exceptions that can be raised by one or more of the methods described in this section. An exception is an abnormal condition that prevents a method invocation from completing. Exceptions involving communications, operating system, database, etc. errors are not included in the list below, and it is assumed that the mechanisms for handling this type of error will already be addressed by the language and/or communications subsystem used in the implementation.

The exceptions described below assume that the baseline exception includes a text field where the specific details of the exception can be spelled out. The additional attributes below provide further information beyond this basic text.

```
exception UnexpectedError {
        string possible_cause;
};

exception UnknownCodeSystem {
        CodeSystemId            codeSystem_id;
};

exception UnknownConceptCode {
        ConceptCode               concept_code;
};

exception UnknownPropertyCode {
        PropertyCode            property_code;
};

exception UnknownLanguageCode {
        LanguageCode            language_code;
};

exception UnknownRelationshipCode {
        RelationshipCode        relationship_code;
};

exception UnknownRelationQualifier {
        RelationQualifierCode   relationQualifier_code;
};

exception UnknownMimeTypeCode {
        MimeTypeCode            mimeType_code;
};

exception InvalidExpansionContext {
};

exception NoApplicableDesignationFound {
        ConceptId       concept_id;
        LanguageCode    language_code;
};

exception CodeSystemNameIdMismatch {
        CodeSystemId    codeSystem_id;
        CodeSystemName  codeSystem_name;
};

exception BadlyFormedMatchText {
        string  matchText;
};

exception TimeoutError {
};

exception UnknownMatchAlgorithm {
        MatchAlgorithmCode      matchAlgorithm_code;
};
```

Listing 32: Vocabulary API Exceptions

- **UnexpectedError** - An error that could not be anticipated by the specification has occurred. This includes things like memory faults, I/O errors, database access errors and any other sort of unexpected event that prevents successful completion of the API call. possible_cause can carry a more detailed description of what actually occurred.
- **UnknownCodeSystem** - the code system identifier isn't recognized by the service.
- **UnknownConceptCode** - the code system identifier was recognized, but the concept_code wasn't defined in the code system.
- **UnknownPropertyCode** - property_code isn't supported by the code system
- **UnknownLanguageCode** - language_code isn't supported by the code system
- **UnknownRelationshipCode** - relationship_code isn't supported by the code system
- **UnknownRelationQualifier** - relationQualifier_code isn't supported by the code system
- **UnknownMimeTypeCode** - mimeType_code isn't supported by the code system
- **NoApplicableDesignationFound** - no designation could be found for concept_id in language language_code
- **CodeSystemNameIdMismatch** - codeSystem_name doesn't identify the same code system as codeSystem_id, or it doesn't name a code system at all.
- **BadlyFormedMatchText** - matchText can't be parsed by the service
- **TimeoutError** - The time limit specified for the function call has expired.
- **UnknownMatchAlgorithm** - The supplied match algorithm isn't supported by the service.

8.3 The CTS Vocabulary Runtime API

This section describes the attributes and methods of the runtime section of the CTS Vocabulary API.

```
interface Runtime : Identification {
                        ...
};
```

Listing 33: Vocabulary Runtime Service

8.3.1 Code Systems Supported by the API

```
struct CodeSystemIdAndVersions {
        CodeSystemId                    codeSystem_id;
        CodeSystemName                  codeSystem_name;
        string                          copyright;
        CodeSystemVersionList           codeSystem_versions;
};
typedef sequence<CodeSystemIdAndVersions> CodeSystemIdAndVersionsList;
```

Listing 34: CodeSystem Information Structure

- **codeSystem_id** - the unique code system identifier, which is usually the ISO OID.
- **codeSystem_name** - the official name of the code system.
- **copyright** - copyright notice associated with code system, if any.
- **codeSystem_versions**- the version or versions of the code system that are supported by this service. This list may be empty if the code system doesn't support individual version identifiers.

```
CodeSystemIdAndVersionsList     getSupportedCodeSystems(
        in long                         timeout,
        in long                         sizeLimit
        )
        raises ( TimeoutError,
                 UnexpectedError);
```

Listing 35: getSupportedCodeSystems

getSupportedCodeSystems provides a list of all code systems and versions supported by the service.

Parameters:

- **timeout** - The time, in milliseconds, that the client is willing to wait for this operation to be completed. A timeout value of 0 means that there is no time limit on the operation.
- **sizeLimit** - The maximum number of elements that the service may return. If exactly sizeLimit items are returned, the client nust assume that there were additional items that weren't returned. A sizeLimit of 0 indicates that the number of items that may be returned is unlimited.

Exceptions:

- **TimeoutError**
- **UnexpectedError**

8.3.2 Look up Identifying Information About a Code System

```
struct  CodeSystemInfo {
        CodeSystemIdAndVersions         codeSystem;
        string                          fullName;
        Description                     codeSystemDescription;
        LanguageCodeList                supportedLanguages;
        RelationshipCodeList            supportedRelations;
        PropertyCodeList                supportedProperties;
        MimeTypeCodeList                supportedMimeTypes;
        RelationQualifierCodeList       supportedRelationQualifiers;
};
```

Listing 36: lookupCodeSystemInfo return structure

- **codeSystem** - the code system id, name and supported version(s)
- **fullName**- the complete name of the code system
- **codeSystemDescription**- the description of the code system contents
- **supportedLanguages**- a list of all of the languages supported by the service for the code system. A "supported" language is recognized by the code system and at least some of the concept designations or properties are available in that language. All code systems must support at least one language. While the service must list all of the primary language subtags that it supports, it is optional whether it lists secondary languages. (i.e., if it supports "en-UK", it must list "en" but may or may not list "en-UK"). The first language in the supportedLanguages list is treated as the default language for the code system.

- **supportedRelations**- a list of all of the relationship codes supported by the service for the code system. A "supported" relationship is recognized by the code system and the code system is able to determine whether any two concept codes in the code system are or are not related to each other under this relationship. A non-hierarchical code system doesn't need to support any relationships. Subsumption is represented by the hasSubtype relationship code.
- **supportedProperties**- a list of all of the properties supported by the service for the code system. A "supported" property means that the code system recognizes the property and at least one coded concept is associated with this property and an optional value.
- **supportedMimeTypes**- a list of the mime types supported by the code system. All code systems must support the "text/plain" mime type, even if there are no properties associated with it.
- **supportedRelationQualifiers**- a list of the relationship qualifiers supported by the code system (if any).

```
CodeSystemInfo lookupCodeSystemInfo(
        in CodeSystemId         codeSystem_id,
        in CodeSystemName       codeSystem_name
        )
        raises (UnknownCodeSystem,
                CodeSystemNameIdMismatch,
                UnexpectedError);
```

Listing 37: lookupCodeSystemInfo

lookupCodeSystemInfo takes a code system identifier (OID) and/or name and returns a detailed description of the code system and elements that are supported by the service.

Parameters:

- **codeSystem_id** - the iso OID of the code system
- **codeSystem_name** - the unique code system name

Exceptions:

- **UnknownCodeSystem**
- **CodeSystemNameIdMismatch**
- **UnexpectedError**

8.3.3 Validate a Concept Identifier

```
boolean isConceptIdValid(
        in ConceptId            concept_id,
        in boolean              activeConceptsOnly
        )
        raises (UnknownCodeSystem,
                UnexpectedError);
```

Listing 38: isConceptIdValid

isConceptIdValid determines whether the supplied concept identifier is valid.

Parameters:

- **concept_id** - the code system and concept code to validate
- **activeConceptsOnly**- TRUE means that only active concepts are considered valid. FALSE means that any concept code known to the code system is ok.

Exceptions:

- **UnknownCodeSystem**
- **UnexpectedError**

8.3.4 Look up a Designation for a Concept Identifier

```
struct StringAndLanguage {
        string          text;
        LanguageCode    language_code;
};
```

Listing 39: StringAndLanguage Structure

The StringAndLanguage structure contains both the text and associated language code. The two part structure exists because it is possible for the language returned by lookupDesignation and other operations to be different than the requested language. As an example, the client may request the designation for a concept in the en-scouse (English Liverpudlian dialect) and the service may return a designation in unqualified English (en). The client software may need to know that the returned value doesn't exactly match the requested value.

Some programming language bindings may already include the language code as part of the return value (e.g. XML/SOAP). For the sake of consistency, the language_code field should be carried in those bindings even if it is redundant.

```
StringAndLanguage lookupDesignation(
        in ConceptId            concept_id,
        in LanguageCode         language_code
        )
        raises (UnknownLanguageCode,
                UnknownCodeSystem,
                UnknownConceptCode,
                NoApplicableDesignationFound,
                UnexpectedError);
```

Listing 40: lookupDesignation

lookupDesignation returns the most appropriate designation for the supplied concept identifier in the specified language. If language_code is blank or null, the default code system language will be used. lookupDesignation will first attempt to find a designation that exactly matches the supplied language. If no matches are found, it will remove the rightmost subtag, if any and try again. This process will be repeated until a matching designation is found or only the primary subtag remains. As an example, "en-UK-south" would match "en-UK-south", "en-UK" and "en" in that order. It would not match "en-US" or "fr".

Parameters:

- **concept_id** - the code system id and concept code
- **language_code** - the desired language of the returned designation

Exceptions:

- **UnknownLanguageCode**
- **UnknownCodeSystem**
- **UnknownConceptCode**
- **NoApplicableDesignationFound**
- **UnexpectedError**

8.3.4.1 lookupDesignation Matching Rules

RFC 3066, Tags for the Identification of Languages specifies a multipart language code. The first part is derived from the two or three character language codes as specified in ISO 639, with two character codes taking precedence to three in the case of duplicates. The second part of the language code, the subtag, specifies the country, region or other variant.

1. Determine whether the primary language code is supported by the code system and, if not, raise UnknownLanguageCode
2. If a designation with a exactly matching language code and preferredForLanguage set to true is located, return it.
3. If there are any designations that exactly match the language code and preferredForLanguage is set to false return the alphabetically earliest of these designations.
4. If the language code has one or more secondary subtags, remove the rightmost tag and repeat steps 1 and 2
5. If the language code doesn't have a second subtag raise NoApplicableDesignationFound.

8.3.5 Determine Whether Two Concept Codes are Related

```
boolean areCodesRelated(
        in CodeSystemId                codeSystem_id,
        in ConceptCode                 source_code,
        in ConceptCode                 target_code,
        in RelationshipCode            relationship_code,
        in RelationQualifierCodeList   relationQualifiers,
        in boolean                     directRelationsOnly
        )
        raises (UnknownConceptCode,
                UnknownCodeSystem,
                UnknownRelationshipCode,
                UnknownRelationQualifier,
                UnexpectedError);
```

Listing 41: areCodesRelated

Determine whether the supplied concept codes are related

Parameters:

- **codeSystem_id** - the code system of the parent and child codes
- **source_code** - the concept code that occurs as the source of the relationship
- **target_code** - the concept code that occurs as the target of the relationship
- **relationship_code** - the concept code that identifies the relationship
- **relationQualifiers** - an optional list of relationship qualifier codes. If the relationQualifiers list contains one or more qualifier codes, only relationship entries that that have qualifiers that match all of the qualifiers in the list will be considered.
- **directRelationsOnly**- TRUE means test direct relationships only, FALSE means that the transitive closure of the relationship is to be tested if the relationship is transitive. If the relationship is not transitive, the result is the same no matter the setting of this flag.

areCodesRelated returns TRUE if one of the following conditions holds:

1. There is a direct relationship of type relationship_code between source_code and target_code according to the specified code system.
2. There is a direct relationship of type relationship_code between target_code and source_code according to the specified code system and the relationship is defined as symmetric.
3. Source_code equals target code and the relationship is reflexive.
4. directRelationsOnly is FALSE, relationship_code is transitive and there is a relationship of type relationship_code between source_code and target_code in the forward transitive closure of relationship_code starting at source_code.
5. directRelationsOnly is FALSE, relationship_code is transitive and symmetric and there is a relationship of type relationship_code between target_code and source_code in the backwards transitive closure of relationship_code starting at source_code

Exceptions:

- **UnknownConceptCode**
- **UnknownCodeSystem**
- **UnknownRelationshipCode**
- **UnknownRelationQualifier**
- **UnexpectedError**

8.4 The CTS Vocabulary Browsing API

The CTS Vocabulary Browser module may be implemented separately or in combination with the CTS Vocabulary Messaging module. The browsing functions are designed to provide additional capabilities that are more appropriate in an authoring and browsing environment where performance isn't an absolutely critical requirement.

The CTS Vocabulary Browsing API uses the same basic types as the runtime API, which are defined in Basic Types (§ 8.2). This section describes the attributes and methods of the browser section of the CTS Vocabulary API.

```
interface Browser : Identification{

                        ...
};
```

Listing 42: Vocabulary Browser Service

The browser inherits the basic attributes of the Identification section, which is described in Service Identification Section (§ 8.2.2)

8.4.1 Code Systems Supported by the API

```
MatchAlgorithmCodeList                 getSupportedMatchAlgorithms() raises (UnexpectedError);

CodeSystemIdAndVersionsList     getSupportedCodeSystems(
        in long                 timeout,
        in long                 sizeLimit)
        raises (TimeoutError,
                UnexpectedError);
```

Listing 43: Supported Browser Properties

getSupportedCodeSystems provides a list of all code systems and versions supported by the service. The CodeSsytemIdAndVersionsList structure is defined in Code Systems Supported by the API (§ 8.3.1)

- **timeout** - The time, in milliseconds, that the client is willing to wait for this operation to be completed. A timeout value of 0 means that there is no time limit on the operation.
- **sizeLimit** - The maximum number of elements that the service may return. If exactly sizeLimit items are returned, the client assume that there were additional itemes that weren't returned. A sizeLimit of 0 indicates that the number of items that may be returned is unlimited.

Exceptions:

- **TimeoutError**
- **UnexpectedError**

8.4.2 Search for Concept Codes by Designation Text

```
ConceptIdList lookupConceptCodesByDesignation (
        in CodeSystemId                 codeSystem_id,
        in string                       matchText,
        in MatchAlgorithmCode           matchAlgorithm_code,
        in LanguageCode                 language_code,
        in boolean                      activeConceptsOnly,
        in long                         timeout,
        in long                         sizeLimit
        )
        raises (UnknownCodeSystem,
                BadlyFormedMatchText,
                UnknownMatchAlgorithm,
                UnknownLanguageCode,
                TimeoutError,
                UnexpectedError);
```

Listing 44: lookupConceptCodesByDesignation

Return a list of concept identifiers having matching designation(s)and criteria

- **codeSystem_id** - code system to be searched.
- **matchText** - If present and non-null, only vocabulary domains having names that match the text. If matchtext absent or null, all vocabulary domains will be returned.
- **matchAlgorithm_code** - If the matchText is present and non-null, the matchAlgorithm_code determines how the match text is applied. See Text Match Algorithms (§ 6.2.2.1) for details.
- **language_code** - if supplied, restrict the search to designations with this language only. (Default: Search all languages). Language code matches will not be more general than the supplied language_code. If, for example, "en" is passed as the language code, it will match designations with a language of "en", "en-UK", "en-UK-south", etc. If, however, "en-UK-south" is passed as the language code, only concepts having designations in that specific language will be returned.
- **activeConceptsOnly**- TRUE means match active concept codes only, FALSE means match any concept code in the code system (Default: TRUE)
- **timeout** - The time, in milliseconds, that the client is willing to wait for this operation to be completed. A timeout value of 0 means that there is no time limit on the operation.
- **sizeLimit** - The maximum number of elements that the service may return. If exactly sizeLimit items are returned, the client assume that there were additional itemes that weren't returned. A sizeLimit of 0 indicates that the number of items that may be returned is unlimited.

Exceptions:

- **UnknownCodeSystem**
- **BadlyFormedMatchText**
- **UnknownMatchCode**
- **UnknownLanguageCode**
- **TimeoutError**
- **UnexpectedError**

8.4.3 Search for Concept Codes by Property

```
ConceptIdList lookupConceptCodesByProperty (
        in CodeSystemId                 codeSystem_id,
        in string                       matchText,
        in MatchAlgorithmCode           matchAlgorithm_code,
        in LanguageCode                 language_code,
        in boolean                      activeConceptsOnly,
        in PropertyCodeList             properties,
        in MimeTypeCodeList             mimeTypes,
        in long                         timeout,
        in long                         sizeLimit
        )
        raises (UnknownCodeSystem,
                BadlyFormedMatchText,
                UnknownMatchAlgorithm,
                UnknownLanguageCode,
                UnknownPropertyCode,
                UnknownMimeTypeCode,
                TimeoutError,
                UnexpectedError);
```

Listing 45: lookupConceptCodesByProperty

Return a list of concept identifiers having one or more properties with values that match the supplied match text string and other criteria:

- **codeSystem_id** - code system to be searched.
- **matchText** - the text to search for. The format and syntax of matchText depends upon the matchAlgorithmCode.
- **matchAlgorithm_code** - The match algorithm to use when comparing the match text with the target designations. See Text Match Algorithms (§ 6.2.2.1) for details.

- **language_code** - if supplied, restrict the search to designations with this language only. (Default: Search all languages). Language code matches will not be more general than the supplied language_code. If, for example, "en" is passed as the language code, it will match designations with a language of "en", "en-UK", "en-UK-south", etc. If, however, "en-UK-south" is passed as the language code, only concepts having designations in that specific language will be returned.
- **activeConceptsOnly**- TRUE means match active concept codes only, FALSE means match any concept code in the code system (Default: TRUE)
- **properties**- list of properties codes to be searched. (Default: search all properties)
- **mimeTypes**- list of mime types to search. (Default: search all MIME types)
- **timeout** - The time, in milliseconds, that the client is willing to wait for this operation to be completed. A timeout value of 0 means that there is no time limit on the operation.
- **sizeLimit** - The maximum number of elements that the service may return. If exactly sizeLimit items are returned, the client assume that there were additional itemes that weren't returned. A sizeLimit of 0 indicates that the number of items that may be returned is unlimited.

Exceptions:

1. UnknownCodeSystem
2. BadlyFormedMatchText
3. UnknownMatchCode
4. UnknownLanguageCode
5. UnknownPropertyCode
6. UnknownMimeTypeCode
7. TimeoutError
8. UnexpectedError

8.4.4 Return a Complete Description of a Coded Concept

The following structures are all used to build the complete coded concept description return structure

8.4.4.1 ConceptDesignation Structure

```
struct ConceptDesignation {
        string                          designation;
        LanguageCode                    language_code;
        boolean                         preferredForLanguage;
};
typedef sequence<ConceptDesignation> ConceptDesignationList;
```

Listing 46: ConceptDesignation structure

- **designation** - a designation for a coded concept
- **language_code** - the language of the designation
- **preferredForLanguage**- TRUE means that this designation should be used for the supplied language unless other criteria apply. At most, one designation may be marked as preferred for a given concept and language.

8.4.4.2 ConceptProperty Structure

```
struct ConceptProperty {
        PropertyCode                    property_code;
        string                          propertyValue;
        LanguageCode                    language_code;
        MimeTypeCode                    mimeType_code;
};
typedef sequence<ConceptProperty> ConceptPropertyList;
```

Listing 47: ConceptProperty structure

- **property_code** - a concept code that identifies a particular property
- **propertyValue** - the value of this property for a given concept
- **language_code** - the language of the propertyValue (optional)
- **mimeTypeCode** - the MIME type of the propertyValue (default: text/plain)

8.4.4.3 CompleteCodedConceptDescription Structure

```
struct ConceptRelationship {
        ConceptId                       sourceConcept_id;
        RelationshipCode                relationship_code;
        RelationQualifierCodeList       relationQualifiers;
        ConceptId                       targetConcept_id;
};
typedef sequence<ConceptRelationship> ConceptRelationshipList;
```

Listing 48: ConceptRelationship structure

- **sourceConcept_id**- the concept identifier (code system id and concept code) of the source concept in the relationship.
- **relationship_code**- the relationship code.
- **relationQualifiers**-an optional list of qualifier codes that further refine or otherwise qualify the relationship entry.
- **targetConcept_id** - concept identifier (code system id and concept code) of the target of the relationship.

```
struct CompleteCodedConceptDescription {
        ConceptId               concept_id;
        ConceptStatusCode       conceptStatus_code;
        CodeSystemVersion       codeSystem_version;
        ConceptDesignationList  designatedBy;
        ConceptPropertyList     hasProperties;
        ConceptRelationshipList sourceFor;
        ConceptRelationshipList targetOf;
};
```

Listing 49: CompleteCodedConceptDescription structure

- **concept_id** - the code system identifier and concept code being retrieved.

- **conceptStatus_code**- the status of the concept in the given code system version.
- **codeSystem_version**- the version of the code system that the description was drawn from.
- **designatedBy** - a list of all of the designations for the coded concept.
- **hasProperties** - a list of all of the properties of the coded concept excluding the ConceptDesignation properties.
- **sourceFor** - a list of all of the relationships in which this concept plays the role of "source".
- **targetOf** - a list of all of the relationships in which this concept plays the role of "target".

```
CompleteCodedConceptDescription lookupCompleteCodedConcept (
        in ConceptId    concept_id
        )
        raises (UnknownCodeSystem,
                UnknownConceptCode,
                UnexpectedError);
```

Listing 50: lookupCompleteCodedConcept

lookupCompleteCodedConcept returns a structure containing everything that is known about a given concept code (from the CTS perspective).

Parameters:

- **concept_id**- a code system identifier and concept code

Exceptions:

- **UnknownCodeSystem**
- **UnknownConceptCode**
- **UnexpectedError**

8.4.5 Look up Designations for a Specific Concept Code

```
ConceptDesignationList lookupDesignations (
        in ConceptId            concept_id,
        in string               matchText,
        in MatchAlgorithmCode   matchAlgorithm_code,
        in LanguageCode         language_code
        )
        raises (UnknownCodeSystem,
                UnknownConceptCode,
                BadlyFormedMatchText,
                UnknownMatchAlgorithm,
                UnknownLanguageCode,
                UnexpectedError);
```

Listing 51: Look up concept designations

lookupDesignations returns selected designations for the supplied concept. The matching rules for lookupDesignations are identical to those defined in Search for Concept Codes by Designation Text (§ 8.4.2). The ConceptDesignationList return structure is described in ConceptDesignation Structure (§ 8.4.4.1).

- **concept_id** - the code system identifier and concept code to look up
- **matchText** - the text to search for. The format and syntax of matchText depends upon the matchAlgorithmCode.
- **matchAlgorithm_code** - The match algorithm to use when comparing the match text with the target designations. See Text Match Algorithms (§ 6.2.2.1) for details.
- **language_code** - if supplied, restrict the search to this language only. (Default: Search all languages)

Exceptions:

- **UnknownCodeSystem**
- **UnknownConceptCode**
- **BadlyFormedMatchText**
- **UnknownMatchAlgorithm**
- **UnknownLanguageCode**
- **UnexpectedError**

8.4.6 Look up Properties for a Concept Code

```
ConceptPropertyList lookupProperties (
        in ConceptId            concept_id,
        in PropertyCodeList     properties,
        in string               matchText,
        in MatchAlgorithmCode   matchAlgorithm_code,
        in LanguageCode         language_code,
        in MimeTypeCodeList     mimeTypes
        )
        raises (UnknownCodeSystem,
                UnknownConceptCode,
                BadlyFormedMatchText,
                UnknownLanguageCode,
                UnknownPropertyCode,
                UnknownMatchAlgorithm,
                UnknownMimeTypeCode,
                UnexpectedError);
```

Listing 52: Look up concept properties

lookupProperties returns selected properties for the supplied concept. Properties, matchText language_code and mimeTypes properties follow the same rules as those in lookupConceptCodesByProperty. The ConceptPropertyList return structure is described in ConceptProperty Structure (§ 8.4.4.2).

- **concept_id**- the code system identifier and concept code to get the properties for
- **properties**- a list of properties to search. If omitted or empty, all properties are searched.
- **matchText** - the text to search for. The format and syntax of matchText depends upon the matchAlgorithmCode.
- **matchAlgorithm_code** - The match algorithm to use when comparing the match text with the target designations. See Text Match Algorithms (§ 6.2.2.1) for details.
- **language_code** - if supplied, restrict the search to this language only. (Default: Search all languages)
- mimeTypes - list of mime types to search. (Default: search all MIME types)

Exceptions:

- **UnknownCodeSystem**
- **UnknownConceptCode**
- **BadlyFormedMatchText**
- **UnknownMatchAlgorithm**
- **UnknownLanguageCode**
- **UnknownPropertyCode**
- **UnknownMimeTypeCode**
- **UnexpectedError**

8.4.7 Return a List of Related ConceptCodes

```
struct RelatedCode {
        short                   pathLength;
        ConceptCode             concept_code;
        string                  designation;
        RelationQualifierCodeList relationQualifiers;
        boolean                 canExpand;
        ExpansionContext        expansionContext;
};
typedef sequence<RelatedCode> RelatedCodeList;
```

Listing 53: lookupCodeExpansion return structure

Parameters:

- **pathLength** - an integer that defines the distance from the codeToExpand in hops. The root node will always have a path length of 0.
- **concept_code** - related concept code
- **designation** - the preferred designation for code in the appropriate language and usage context, or the default designation if none is stated explicitly as being preferred
- **relationQualifiers** - list of relationship qualifiers that apply to this node
- **canExpand** - TRUE means that there are additional concept codes directly related to concept_code that can be further expanded.
- **expansionContext**- if canExpand is true, this field contains an opaque context that can be used to further expand the code in this node.

```
RelatedCodeList lookupCodeExpansion (
        in ConceptId            expandConcept_id,
        in RelationshipCode     relationship_code,
        in boolean              sourceToTarget,
        in boolean              directRelationsOnly,
        in LanguageCode         designationLanguage_code,
        in long                 timeout,
        in long                 sizeLimit
        )
        raises (UnknownConceptCode,
                UnknownCodeSystem,
                UnknownRelationshipCode,
                UnknownLanguageCode,
                TimeoutError,
                UnexpectedError);
```

Listing 54: Look up a hierarchical code expansion

- **expandConcept_id** - code system identifier and concept code to be expanded. If the relationship_code is "hasSubtype" and souceToTarget is true, the concept code can be omitted, meaning that we want all "root" concepts in the subtype relationship - all concepts that do not appear as the target of one or more hasSubtype relationships.
- **relationship_code** - relationship to be expanded
- **sourceToTarget**- TRUE means expand from source to target. FALSE means expand from target to source.
- **directRelationsOnly**- if TRUE or relationship_code is not transitive, only the direct targets (or sources if sourceToTarget is FALSE) of expandConcept_id are returned. If FALSE and relationship_code is transitive, descendants or ancestors are returned as well.
- **designationlanguage_code** - language that the returned concept designations should be in
- **timeout** - The time, in milliseconds, that the client is willing to wait for this operation to be completed. A timeout value of 0 means that there is no time limit on the operation.
- **sizeLimit** - The maximum number of elements that the service may return. If exactly sizeLimit items are returned, the client assume that there were additional itemes that weren't returned. A sizeLimit of 0 indicates that the number of items that may be returned is unlimited.

A depth-first tree-walk is used to return the descendants or ancestors of the node. if a node occurs in more than one branch, it is replicated for the return structure. If directRelationsOnly is false, the relationship is transitive, and there is a cycle in the relationship structure, expansion will stop at the last non-repeating node. canExpand will be set to true for this node which will allow the client to manually descend the cyclic structure if it chooses, one cycle at a time.

language_code is used to determine which designation to return in the RelatedCode structure. The rules for determining the correct designation are the same as in the lookupDesignation method, except that this method doesn't throw a NoApplicableDesignationFound exception. If there isn't an applicable designation for a particular node, the designation field will contain an empty string.

Exceptions:

- **UnknownCodeSystem**
- **UnknownConceptCode**
- **UnknownRelationshipCode**
- **UnknownLanguageCode**
- **TimeoutError**
- **UnexpectedError**

8.4.7.1 lookupCodeExpansion detail

In the diagram below, the vertices represent an arbitrary transitive relationship with the arrows pointing from the source concepts to the target concepts.

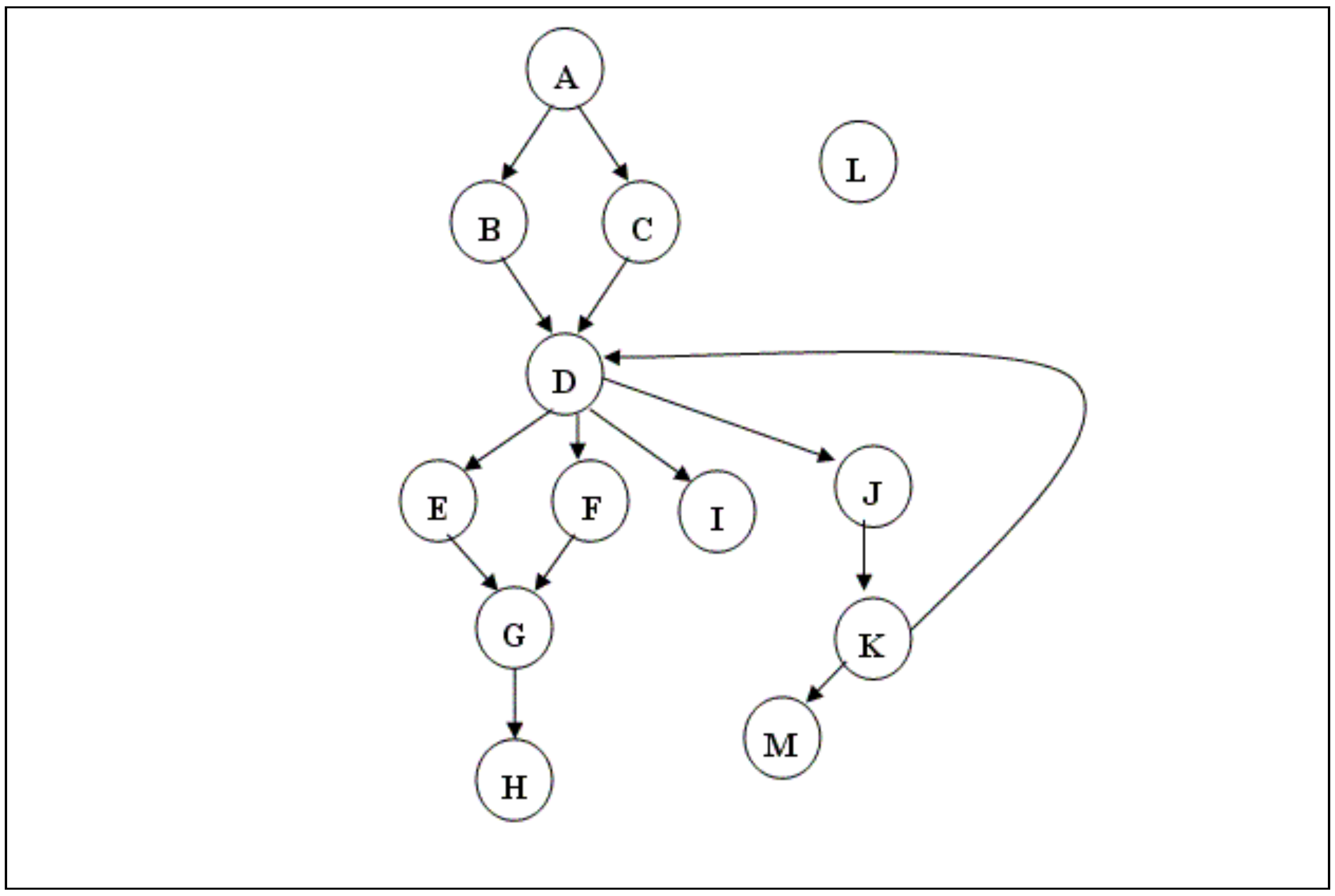


Figure 7: Sample Graph for lookupCodeExpansion examples

The following tables show the values that would be returned from lookupCodeExpansion using the information described in the figure above and the supplied parameters.

pathLength	concept_code	canExpand	expansionContext
1	E	TRUE	(ecE)
1	F	TRUE	(ecF)
1	I	FALSE	-
1	J	TRUE	(ecJ)

Table 23: lookupCodeExpansion(expandConcept_id=D, sourceToTarget=TRUE, directRelationsOnly=TRUE)

pathLength	concept_code	canExpand	expansionContext
1	E	FALSE	-
2	G	FALSE	-
3	H	FALSE	-
1	F	FALSE	-
2	G	FALSE	-
3	H	FALSE	-
1	I	FALSE	-
1	J	FALSE	-
2	K	TRUE	(ecK)
3	M	FALSE	-

Table 24: lookupCodeExpansion (expandConcept_id=D, sourceToTarget=TRUE, directRelationsOnly=FALSE)

Note that node 2K was returned in the example above as expandable because a cycle was detected.

pathLength	concept_code	canExpand	expansionContext
1	B	FALSE	-
2	A	FALSE	-
1	C	FALSE	-
2	K	FALSE	-
1	J	FALSE	-
2	D	TRUE	(ecD)

Table 25: lookupCodeExpaionsion (expandConcept_id=D, sourceToTarget=FALSE, directRelationsOnly=FALSE)

pathLength	concept_code	canExpand	expansionContext
1	A	TRUE	(ecA)
1	L	FALSE	-

Table 26: lookupCodeExpansion (expandConcept_id=, sourceToTarget=TRUE, directRelationsOnly=TRUE)

pathLength	concept_code	canExpand	expansionContext
1	L	FALSE	-
1	A	FALSE	-
2	B	FALSE	-
3	D	FALSE	-
4	E	FALSE	-
5	G	FALSE	-
6	H	FALSE	-
4	F	FALSE	-
5	G	FALSE	-
6	H	FALSE	-
4	I	FALSE	-
4	J	FALSE	-
5	K	TRUE	(ecK)
6	M	FALSE	-
2	C	FALSE	-
3	D	FALSE	-
4	E	FALSE	-
5	G	FALSE	-
6	H	FALSE	-
4	F	FALSE	-
5	G	FALSE	-
6	H	FALSE	-
4	I	FALSE	-
4	J	FALSE	-
5	K	TRUE	(ecK)
6	M	FALSE	-
2	C	FALSE	-

Table 27: lookupCodeExpansion (expandConcept_id =, sourceToTarget=TRUE, directRelationsOnly=FALSE)

pathLength	concept_code	canExpand	expansionContext
1	H	TRUE	(ecH)
1	I	TRUE	(ecI)
1	M	TRUE	(ecM)
1	L	FALSE	-

Table 28: lookupCodeExpansion (expandConcept_id =, sourceToTarget=FALSE, directRelationsOnly=FALSE)

pathLength	concept_code	canExpand	expansionContext
-	-	-	-

Table 29: lookupCodeExpansion (expandConcept_id=L, sourceToTarget=TRUE, directRelationsOnly=FALSE)

8.4.8 Further Expand a Returned Node

```
RelatedCodeList expandCodeExpansionContext(
        in ExpansionContext     contextToExpand
        )
        raises (InvalidExpansionContext,
                TimeoutError,
                UnexpectedError);

};
```

Listing 55: Further expand a previously expanded node

expandCodeExpansionContext further expands a RelatedCodeList node that was returned by a previous lookupCodeExpansion or expandCodeExpansionContext call.

- **contextToExpand**- an opaque identifier that was returned in an expansionContext attribute of a RelatedCodeList node in a previous lookupCodeExpansion or expandCodeExpansionContext call

Exceptions:

- **InvalidExpansionContext**

  Note: expansion contexts are not necessarily permanent. The temporal validity of an expansion context is a function of the specific service implementation.

- **NoApplicableDesignationFound**
- **TimeoutError**
- **UnexpectedError**

8.4.8.1 expandCodeExpansionContext Detail

Extending the examples in Return a List of Related ConceptCodes (§ 8.4.7).
1) expandCodeExpansionContext((expandContext=ecE) using ecE returned by example one in the previous section

pathLength	concept_code	canExpand	expansionContext
1	G	TRUE	(ecG)

Note that the path length is continued from the previous call.
2) expandCodeExpansionContext((expandContext=ecG) using ecG returned by the immediately preceeding call

pathLength	concept_code	canExpand	expansionContext
1	H	FALSE	-

3) expandCodeExpansionContext (expandContext=ecK) following example 2 in the previous section would yield:

pathLength	concept_code	canExpand	expansionContext
3	D	FALSE	-
4	E	FALSE	-
5	G	FALSE	-
6	H	FALSE	-
4	F	FALSE	-
5	G	FALSE	-
6	H	FALSE	-
4	I	FALSE	-
4	J	FALSE	-
5	K	TRUE	(ecK)
6	M	FALSE	-

9 Code Mapping Model

9.1 Introduction

HL7 messaging software may need to map from local concepts to the standardized codes used in the HL7 messaging environment, or between different standardized code sets. The Code Mapping API provides an interface for performing these mappings. Note that the mapping model in this version of the specification is quite limited, and there is no way to map one source code into multiple target codes. It is anticipated that the mapping functionality will be expanded in a later version of this specification.

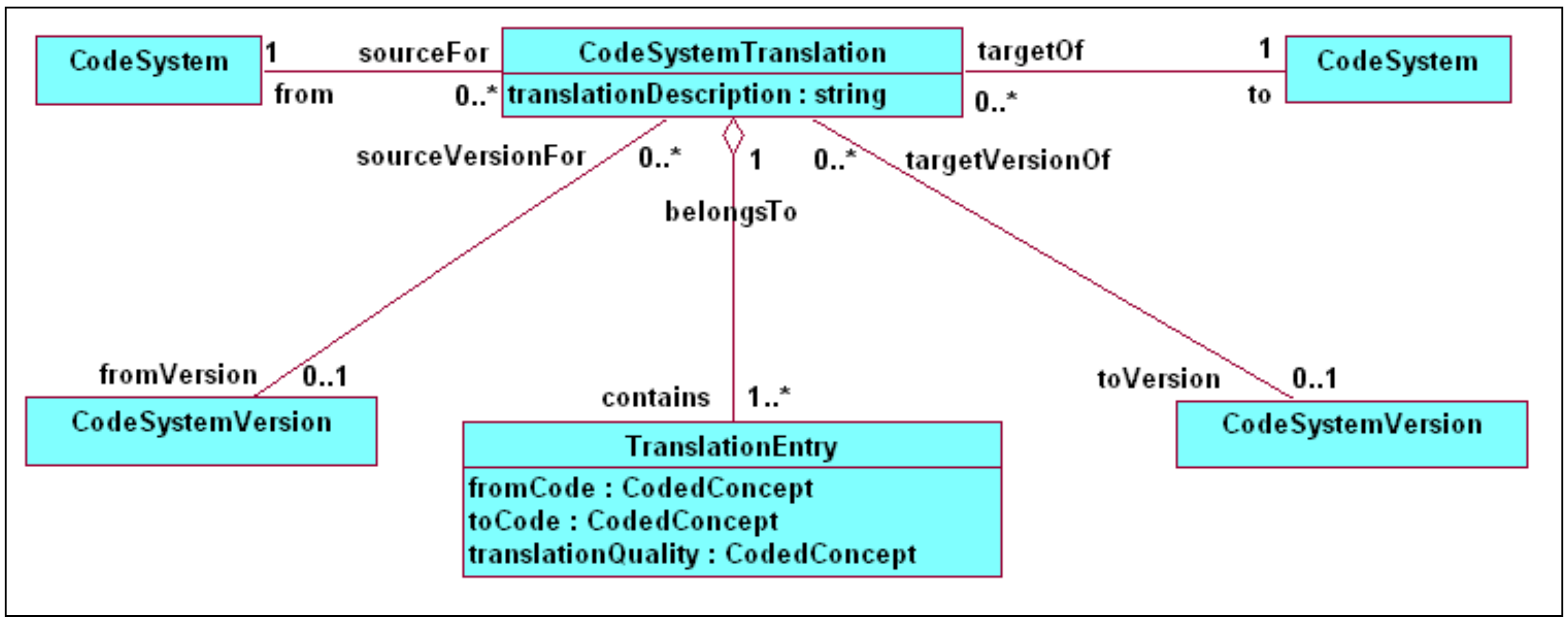


Figure 8: CTS Code Mapping Model

9.2 CTS Code Mapping

A **CodeMap** maps some or all of the concept codes from a source **CodeSystem** to the closest corresponding concept code in the target **CodeSystem**. A **CodeMap** is unidirectional - mapping from the source to the target does not imply the ability to map from the target back to the source. A **CodeMap** may optionally identify the from or to versions of the code systems being translated.

A **CodeMap** 'contains' one or more **MapEntries**. Note that the representation of this in this document is strictly symbolic and is not intended to specify how code mappings are actually performed.

Each **CodeMap** is identified by a unique *map_name*. It is possible for there to be more than one **CodeMap** between the same source and target **CodeSystems**. Each **CodeMap** has an optional *mapDescription* attribute that describes the source of the translation, when it was created, how it was done, etc.

9.3 MapEntry

A **TranslationEntry** represents a translation from the *fromCode* concept code in the source **CodeSystem** to a corresponding *toCode* concept code in the target **CodeSystem**. *mapQuality_code* signifies the 'quality' of the translation from the source code to the target code, and can be one of "exact", "narrower", "broader" or "partial overlap".

10 Code Mapping Specification

10.1 Introduction

The Code Mapping Interface supports the map of concept codes in one or more source code systems into their equivalent codes in a target system.

10.2 Mapping Service Identification

```
interface CodeMapping : Identification {
                        ...
};
```

Listing 56: Code Mapping Identification

10.3 Code Map

```
struct CodeMap {
        string                          map_name;
        Description                     mapDescription;
        CodeSystemId                    fromCodeSystem_id;
        CodeSystemName                  fromCodeSystem_name;
        CodeSystemVersion               fromCodeSystem_version;
        CodeSystemId                    toCodeSystem_id;
        CodeSystemName                  toCodeSystem_name;
        CodeSystemVersion               toCodeSystem_version;

};
typedef sequence<CodeMap> CodeMapList;
```

Listing 57: Code Map Description

a CodeMap identifies a particular mapping

- **map_name -**the unique name of the particular code map.
- **fromCodeSystem_id** - the ISO OID of the source code system .
- **fromCodeSystem_name** - the name of the source code system.
- **fromCodeSystem_version** - the version of the source code system used in the map (optional).
- **toCodeSystem_id** - the ISO OID of the target code system.
- **toCodeSystem_name** - the name of the target code system.
- **toCodeSystem_version** - the version of the target code system used in the map (optional).
- **description** - a description of the map (source, version, date, location, etc.)

```
CodeMapList getSupportedMaps() raises (UnexpectedError);
```

Listing 58: Get Supported Code Maps

getSupportedMaps returns the set of code maps that are supported by this particular service. Note that code maps are not symmetric. The fact that a service supports a map from code system A to code system B does not imply that it also supports a map in the reverse direction.

10.4 Exceptions

The following table contains the exceptions that can be raised by one or more of the methods described in this chapter. An exception is an abnormal condition that prevents a method invocation from completing. Exceptions involving communications, operating system, database, etc. errors are not included in the list below, and it is assumed that the mechanisms for handling this type of error will already be addressed by the language and/or communications subsystem used in the implementation.

The exceptions described below assume that the baseline exception includes a text field where the specific details of the exception can be spelled out. The additional attributes below provide further information beyond this basic text.

```
exception MappingNotAvailable  {
        CodeSystemId            fromCodeSystem_id;
        CodeSystemId            toCodeSystem_id;
};

exception UnableToMap {
};

exception UnknownMapName {
        string                  map_name;
};

exception AmbiguousMapRequest {
        sequence<string>        possible_maps;
};

exception MapNameSourceMismatch {
        CodeSystemId            fromCodeSystem_id;
        CodeSystemId            mapSourceCodeSystem_id;
};

exception MapNameTargetMismatch {
        CodeSystemId            toCodeSystem_id;
        CodeSystemId            mapTargetCodeSystem_id;
};
```

Listing 59: Code Mapping Exceptions

- **UnknownCodeSystem** - the code system identifier isn't recognized by the service.
- **UnknownConceptCode** - the code system identifier was recognized, but the concept_code wasn't defined in the code system.
- **MappingNotAvailable** - the service doesn't support translation between fromCodeSystem_id and toCodeSystem_id.
- **UnableToMap** - the service wasn't able to perform the requested mapping.
- **UnknownMapName** - the supplied map name isn't recognized by the service.
- **AmbiguousMapRequest** - there is more than one possible code map between the supplied source and target code systems. *possible_maps* contains the list of applicable map names.
- **MapNameSourceMismatch** - the code system id supplied in the fromConcept_id parameter didn't match the source code system id in the map named in map_name.
- **MapNameTargetMismatch** - the code system id supplied in the toCodeSystem_id parameter didn't match the target code system id in the map named in map_name.
- **UnexpectedError** - An error that could not be anticipated by the specification has occurred. This includes things like memory faults, I/O errors, database access errors and any other sort of unexpected event that prevents successful completion of the API call. possible_cause can carry a more detailed description of what actually occurred.

10.5 Map a Concept Code

```
struct MappedConceptCode {
        ConceptId               mappedConcept_id;
        MapQualityCode          mapQuality_code;
};
```

Listing 60: mapConceptCode Output Structure

- **mappedConcept_id**- a code system identifier and concept code.
- **mapQuality_code**- a code representing the 'quality' of the mapping operation, such as exact, broader, narrower, etc.

```
MappedConceptCode mapConceptCode(
        in ConceptId                    fromConcept_id,
        in CodeSystemId                 toCodeSystem_id,
        in string                       map_name
        )
        raises ( UnknownCodeSystem,
                 UnknownConceptCode,
                 MappingNotAvailable,
                 UnknownMapName,
                 AmbiguousMapRequest,
                 MapNameSourceMismatch,
                 MapNameTargetMismatch,
                 UnableToMap,
                 UnexpectedError);
```

Listing 61: Map a concept code

mapConceptCode maps the supplied code system id and concept code to the corresponding code (if any) in the target system. fromConcept_id must be supplied. Either toCodeSystem_id, map_name or both may be specified. If only map_name is supplied, toCodeSystem_id is determined by the name. If both parameters are supplied, they must be in agreement.

Parameters:

- **fromConcept_id** - code system id and concept code to be mapped
- **toCodeSystem_id** - code system id of the mapping target
- **map_name** - name of the map to use

Exceptions:

- **UnknownCodeSystem**
- **UnknownConceptCode**
- **MappingNotAvailable**
- **UnknownMapName**
- **AmbiguousMapRequest**
- **MapNameSourceMismatch**
- **MapNameTargetMismatch**
- **UnableToMap**
- **UnexpectedError**

11 CTS Programming Language Bindings (Informative)

11.1 Translating from IDL into Java

The interface signatures for CTS have been specified using the interface definition language as specified in *ISO/IEC 14750:1999 -- Open Distributed Processing -- Interface Definition Language (IDL)*. The Object Management Group (OMG) has specified mappings between IDL and many common programming languages, including ADA, C, C++, COBOL, Java, Lisp, PL/1, Python, and Smalltalk. Bindings also exist between IDL and the Microsoft Common Object Model (COM).

Unfortunately, these language mappings are not completely independent of the underlying CORBA architecture. As an example, in the Java language mapping:

- Exception classes extend org.omg.CORBA.UserException and invoke it as a superclass with [class]Helper.id() as an argument.
- Struct classes implement org.omg.CORBA.portable.IDLEntity
- Interface classes are named [class]Operations.java
- Many additional support files are created ([class]Holder.java, [class]Helper.java., _[class]Stub.java, [class]POA.java, [class]POATie.java

It isn't a difficult task, however, to remove these CORBA remnants, yielding an implementation-neutral interface specification.

The steps used to get from IDL to the target languages of SOAP and Java are:

1. Transform the IDL into Java:
   java com.sun.tools.corba.se.idl.toJavaPortable.Compile -fallTIE -pkgPrefix types org.hl7 -pkgPrefix CTSMAPI org.hl7 -pkgPrefix CTSVAPI org.hl7 CTSVAPI.idl
2. Remove the baseline interface files - those with corresponding 'xxxOperations.java'. As an example, the file 'Browser.java' is removed because there is a corresponding 'BrowserOperations.java'.
3. Remove all of the output files that end in 'Holder.java', 'Helper.java', 'Stub.java', 'POA.java' and 'POATie.java'
4. Replace "extends org.omg.CORBA.UserException" with "extends java.lang.Exception" and remove all super invocations from the exception class constructors.
5. Change the comments from /* to /** so that they are included in the Java doc.
   The references to org.omg.CORBA.portable.IDLEntity are not removed, as they reference an empty class that may still be useful to distinguish different types.

The Figures below show samples of these transformations:

```
/* CTS Specification Version Identifier */
        struct CTSVersionId {
                short   major;
                short   minor;
        };
```

Example 1: Structure Declaration in IDL

```
package org.hl7.CTSVAPI;


/**
* org/hl7/CTSVAPI/CTSVersionId.java .
* Generated by the IDL-to-Java compiler (portable), version "3.1"
* from idl/CTSVAPI.idl
* Monday, March 8, 2004 11:17:26 PM CST
*/


/**
 *$lt;PRE>CTS Specification Version Identifier </PRE>
 */
public final class CTSVersionId implements org.omg.CORBA.portable.IDLEntity
{
  public short major = (short)0;
  public short minor = (short)0;

  public CTSVersionId ()
  {
  } // ctor

  public CTSVersionId (short _major, short _minor)
  {
    major = _major;
    minor = _minor;
  } // ctor

} // class CTSVersionId
```

Example 2: Structure Declaration in Java

```
/*
 * A property code was used that wasn't valid for the code system
 */
        exception UnknownPropertyCode {
                PropertyCode            property_code;
        };
```

Example 3: Exception Declaration in IDL

```
package org.hl7.CTSVAPI;


/**
* org/hl7/CTSVAPI/UnknownPropertyCode.java .
* Generated by the IDL-to-Java compiler (portable), version "3.1"
* from idl/CTSVAPI.idl
* Monday, March 8, 2004 11:17:26 PM CST
*/

public final class UnknownPropertyCode extends java.lang.Exception
{
  public String property_code = null;

  public UnknownPropertyCode ()
  {
    // super(UnknownPropertyCodeHelper.id());
  } // ctor

  public UnknownPropertyCode (String _property_code)
  {
    // super(UnknownPropertyCodeHelper.id());
    property_code = _property_code;
  } // ctor


  public UnknownPropertyCode (String $reason, String _property_code)
  {
    // super(UnknownPropertyCodeHelper.id() + "  " + $reason);
    property_code = _property_code;
  } // ctor

} // class UnknownPropertyCode
```

Example 4: Exception Declaration in Java

```
/*************************************************
 *      Code mapping interface                   *
 *                                               *
 * The code mapping interface represents one     *
 * or more mappings between code systems         *
 *************************************************/
        interface CodeMapping : Identification {

/* List of mappings supported by the service */
                CodeMapList getSupportedMaps() raises (UnexpectedError);


/* Map a concept code from one code system into the closest equivalent (if any)
 * in the target code system
 *      fromConcept_id                   - The code system / concept code to map
 *      toCodeSystem_id                  - The target code system
 *      map_name                         - Name of the map to use.  Can be omitted if there is only one possible map
 *                                         from the fromConcept_id code system to the toCodeSystem_id.
 *
 *      Returns                         - Mapped concept in target system
 *
 *      Exceptions
 *              UnknownCodeSystem       - Either the from or the to code system isn't supported
 *                                        by this map service
 *              UnknownConceptCode      - The concept code to be mapped isn't part of the code system
 *              MappingNotAvailable     - There is not a map from the supplied concept code to the
 *                                        target code system.
 *
 *              UnableToMap             - The requested concept code could not be mapped
 *              UnknownMapName          - mapping_name is not understood by the service
 *              MapSourceMismatch       - source code system id in map didn't match fromConcept_id code system
 *              MapTargetMismatch       - target code system id in map didn't match targetCodeSystem_id in call
 *              AmbiguousMapRequest     - There is more than one possible map between the source concept and target
 *              UnexpectedError         - An unspecified error occurred that prevented successful completion
 *                                        of the request
 */
                MappedConceptCode mapConceptCode(
                        in ConceptId                    fromConcept_id,
                        in CodeSystemId                 toCodeSystem_id,
                        in string                       map_name
                        )
                        raises ( UnknownCodeSystem,
                                 UnknownConceptCode,
                                 MappingNotAvailable,
                                 UnknownMapName,
                                 AmbiguousMapRequest,
                                 MapNameSourceMismatch,
                                 MapNameTargetMismatch,
                                 UnableToMap,
                                 UnexpectedError);
        };
```

Example 5: Interface Declaration in IDL

```
package org.hl7.CTSVAPI;


/**
* org/hl7/CTSVAPI/CodeMappingOperations.java .
* Generated by the IDL-to-Java compiler (portable), version "3.1"
* from idl/CTSVAPI.idl
* Monday, March 8, 2004 11:17:27 PM CST
*/


/*************************************************
 *      Code mapping interface                   *
 *                                               *
 * The code mapping interface represents one     *
 * or more mappings between code systems         *
 *************************************************/
public interface CodeMappingOperations  extends org.hl7.CTSVAPI.IdentificationOperations
{

  /**
 *<PRE>List of mappings supported by the service </PRE>
 */
  org.hl7.CTSVAPI.CodeMap[] getSupportedMaps () throws org.hl7.CTSVAPI.UnexpectedError;

  /**
 *<PRE>Map a concept code from one code system into the closest equivalent (if any)
   * in the target code system
   *      fromConcept_id                   - The code system / concept code to map
   *      toCodeSystem_id                  - The target code system
   *    map_name                           - Name of the map to use.  Can be omitted if there is only one possible map
   *                                         from the fromConcept_id code system to the toCodeSystem_id.
   *
   *      Returns                         - Mapped concept in target system
   *
   *      Exceptions
   *              UnknownCodeSystem       - Either the from or the to code system isn't supported
   *                                        by this map service
   *              UnknownConceptCode      - The concept code to be mapped isn't part of the code system
   *              MappingNotAvailable     - There is not a map from the supplied concept code to the
   *                                        target code system.
   *
   *            UnableToMap             - The requested concept code could not be mapped
   *            UnknownMapName          - mapping_name is not understood by the service
   *            MapSourceMismatch       - source code system id in map didn't match fromConcept_id code system
   *            MapTargetMismatch       - target code system id in map didn't match targetCodeSystem_id in call
   *            AmbiguousMapRequest     - There is more than one possible map between the source concept and target
   *            UnexpectedError         - An unspecified error occurred that prevented successful completion
   *                                        of the request
   </PRE>
 */
  org.hl7.CTSVAPI.MappedConceptCode mapConceptCode (org.hl7.CTSVAPI.ConceptId fromConcept_id, String toCodeSystem_id, String map_name)
                throws org.hl7.CTSVAPI.UnknownCodeSystem, org.hl7.CTSVAPI.UnknownConceptCode, org.hl7.CTSVAPI.MappingNotAvailable,
                       org.hl7.CTSVAPI.UnknownMapName, org.hl7.CTSVAPI.AmbiguousMapRequest, org.hl7.CTSVAPI.MapNameSourceMismatch,
                       org.hl7.CTSVAPI.MapNameTargetMismatch, org.hl7.CTSVAPI.UnableToMap, org.hl7.CTSVAPI.UnexpectedError;
} // interface CodeMappingOperations
```

Example 6: Interface Declaration in Java

11.2 Translating from IDL to WSDL

The Apache Axis 1.1 java2wsdl compiler is used to transform the Java into WSDL. An example invocation method for CTSVAPI would be:

```
java org.apache.axis.wsdl.Java2WSDL -n urn://HL7.org/CTSVAPI -i org.hl7.refImpl.CTSVAPI -
lhttp://localhost:8080/axis/services/CTSVAPIService org.hl7.CTSVAPI
```

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="urn://hl7.org/CTSVAPI" xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:apachesoap="http://xml.apache.org/xml-soap"
                     xmlns:impl="urn://hl7.org/CTSVAPI" xmlns:intf="urn://hl7.org/CTSVAPI" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/
                     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
                     xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <wsdl:types>
                <schema targetNamespace="urn://hl7.org/CTSVAPI" xmlns="http://www.w3.org/2001/XMLSchema">
                        <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
                        <complexType name="SupportedMap">
                                <sequence>
                                        <element name="map_name" nillable="true" type="xsd:string"/>
                                        <element name="mapDescription" nillable="true" type="xsd:string"/>
                                        <element name="fromCodeSystem_id" nillable="true" type="xsd:string"/>
                                        <element name="fromCodeSystem_name" nillable="true" type="xsd:string"/>
                                        <element name="fromCodeSystem_version" nillable="true" type="xsd:string"/>
                                        <element name="toCodeSystem_id" nillable="true" type="xsd:string"/>
                                        <element name="toCodeSystem_name" nillable="true" type="xsd:string"/>
                                        <element name="toCodeSystem_version" nillable="true" type="xsd:string"/>
                                </sequence>
                        </complexType>
                        <complexType name="ArrayOfSupportedMap">
                                <complexContent>
                                        <restriction base="soapenc:Array">
                                                <attribute ref="soapenc:arrayType" wsdl:arrayType="impl:SupportedMap[]"/>
                                        </restriction>
                                </complexContent>
                        </complexType>
                        <complexType name="UnexpectedError">
                                <sequence>
                                        <element name="possible_cause" nillable="true" type="xsd:string"/>
                                </sequence>
                        </complexType>
                        <complexType name="ConceptId">
                                <sequence>
                                        <element name="codeSystem_id" nillable="true" type="xsd:string"/>
                                        <element name="concept_code" nillable="true" type="xsd:string"/>
                                </sequence>
                        </complexType>
                        <complexType name="MappedConceptCode">
                                <sequence>
                                        <element name="mappedConcept_id" nillable="true" type="impl:ConceptId"/>
                                        <element name="mapQuality_code" nillable="true" type="xsd:string"/>
                                </sequence>
                        </complexType>
                        <complexType name="MappingNotAvailable">
                                <sequence>
                                        <element name="fromCodeSystem_id" nillable="true" type="xsd:string"/>
                                        <element name="toCodeSystem_id" nillable="true" type="xsd:string"/>
                                </sequence>
                        </complexType>
                        <complexType name="UnableToMap">
                                <sequence/>
                        </complexType>
                        <complexType name="UnknownConceptCode">
                                <sequence>
                                        <element name="concept_code" nillable="true" type="xsd:string"/>
                                </sequence>
                        </complexType>
                        <complexType name="UnknownCodeSystem">
                                <sequence>
                                        <element name="codeSystem_id" nillable="true" type="xsd:string"/>
                                </sequence>
                        </complexType>
                        <complexType name="UnknownMapName">
                                <sequence>
                                        <element name="map_name" nillable="true" type="xsd:string"/>
                                </sequence>
                        </complexType>
                        <complexType name="AmbiguousMapRequest">
                                <sequence>
                                        <element maxOccurs="unbounded" name="possible_maps" nillable="true" type="xsd:string"/>
                                </sequence>
                        </complexType>
                        <complexType name="CTSVersionId">
                                <sequence>
                                        <element name="major" type="xsd:short"/>
                                        <element name="minor" type="xsd:short"/>
                                </sequence>
                        </complexType>
                </schema>
        </wsdl:types>
        <wsdl:message name="UnknownMapName">
                <wsdl:part name="fault" type="impl:UnknownMapName"/>
        </wsdl:message>
        <wsdl:message name="AmbiguousMapRequest">
```

```
             <wsdl:part name="fault" type="impl:AmbiguousMapRequest"/>
     </wsdl:message>
     <wsdl:message name="getServiceDescriptionResponse">
             <wsdl:part name="getServiceDescriptionReturn" type="xsd:string"/>
     </wsdl:message>
     <wsdl:message name="MappingNotAvailable">
             <wsdl:part name="fault" type="impl:MappingNotAvailable"/>
     </wsdl:message>
     <wsdl:message name="getServiceVersionRequest">

</wsdl:message>
     <wsdl:message name="getSupportedMapsResponse">
             <wsdl:part name="getSupportedMapsReturn" type="impl:ArrayOfSupportedMap"/>
     </wsdl:message>
     <wsdl:message name="UnknownConceptCode">
             <wsdl:part name="fault" type="impl:UnknownConceptCode"/>
     </wsdl:message>
     <wsdl:message name="UnexpectedError">
             <wsdl:part name="fault" type="impl:UnexpectedError"/>
     </wsdl:message>
     <wsdl:message name="UnknownCodeSystem">
             <wsdl:part name="fault" type="impl:UnknownCodeSystem"/>
     </wsdl:message>
     <wsdl:message name="UnableToMap">
             <wsdl:part name="fault" type="impl:UnableToMap"/>
     </wsdl:message>
     <wsdl:message name="getCTSVersionRequest">

</wsdl:message>
     <wsdl:message name="getServiceNameResponse">
             <wsdl:part name="getServiceNameReturn" type="xsd:string"/>
     </wsdl:message>
     <wsdl:message name="getServiceNameRequest">

</wsdl:message>
     <wsdl:message name="getServiceDescriptionRequest">

</wsdl:message>
     <wsdl:message name="mapConceptCodeRequest">
             <wsdl:part name="fromConcept_id" type="impl:ConceptId"/>
             <wsdl:part name="toCodeSystem_id" type="xsd:string"/>
             <wsdl:part name="map_name" type="xsd:string"/>
     </wsdl:message>
     <wsdl:message name="mapConceptCodeResponse">
             <wsdl:part name="mapConceptCodeReturn" type="impl:MappedConceptCode"/>
     </wsdl:message>
     <wsdl:message name="getSupportedMapsRequest">

</wsdl:message>
     <wsdl:message name="getCTSVersionResponse">
             <wsdl:part name="getCTSVersionReturn" type="impl:CTSVersionId"/>
     </wsdl:message>
     <wsdl:message name="getServiceVersionResponse">
             <wsdl:part name="getServiceVersionReturn" type="xsd:string"/>
     </wsdl:message>
     <wsdl:portType name="CodeMappingOperations">
             <wsdl:operation name="getSupportedMaps">
                     <wsdl:input message="impl:getSupportedMapsRequest" name="getSupportedMapsRequest"/>
                     <wsdl:output message="impl:getSupportedMapsResponse" name="getSupportedMapsResponse"/>
                     <wsdl:fault message="impl:UnexpectedError" name="UnexpectedError"/>
             </wsdl:operation>
             <wsdl:operation name="mapConceptCode" parameterOrder="fromConcept_id toCodeSystem_id map_name">
                     <wsdl:input message="impl:mapConceptCodeRequest" name="mapConceptCodeRequest"/>
                     <wsdl:output message="impl:mapConceptCodeResponse" name="mapConceptCodeResponse"/>
                     <wsdl:fault message="impl:AmbiguousMapRequest" name="AmbiguousMapRequest"/>
                     <wsdl:fault message="impl:UnexpectedError" name="UnexpectedError"/>
                     <wsdl:fault message="impl:UnableToMap" name="UnableToMap"/>
                     <wsdl:fault message="impl:MappingNotAvailable" name="MappingNotAvailable"/>
                     <wsdl:fault message="impl:UnknownMapName" name="UnknownMapName"/>
                     <wsdl:fault message="impl:UnknownConceptCode" name="UnknownConceptCode"/>
                     <wsdl:fault message="impl:UnknownCodeSystem" name="UnknownCodeSystem"/>
             </wsdl:operation>
             <wsdl:operation name="getCTSVersion">
                     <wsdl:input message="impl:getCTSVersionRequest" name="getCTSVersionRequest"/>
                     <wsdl:output message="impl:getCTSVersionResponse" name="getCTSVersionResponse"/>
                     <wsdl:fault message="impl:UnexpectedError" name="UnexpectedError"/>
             </wsdl:operation>
             <wsdl:operation name="getServiceDescription">
                     <wsdl:input message="impl:getServiceDescriptionRequest" name="getServiceDescriptionRequest"/>
                     <wsdl:output message="impl:getServiceDescriptionResponse" name="getServiceDescriptionResponse"/>
                     <wsdl:fault message="impl:UnexpectedError" name="UnexpectedError"/>
             </wsdl:operation>
             <wsdl:operation name="getServiceName">
                     <wsdl:input message="impl:getServiceNameRequest" name="getServiceNameRequest"/>
                     <wsdl:output message="impl:getServiceNameResponse" name="getServiceNameResponse"/>
                     <wsdl:fault message="impl:UnexpectedError" name="UnexpectedError"/>
             </wsdl:operation>
             <wsdl:operation name="getServiceVersion">
                     <wsdl:input message="impl:getServiceVersionRequest" name="getServiceVersionRequest"/>
                     <wsdl:output message="impl:getServiceVersionResponse" name="getServiceVersionResponse"/>
                     <wsdl:fault message="impl:UnexpectedError" name="UnexpectedError"/>
             </wsdl:operation>
     </wsdl:portType>
     <wsdl:binding name="CodeMappingServiceSoapBinding" type="impl:CodeMappingOperations">
             <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
             <wsdl:operation name="getSupportedMaps">
                     <wsdlsoap:operation soapAction=""/>
                     <wsdl:input name="getSupportedMapsRequest">
                             <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn://hl7.org/CTSVAPI" use="encoded"/>
                     </wsdl:input>
                     <wsdl:output name="getSupportedMapsResponse">
                             <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn://hl7.org/CTSVAPI" use="encoded"/>
                     </wsdl:output>
                     <wsdl:fault name="UnexpectedError">
                             <wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn://hl7.org/CTSVAPI" use="encoded"/>
                     </wsdl:fault>
             </wsdl:operation>
             <wsdl:operation name="mapConceptCode">
                     <wsdlsoap:operation soapAction=""/>
                     <wsdl:input name="mapConceptCodeRequest">
                             <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn://hl7.org/CTSVAPI" use="encoded"/>
                     </wsdl:input>
                     <wsdl:output name="mapConceptCodeResponse">
                             <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn://hl7.org/CTSVAPI" use="encoded"/>
                     </wsdl:output>
                     <wsdl:fault name="AmbiguousMapRequest">
                             <wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn://hl7.org/CTSVAPI" use="encoded"/>
                     </wsdl:fault>
```

```
				<wsdl:fault name="UnexpectedError">
					<wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn://hl7.org/CTSVAPI" use="encoded"/>
				</wsdl:fault>
				<wsdl:fault name="MapNameSourceMismatch">
					<wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn://hl7.org/CTSVAPI" use="encoded"/>
				</wsdl:fault>
				<wsdl:fault name="MapNameTargetMismatch">
					<wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn://hl7.org/CTSVAPI" use="encoded"/>
				</wsdl:fault>
				<wsdl:fault name="UnableToMap">
					<wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn://hl7.org/CTSVAPI" use="encoded"/>
				</wsdl:fault>
				<wsdl:fault name="MappingNotAvailable">
					<wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn://hl7.org/CTSVAPI" use="encoded"/>
				</wsdl:fault>
				<wsdl:fault name="UnknownMapName">
					<wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn://hl7.org/CTSVAPI" use="encoded"/>
				</wsdl:fault>
				<wsdl:fault name="UnknownConceptCode">
					<wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn://hl7.org/CTSVAPI" use="encoded"/>
				</wsdl:fault>
				<wsdl:fault name="UnknownCodeSystem">
					<wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn://hl7.org/CTSVAPI" use="encoded"/>
				</wsdl:fault>
			</wsdl:operation>
			<wsdl:operation name="getCTSVersion">
				<wsdlsoap:operation soapAction=""/>
				<wsdl:input name="getCTSVersionRequest">
					<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn://hl7.org/CTSVAPI" use="encoded"/>
				</wsdl:input>
				<wsdl:output name="getCTSVersionResponse">
					<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn://hl7.org/CTSVAPI" use="encoded"/>
				</wsdl:output>
				<wsdl:fault name="UnexpectedError">
					<wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn://hl7.org/CTSVAPI" use="encoded"/>
				</wsdl:fault>
			</wsdl:operation>
			<wsdl:operation name="getServiceDescription">
				<wsdlsoap:operation soapAction=""/>
				<wsdl:input name="getServiceDescriptionRequest">
					<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn://hl7.org/CTSVAPI" use="encoded"/>
				</wsdl:input>
				<wsdl:output name="getServiceDescriptionResponse">
					<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn://hl7.org/CTSVAPI" use="encoded"/>
				</wsdl:output>
				<wsdl:fault name="UnexpectedError">
					<wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn://hl7.org/CTSVAPI" use="encoded"/>
				</wsdl:fault>
			</wsdl:operation>
			<wsdl:operation name="getServiceName">
				<wsdlsoap:operation soapAction=""/>
				<wsdl:input name="getServiceNameRequest">
					<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn://hl7.org/CTSVAPI" use="encoded"/>
				</wsdl:input>
				<wsdl:output name="getServiceNameResponse">
					<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn://hl7.org/CTSVAPI" use="encoded"/>
				</wsdl:output>
				<wsdl:fault name="UnexpectedError">
					<wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn://hl7.org/CTSVAPI" use="encoded"/>
				</wsdl:fault>
			</wsdl:operation>
			<wsdl:operation name="getServiceVersion">
				<wsdlsoap:operation soapAction=""/>
				<wsdl:input name="getServiceVersionRequest">
					<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn://hl7.org/CTSVAPI" use="encoded"/>
				</wsdl:input>
				<wsdl:output name="getServiceVersionResponse">
					<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn://hl7.org/CTSVAPI" use="encoded"/>
				</wsdl:output>
				<wsdl:fault name="UnexpectedError">
					<wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn://hl7.org/CTSVAPI" use="encoded"/>
				</wsdl:fault>
			</wsdl:operation>
		</wsdl:binding>
		<wsdl:service name="CodeMappingOperationsService">
			<wsdl:port binding="impl:CodeMappingServiceSoapBinding" name="CodeMappingService">
				<wsdlsoap:address location="http://localhost:8080/axis/services/CodeMappingService"/>
			</wsdl:port>
		</wsdl:service>
</wsdl:definitions>
```

Example 8: Interface Declaration in WSDL

A Annex A: Summary of Service Requirements (Informative)

A.1 Introduction

This section summarizes the broad scope of terminology services which draws on material and experience from the other relevant work identified in section 2.

The detailed proposals in this document address a limited subset of these services which are of most immediate relevance to assist implementation of HL7 Version 3.

A.2 Server information services

A.2.1 Introduction

This section is concerned with services that provide information about the terminology server and the terminologies it supports.

A.2.2 Retrieve information about the capabilities and version of the server

Use case: To enable a client application to interact appropriately with servers supporting different versions of a terminology server specification.

All specifications are subject to evolution and particular implementations may provide additional features. Therefore, a version independent means of establishing the capabilities of a server is essential.

A.2.3 Retrieve information about terminologies accessible via the server

Use case: To enable a client application to establish whether a server can be used to access a specified terminology.

A single instance of the server may provide access to one or more terminologies and a client application may have access to several alternative terminology servers.

This service could be provided either in the form of a test for a specific terminology (E.g. does server support terminology X) or a request for a list of supported terminologies.

A.3 Terminology metadata services

A.3.1 Introduction

This section is concerned with services which access the information about a terminology as supported by the server.

All use cases envisaged in this section require access to a single specified terminology each time a service is invoked. Each request for a service specifies one identified terminology that is available on the server.

A.3.2 Retrieve information about a terminology accessible via the server

Use case: To allow the client application to determine whether the server provides data that is sufficiently complete, up-to-date and authoritative.

The client application must able to access various properties of the available terminologies (e.g. version numbers, release date, authorized source and whether it's available in its entirety or is limited in some way).

A.3.3 Retrieve details of properties of concepts or terms in this terminology

Use case: To allow the client application to determine the properties available for each concept or term in the terminology in order to process or display them in an appropriate way.

Some terminologies have more extensive sets of properties associated with each concept or term. A server may provide access to all these properties or to a limited set of properties. The client application needs to be aware of the properties of a terminology that are accessible via the server.

A.3.4 Retrieve details of relationship types used in this terminology

Use case: To allow the client application to determine the relationship types that are supported by this terminology so that it can make valid requests for relationship information.

Some terminologies support a variety of types of relationships between term and/or concepts. For example:

- several terms may be synonyms representing the same concept.
- a concept may be related to other more general supertype concepts (e.g. "pneumonia" as a subtype of "lung disease" and also a subtype of "infection")
- a concept may be defined in terms of several other concepts.
- a concept may be associated with other concepts which can be used to qualify it.

A.3.5 Retrieve details of search criteria supported for this terminology

Use case: To allow the client application to determine the permissible search criteria that are supported by the server in respect of a particular accessible terminology.

There are a variety of potential search criteria that may be applied to locate a concept or term. Servers may vary in the types of criteria they can support and the criteria appropriate for searching particular accessible terminologies may also vary.

A.4 Terminology content services

A.4.1 Introduction

This section is concerned with services which access the information content of a terminology.

Most use cases envisaged here require access to a single specified terminology each time a service is invoked. Each request for a service specifies one identified terminology that is available on the server. An alternative approach would be to instantiate a session for a particular terminology in which all subsequent service requests apply.

Some use cases may arguably benefit from an option to apply the same service request across all or a selected set of terminologies supported by a server. For example, to search for all terms in all supported terminologies containing a specified phrase. However, this imposes a significant set of complications:

- The server would need to integrate results from various source terminologies
- The terminologies may not share a common structure and may not be searchable in the same ways.
- It requires each retrieved item to identify the terminology at

A.4.2 Retrieve the properties of a concept or term

A service is required which returns the accessible properties of a single concept or term. The required concept or term is selected using it's a unique identifier. The result returned is a set of properties of the selected item. All the properties may be returned or a selected set of properties could be specified in the request.

Use case: To enable a client application to display or validate the textual meaning of a concept or term represented by a unique identifier.

If an identifier is stored or communicated without its associated textual description, this service is essential to enable the client to render the information in a human-readable form. If the text is stored or communicated with the identifier, this service provides a means of validation that the identifier (which is presumable subject to machine processing or retrieval) has the same meaning as the accompanying text.

Use case: To enable the client application to display alternative textual translations of the meaning of the identified concept.

Some terminologies allow multiple terms associated with one concept as synonyms or language/dialect translations. The server should provide the client application with access to these term variants and where they exist.

Use case: To enable the client application to access additional properties related to a concept or term so that it can be processed and/or rendered appropriately.

Some terminologies recognize additional properties associated with each concept and/or term. For example a concept may have a property which indicates if it is intended for current use or retained to support legacy data. A term may have a properties indicate its language and whether it is a preferred term or a synonym.

A.4.3 Retrieve a set of concepts or terms matching specified text criteria

A.4.3.1 General use case

Use case: To allow a concept or term to be found by entering one or more words. Modes of entry for a search may include typing, speech recognition and parsing of selected text in an electronic document. The results should be a set of matches that may be used to populate a menu, list or other user interface control in a client application.

The range of useful search criteria is likely to vary according to the size, structure and richness of the terminology. Criteria are identified below in relation to different use cases.

Performance is critical issue for this service as it may return many terms and concepts for real time display. Return of the complete concept or term with all its associated properties and relationships is unnecessary and may impact performance. By default this service should therefore return only the text of the matching term and the associated unique identifier.

A.4.3.2 Text criteria for retrievals

Various types of text criteria may be valuable according to the size of the terminology (the larger the terminology the more extensive and selective the criteria need to be) and the mode of entry. The list below outlines the main types of text criteria that may be required:

Phrase matches

- Phrase match complete - complete verbatim match between phrase and term
- Phrase match start - a match between phrase and start of term
- Phrase match in term - a match between phrase and any part of the term

Word matches

- Word match complete - the complete word appears somewhere in the term
- Word match start - the (part) word matches the start of a word in the term
- Pattern extended word (the pattern is tested only in the context of a word
- Word form matches (e.g. "tuberculosis" for "tuberculosis")

Multiple words matches

- Apply one of the word match criteria for each word. Variations on multiple word matches
- Multiple word order independent match - All words typed appear somewhere in the term.
- Word order dependent - All words typed appear somewhere in the term.

Pattern matches

- Phrase pattern matches - the term matches a regular expression.
- Word pattern matches - as for word matches but the word is represented by a regular expression.

Modifications applicable to most types of text search

- Case dependent/independent searching
- Accent and special character handling in searches
- Defining word breaks for word searches
- Soundex matching
- Match words/phrases/abbreviations with similar meanings (e.g. "renal" for "kidney", "MI" for "myocardial infarction" or "mitral incompetence")

Text searches may need to be moderated by other criteria noted in the next section.

A.4.3.3 Additional criteria modifying text retrieval

To meet some use cases for text based retrieval additional criteria need to be applied. The following three types of additional criteria may be applicable to some or all terminologies supported by the server.

Status criteria

Use case: To allow a client application to retrieve only concepts or terms in current use.

A terminology may contain concepts or terms that are no longer intended for current use but which are retained to support interpretation of legacy data. Vocabulary domain criteria

Use case: To allow a client application to limit a requested retrieval to concepts or terms in a specified vocabulary domain. For example, to limit the search to values that are permissible in a particular attribute of an HL7 message.

A terminology may contain values applicable to different contexts and these may be permitted for use in different attributes of particular HL7 messages.

Support for these criteria requires the server to have access to the appropriate HL7 vocabulary domain tables.

Relationship criteria

Use case: To allow a client application to limit a requested retrieval to terms or concepts that have a particular relationship between another concept. For example, limit retrieval for the word "appendix" to concepts that have a subtype relationship with "surgical procedure".

A terminology may contain relationships which provide a useful mechanism for refining a search. Support for this type of search will only be available for some terminologies and the degree of sophistication of the criteria will vary according to the nature of the supported relationships.

A.4.4 Retrieve a set of concepts or terms specified by relationships

A.4.4.1 Retrieve all relationships for an identified concept or term

Use case: To allow the client application to display the relationships, make semantic deduction or offer choices of valid qualifiers that can be applied to a concept.

This complete retrieval option may be efficient when many relationships need to be reviewed by the client application. However, the following pair of more selective relationship retrieval services may be more efficient for specific review and navigation.

A.4.4.2 Retrieve the relationship types available for an identified concept or term

Use case: To allow the client application to determine which relationships are available prior to selective retrieval of concepts with a specified relationship (see below)

This offers no additional functionality compared with retrieving all relationships (see above). However, it may be more efficient for identifying relationship types to be processed in more individually (see below).

Relationship types may include some that are specific to a terminology. However, some general types of relationship should be represented in a common way by this service if they are supported by the terminology:

- The subtype relationship
- The relationship between a concept and its synonymous terms.

A.4.4.3 Retrieve concepts/terms with a specified relationship type to a concept/term

Use case: To allow the client application to offer the user options to navigate across specified semantic links, for example, to refine a concept or to generalize a concept.

Use case: To allow the client application to display synonyms associated with a concept.

Use case: To allow the client application to make semantic deductions based on subtype relationships, for example, to count instances of "pneumonia" as instances of "lung disease" and "infection".

This offers no additional functionality compared with retrieving all relationships (see above). However, this selective service may be more efficient for specific review and navigation.

By default this service should return the unique identifier and term for each concept or term.

A.4.5 Retrieve all concepts or terms in a particular vocabulary domain

Use case: To allow the client application to access the complete set of values that may be applicable to a specified attribute in an HL7 message.

This service is closely related to the text search with additional vocabulary domain criteria. However, in this case all concepts in the vocabulary domain are returned without any attempt at text matching.

By default this service should return the unique identifier and term for each concept or term.

A.4.6 Validate a unique identifier

A.4.6.1 Check that a unique identifier is valid in the terminology

Use case: To allow a client application to validate the content of a message by ensuring that identifiers (or codes) in the message are valid in the referenced terminology.

This use case can also be met by attempting to retrieve the required concept or term. However, requesting retrieval created a processing and data overhead when compared with a simple test. Validation may be required for many messages each containing many identifiers so a light and efficient validation service is required.

A.4.6.2 Additional criteria applicable to identifier validation

Status criteria

Use case: To allow a client application to test if an identifier in a message or record is in current use.

A terminology may contain concepts or terms that are no longer intended for current use but which are retained to support interpretation of legacy data. Vocabulary domain criteria

Use case: To allow a client application to conform that an identifier in a message is a value within the appropriate vocabulary domain for that message. For example, to limit the search to values that are permissible in a particular attribute of an HL7 message.

A terminology may contain values applicable to different contexts and these may be permitted for use in different attributes of particular HL7 messages.

Support for these criteria requires the server to have access to the appropriate HL7 vocabulary domain tables.

Relationship criteria

Use case: To allow a client application to test whether a concept should be counted as a subtype of another concept (e.g. a concept used in a decision support algorithm or as a criteria for analysis).

A terminology may contain relationships which can be used to determine whether a concept is a subtype of another concept.

A.4.7 Mapping and transformation of codes and code phrases

A.4.7.1 Retrieve identifier with the same meaning in another terminology

Use case: To allow a client application to translate an identifier used in its native environment into the terminology required in a message (and vice versa).

In many cases this is a non-trivial task as maps may be one-to-many and may require other contextual information to select a specific value.

However, for vocabulary domains with a relatively small number or possible values with know one-to-one (or many-to-one) maps lists this is a useful and service.

A.4.7.2 Decompose a concept into a post-coordinated expression

Use case: To allow a client application to break down a complex concept into constituent codes which represent different aspect of the meaning of that concept. For example, to represent "appendectomy" as "surgical removal" applied to the "vermiform appendix".

Some terminologies which include both complex and simple concepts provide relationships that support this transformation. This service may be useful for populating messages which represent different aspects on a complex concept in separated coded elements. The end result of this service is an expression (or code phrase) consisting of attribute value pairs (possibly nested). The client application then has to select which elements are represented in which message attributes.

This service may not be strictly necessary in this form. An alternative approach is to retrieve specific relationship types for each message attribute to be populated.

A.4.7.3 Compose a post-coordinated expression into a pre-coordinated concept

Use case: To allow a client application to convert a collection of identifiers representing different parts of a more complex concept into a single concept identifier.

This is the inverse of the decomposition service. It may not be possible to compose an expression into a single concept as there may be no single concept that represents the full meaning of the post-coordinated expression. Therefore the result of this may be the same post-coordinated expression or another post-coordinated expression with fewer sub-elements. For example, "surgical removal" or "vermiform appendix" as an "emergency" might be composed into "appendectomy" as and "emergency".

B Annex B: Code Systems Used in the CTS API (Informative)

Table 30:

OID	Code System Name	Description	Sample Values
CodeSystem	2.16.840.1.113883.5.22	The set of code systems known to HL7	CAS (Chemical abstract codes) IETF1766 (Language identifiers) CodeSystem (Code System)
ConceptStatus	2.16.840.1.113883.5.1086	The status of a domain entry as pertains to its review and incorporation into the HL7 domain specification database.	A (Active) D (Deleted) P (Proposed) R (Retired)
VocabularyDomainQualifier	2.16.840.1.113883.5.147	The extensibility of coding determines whether or not exceptions are allowed in the domain of a coded attribute	CWE (coded with exceptions) CNE (coded no exceptions)
Language	2.16.840.1.113883.6.99	The human language that is used in textual descriptions or communications. From ISO 639.	EN (English) DE (German) EN-US (English - US dialect) EN-GB (English - GB dialect)
TranslationQuality	2.16.840.1.113883.5.1093	The relationship between concepts between two code systems.	Exact Broader Than Narrower Than Different
ConceptProperty	2.16.840.1.113883.5.1087	Concept property identifiers	openIssue appliesTo howApplies OID
ConceptCodeRelationship	2.16.840.1.113883.5.1088	The relationship between two concepts in the same code system.	hasSubtype hasPart smallerThan
MatchAlgorithm	2.16.840.1.113883.5.1094	The match algorithm to use when searching text.	Identical IdenticalIgnoreCase StartsWith StartsWithIgnoreCase EndsWith EndsWithIgnoreCase ContainsPhrase ContainsPhraseIgnoreCase WordsAnyOrder WordsAnyOrderIgnoreCase WildCards WildCardsIgnoreCase RegularExpression

Concept Code	Designation	Description	Transitive	Inverse
hasSubtype	has subtype	A generic subsumption relationship. This relationship includes both exhaustive and non-exhaustive subtypes as well as mutually exclusive vs. overlapping.	True	isSubtypeOf
hasPart	has part	A generic partitive relationship with no further details specified.	True	isPartOf
smallerThan	is smaller	A type of ordering relationship indicating quantitatively fewer entities	True	greaterThan

	than			

Table 31: VocabularyConceptRelationship Domain

C Annex C: IDL Specification for the CTS API (Normative)

The following listings contain the complete IDL used to generate the CTS API

C.1 Basic HL7 Data Types

- types.idl - IDL for Basic HL7 Data Types used by CTS

C.2 IDL Definition of CTS Message API

- CTSMAPI.idl - IDL Definition of CTS Message API

C.3 IDL Definition of CTS Vocabulary API

- CTSVAPI.idl - IDL Definition of CTS Vocabulary and Code Mapping API

D Annex D: XML Binding of the CTS API (Normative)

The following sections containn the WSDL binding of the 5 CTS API modules. Both the IDL description of the API and the WSDL binding are normative descriptions of the CTS API

D.1 WSDL Binding for the CTS Message Runtime API

- MessageRuntime.wsdl - CTS Message Runtime Service

D.2 WSDL Binding for the CTS Message Browser API

- MessageBrowser.wsdl - CTS Message Browser Service

D.3 WSDL Binding for the CTS Vocabulary Runtime API

- VocabRuntime.wsdl - CTS Vocabulary Runtime Service

D.4 WSDL Binding for the CTS Vocabulary Browser API

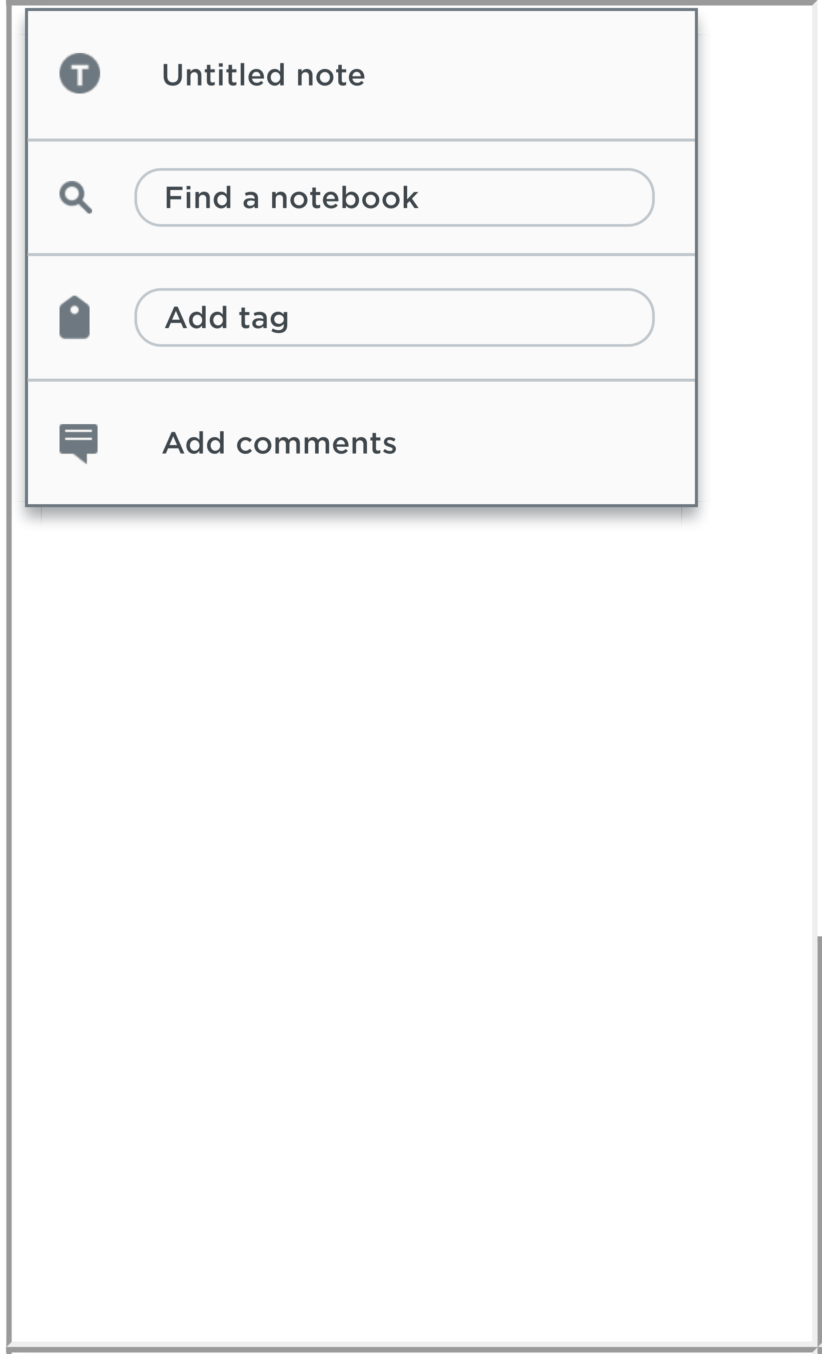
- VocabBrowser.wsdl - CTS Vocabulary Browser Service

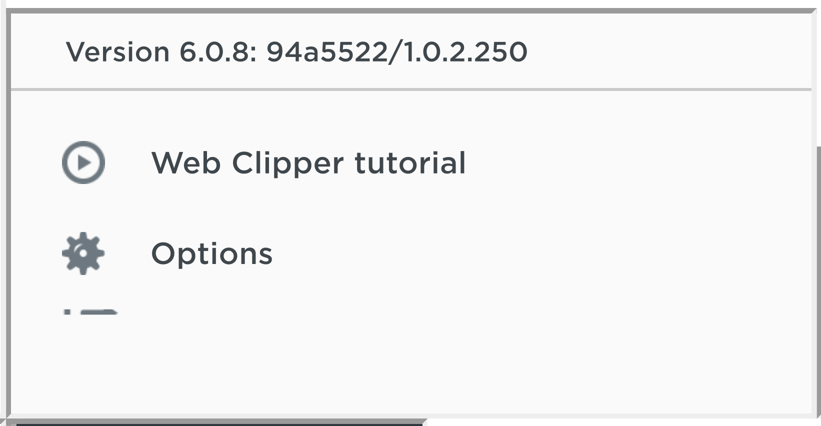
D.5 WSDL Binding for the CTS Code Mapping API

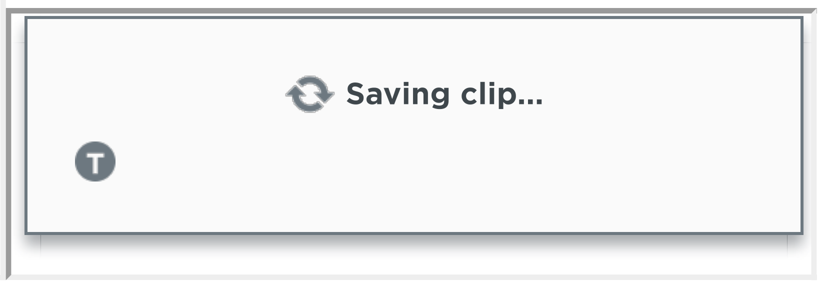
- CodeMapping.wsdl - CTS Code Mapping Service

Endnotes

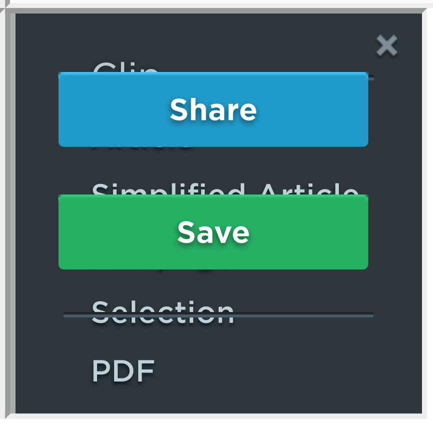
1. [source] While it is intended to be generic, it represents a distinct subset of the sort of capabilities that a generalized vocabulary API should provide. In addition, much of the nomenclature used in the CTS Vocabulary API is HL7-centric and would have to under go translation were it to be put into more generalized use.
2. [source] When used in this context, the word "terminology" is intended to describe any organized set of codes, including the entities commonly referred to as "code sets", "ontologies", "vocabularies", "classification systems", etc.
3. [source] Within this document, the term "code system" refers to a system of codes, descriptions, designations, properties and relationships. Other common names for this entity include "vocabulary", "terminology", "coding scheme", "classification scheme" and "ontology".
4. [source] Note: In this context, constraint refers to constraints on model itself, not attribute domains.
5. [source] This will most likely be an ISO OID, rooted at 2.16.840.1.113883.1.11, but this decision has not be finalized.
6. [source] It is possible for a constructed value set to reference other value sets that are drawn from more than one code system. A given value set may, howver, only directly reference concepts drawn from a single code system.
7. [source] This document doesn't address the issue of how to reference a specific version of a code system or the state of a value set at a particular point in time. These topics will be covered in a subsequent release of the specification.
8. [source] Within this document, the term "code system" is used to designate a system of codes, descriptions, designations, properties and relationships. Other terms that are often used to designate this entity include "vocabulary", "terminology", "coding scheme", "classification scheme" and "ontology".
9. [source] The ability to access past release versions of a code system will be implemented in a subsequent revision of this specification.
10. [source] The reason that a designation is shown as associated with more than one coded concept is to stress the fact that it is possible for more than one coded concept to share the same designation *text*. The model does not making any assertions about whether a designation is a "first class object" - that is whether it has an identity beyond the textual portion.
11. [source] This model is not complete and does not fully represent of the semantics of ontology and terminology systems. The assertion of a relationship between two coded terms is obviously not sufficient to describe the semantic implication of the relationship to either the associated coded terms or instances thereof. It is also important to note that this document makes no statement on the implications of the absence of an asserted relationship between two codes. It is left to the discretion of the individual terminology providers to fill in the additional semantics as is appropriate for their own model and content.


Untitled note
Find a notebook
Add tag
Add comments


Version 6.0.8: 94a5522/1.0.2.250
Web Clipper tutorial
Options


Saving clip...

To:


Clip
Share
Simplified Article
Save
Selection
PDF