LexEVS 5.x Design and Architecture Guide

LexEVS 5.x Migration LexEVS model > LexEVS 5.x Migration LexEVS database > LexEVS 5.x Migration EVS API to LexEVS API > LexEVS 5.x Migration OWL loader > LexEVS 5.x Design and Architecture Guide

Contents [hide]

Introduction

This document is intended for LexEVS developers.

Document sections

1. LexGrid Model
2. LexBIG Extensions
3. LexEVS Information Models
4. LexEVS Architecture

Related documents

- Programmer's Guide for information about using the LexEVS core services and the APIs.

Service integration with caBIG: diagrams

LexEVS software architecture and implementation is designed to facilitate flexibility and future expansion. The following diagrams are intended to aid the understanding of LexEVS service integration in context of the larger caBIG® universe and specific deployment scenarios:

vocabkc contents
- Main Page
- What's New
- Forums
- Bugzilla
- Code Repository
- Feedback
- Contact Us

tools
- LexBIG/LexEVS
- LexWiki
- NCI Protégé
- Related Tools and Models

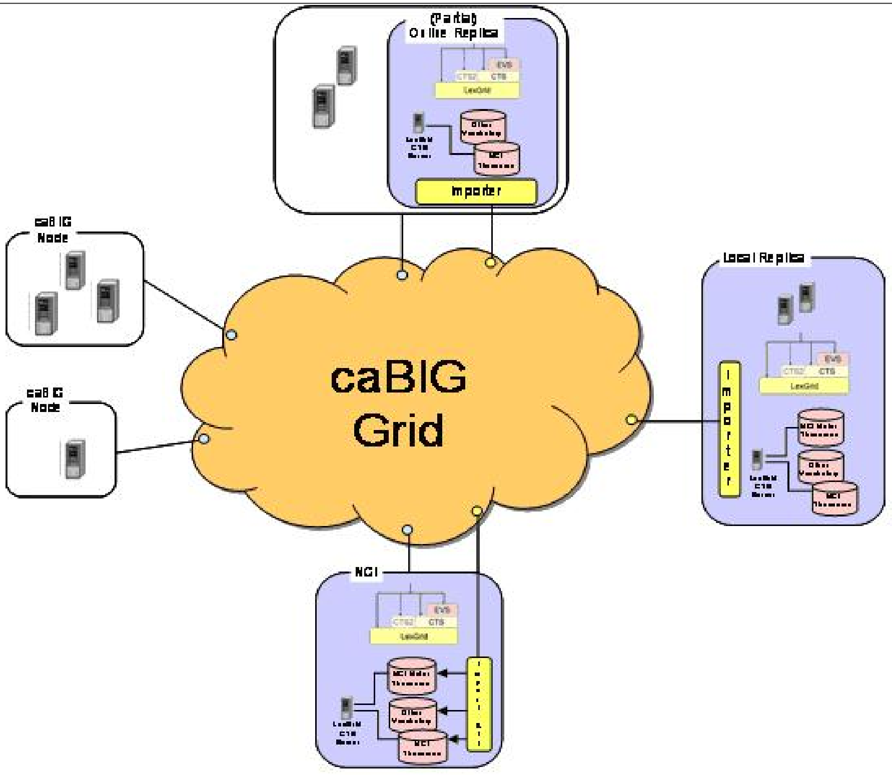
projects
- LexAjax
- LexGrid
- Cancer Data Standards Repository (caDSR)
- Common Terminology Criteria for Adverse Events (CTCAE)
- Open Health Natural Language Processing (OHNLP) Consortium
- Ontology Development and Information Extraction (ODIE)

semantic infrastructure
- SI Main Page
- Initiatives
- Requirements

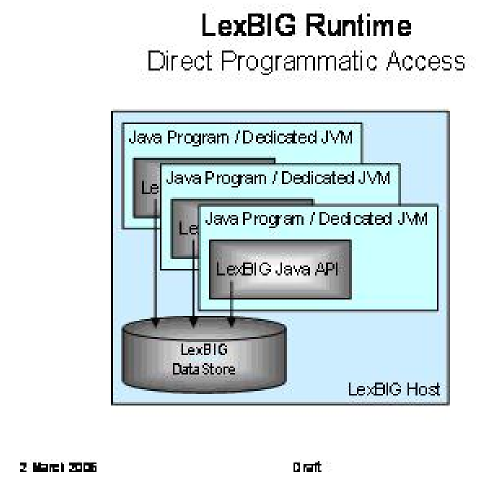
other resources
- Library of Documents
- Documentation and Training for Tools
- Index of Terminologies
- Standards and Standards Influencing Organizations
- Outreach

external links
- VCDE Workspace
- caBIG® Community Website
- caBIG® Support Service Providers

help
- Editing Wiki Pages

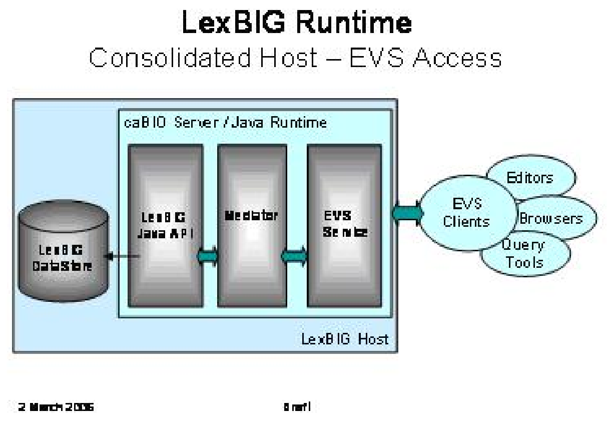


This diagram depicts the LexBIG vision. Individual Cancer Centers will be able to use the existing set of caCORE EVS services. If desired, local instances of vocabularies can be installed.


LexBIG Runtime
Direct Programmatic Access
Java Program / Dedicated JVM
Java Program / Dedicated JVM
Java Program / Dedicated JVM
LexBIG Java API
LexBIG Data Store
LexBIG Host
2 March 2006
Draft

This diagram depicts direct Java-to-Java access to LexBIG functions. This is the primary deployment scenario for phase 1.

Note: It is not required that the database be located on the same system as the program runtime.

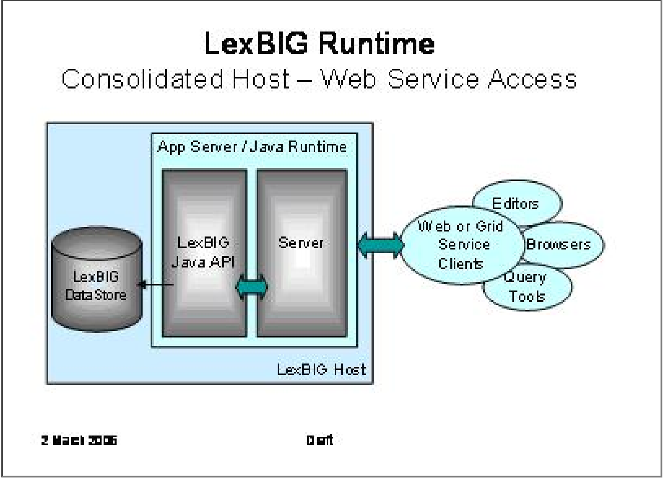


This diagram depicts access through caCORE Enterprise Vocabulary Services (EVS) to a LexBIG vocabulary engine.

The primary goal is to provide a compatible experience for existing EVS browsers and client applications.

Note: this diagram shows the possible inclusion of a mediation layer between EVS and the LexBIG runtime.

This would be done to facilitate alternate communications with the LexBIG server (e.g. through web services as described below).

 LexBIG Runtime Consolidated Host – Web Service Access App Server / Java Runtime LexBIG Java API Server LexBIG Data Store LexBIG Host Editors Web or Grid Service Clients Browsers Query Tools 2 March 2006 Draft	The LexBIG API is designed with web and grid-level enablement in mind. This diagram depicts deployments that wrap the current API to allow the runtime to be accessed through web or grid services.

What is LexGrid?

LexGrid is an initiative of the Mayo Clinic Division of Biomedical Informatics that focuses on the representation, storage, and dissemination of vocabularies. This effort centers on, but is not limited to, the domain of medical vocabularies and nomenclatures. Focal points of the LexGrid project include the development and promotion of standards, tools, and content that:

- Provide flexibility to represent yesterday's, today's and tomorrow's terminological resources using a single information model.
- Provide the ability for these resources to be published online, cross-linked, and indexed.
- Provide standardized building blocks and tools that allow applications and users to take advantage of the content where and when it is needed.
- Provide consistency and standardization required to support large-scale terminology adoption and use.

Additional information for LexGrid is available at http://informatics.mayo.edu .

What is LexBIG?

LexBIG is a more specific project that applies LexGrid vision and technologies to requirements of the caBIG® community. The goal of the project is to build a vocabulary server accessed through a well-structured application programming interface (API) capable of accessing and distributing vocabularies as commodity resources. The server is to be built using standards-based and commodity technologies. Primary objectives for the project include:

- Provide a robust and scalable open source implementation of EVS-compliant vocabulary services. The API specification will be based on but not limited to fulfillment of the caCORE EVS API. The specification will be further refined to accommodate changes and requirements based on prioritized needs of the caBIG® community.
- Provide a flexible implementation for vocabulary storage and persistence, allowing for alternative mechanisms without impacting client applications or end users. Initial development will focus on delivery of open source freely available solutions, though this does not preclude the ability to introduce commercial solutions (e.g. Oracle).
- Provide standard tooling for load and distribution of vocabulary content. This includes but is not limited to support of standardized representations such as UMLS Rich Release Format (RRF), the OWL web ontology language, and Open Biomedical Ontologies (OBO) .

The goal for the initial year of development was to achieve the Bronze level of compatibility with regard to the caBIG® requirements. Silver-level compatibility is being pursued.

Categories: VKC Contents | Documentation | LexEVS

LexEVS 5.x Design and Architecture Guide LexGrid Model

Contents [hide]

Introduction

This document is a section of the Design and Architecture Guide.

LexGrid model overview

The LexGrid Model is Mayo's proposal for standard storage of controlled vocabularies and ontologies. The LexGrid Model defines how vocabularies should be formatted and represented programmatically, and is intended to be flexible enough to accurately represent a wide variety of vocabularies and other lexically-based resources. The model also defines several different server storage mechanisms and an XML format. This model provides the core representation for all data managed and retrieved through the LexBIG system, and is now rich enough to represent vocabularies provided in numerous source formats such as OWL (NCI Thesaurus) and RRF (NCI MetaThesaurus).

Once the vocabulary information is represented in a standardized format, it becomes possible to build common repositories to store vocabulary content and common programming interfaces and tools to access and manipulate that content. The LexBIG API developed for caBIG® is one such interface, and is described in additional detail in LexBIG APIs.

Following are some of the higher-level objects incorporated into the model definition:

Code systems

Each service defined to the LexGrid model can encapsulate the definition of one or more vocabularies. Each vocabulary is modeled as an individual code system, known as a *codingScheme*. Each scheme tracks information used to uniquely identify the code system, along with relevant metadata. The collection of all code systems defined to a service is encapsulated by a single *codingSchemes* container.

Concepts

A code system may define zero or more coded concepts, encapsulated within a single container. A concept represents a coded entity (identified in the model as a *concept*) within a particular domain of discourse. Each concept is unique within the code system that defines it. To be valid, a concept must be qualified by at least one designation, represented in the model as a *property*. Each property is an attribute, facet, or some other characteristic that may represent or help define the intended meaning of the encapsulating concept. A concept may be the source for and/or the target of zero or more relationships. Relationships are described in more detail in a following section.

Relations

Each code system may define one or more containers to encapsulate relationships between concepts. Each named relationship (e.g. "hasSubtype" or "hasPart") is represented as an *association* within the LexGrid model. Each relations container must define one or more association. The association definition may also further define the nature of the relationship in terms of transitivity, symmetry, reflexivity, forward and inverse names, etc. Multiple instances of each association can be defined, each of which provide a directed relationship between one source and one or more target concepts.

vocabkc contents
- Main Page
- What's New
- Forums
- Bugzilla
- Code Repository
- Feedback
- Contact Us

tools
- LexBIG/LexEVS
- LexWiki
- NCI Protégé
- Related Tools and Models

projects
- LexAjax
- LexGrid
- Cancer Data Standards Repository (caDSR)
- Common Terminology Criteria for Adverse Events (CTCAE)
- Open Health Natural Language Processing (OHNLP) Consortium
- Ontology Development and Information Extraction (ODIE)

semantic infrastructure
- SI Main Page
- Initiatives
- Requirements

other resources
- Library of Documents
- Documentation and Training for Tools
- Index of Terminologies
- Standards and Standards Influencing Organizations
- Outreach

external links
- VCDE Workspace
- caBIG® Community Website
- caBIG® Support Service Providers

help
- Editing Wiki Pages

- Editing Forum Posts
- Contact Us

search

Search

toolbox

- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link
- Print as PDF

Source and target concepts may be contained in the same code system as the association or another if explicitly identified. By default, all source and target concepts are resolved from the code system defining the association. The code system can be overridden by each specific association, relation source (*associationInstance*), or relation target (*associationTarget*).

Categories: VKC Contents | Documentation | LexEVS

LexEVS 5.x Design and Architecture Guide LexBIG Extensions

LexEVS 5.x Migration EVS API to LexEVS API > LexEVS 5.x Migration OWL loader > LexEVS 5.x Design and Architecture Guide > LexEVS 5.x Design and Architecture Guide LexGrid Model > LexEVS 5.x Design and Architecture Guide LexBIG Extensions

Introduction

This document is a section of the Design and Architecture Guide.

LexBIG extensions

The LexBIG vocabulary model extends the LexGrid model to provide unique constructs or granularity required by caBIG® that are not present in the core model. While many extensions exist, this document will focus on some of direct relevance to the high-level architecture.

Concept resolution

LexBIG allows the service runtime to provide managed resolution of code-based objects that are referenced through LexBIG-specific lists and iterators (mechanism that allow streaming of list content). These lists and iterators are typically returned when requesting sets or graphs of vocabulary terms through the LexBIG API (described in LexBIG APIs). Some model components involved in the resolution process include:

`ConceptReference` – A globally unique reference to a concept code.

`ResolvedConceptReference` - A concept reference for which additional information has been resolved, including description and relationship participation.

`AssociatedConcept` - A concept reference that contains full detail in participation as a source or target of an association, including indications of navigability and qualification.

Note: Formal representation of the LexGrid and LexBIG models are discussed in LexEVS Information Models.

Categories: VKC Contents | Documentation | LexEVS

- Editing Forum Posts
- Contact Us

search

Search

toolbox

- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link
- Print as PDF

vocabkc contents
- Main Page
- What's New
- Forums
- Bugzilla
- Code Repository
- Feedback
- Contact Us

tools
- LexBIG/LexEVS
- LexWiki
- NCI Protégé
- Related Tools and Models

projects
- LexAjax
- LexGrid
- Cancer Data Standards Repository (caDSR)
- Common Terminology Criteria for Adverse Events (CTCAE)
- Open Health Natural Language Processing (OHNLP) Consortium
- Ontology Development and Information Extraction (ODIE)

semantic infrastructure
- SI Main Page
- Initiatives
- Requirements

other resources
- Library of Documents
- Documentation and Training for Tools
- Index of Terminologies
- Standards and Standards Influencing Organizations
- Outreach

LexEVS 5.x Design and Architecture Guide LexEVS Information Models

Contents [hide]

Introduction

This document is a section of the Design and Architecture Guide.

Information models overview

The information below is provided for introductory purposes. A full description of all available model components is also available in the javadoc distributed with the LexEVS installation package (see file breakdown in the Installation Guide). Since the javadoc is automatically generated and synchronized during the build process, it is recommended as the primary reference for use by LexEVS developers.

LexGrid model

The LexGrid model is mastered in XML schema. The LexBIG project currently builds on the 2009 version of the LexGrid schema. A formal representation, showing portions of this structure that are of primary interest to the LexEVS project, is presented below. A complete version of the model is available at http://informatics.mayo.edu?page=lgm .

CodingSchemes

The CodingSchemes branch of the model defines high level containers for concepts and relations. Each CodingScheme represents a unique code system or version in the LexBIG service. Components of interest include:

codingSchemes

A collection of one or more coding schemes.

codingScheme

A resource that makes assertions about a collection of terminological entities.

entities

A set of entity codes and their lexical descriptions

relations

A collection of relations that represent a particular point of view or community.

versions

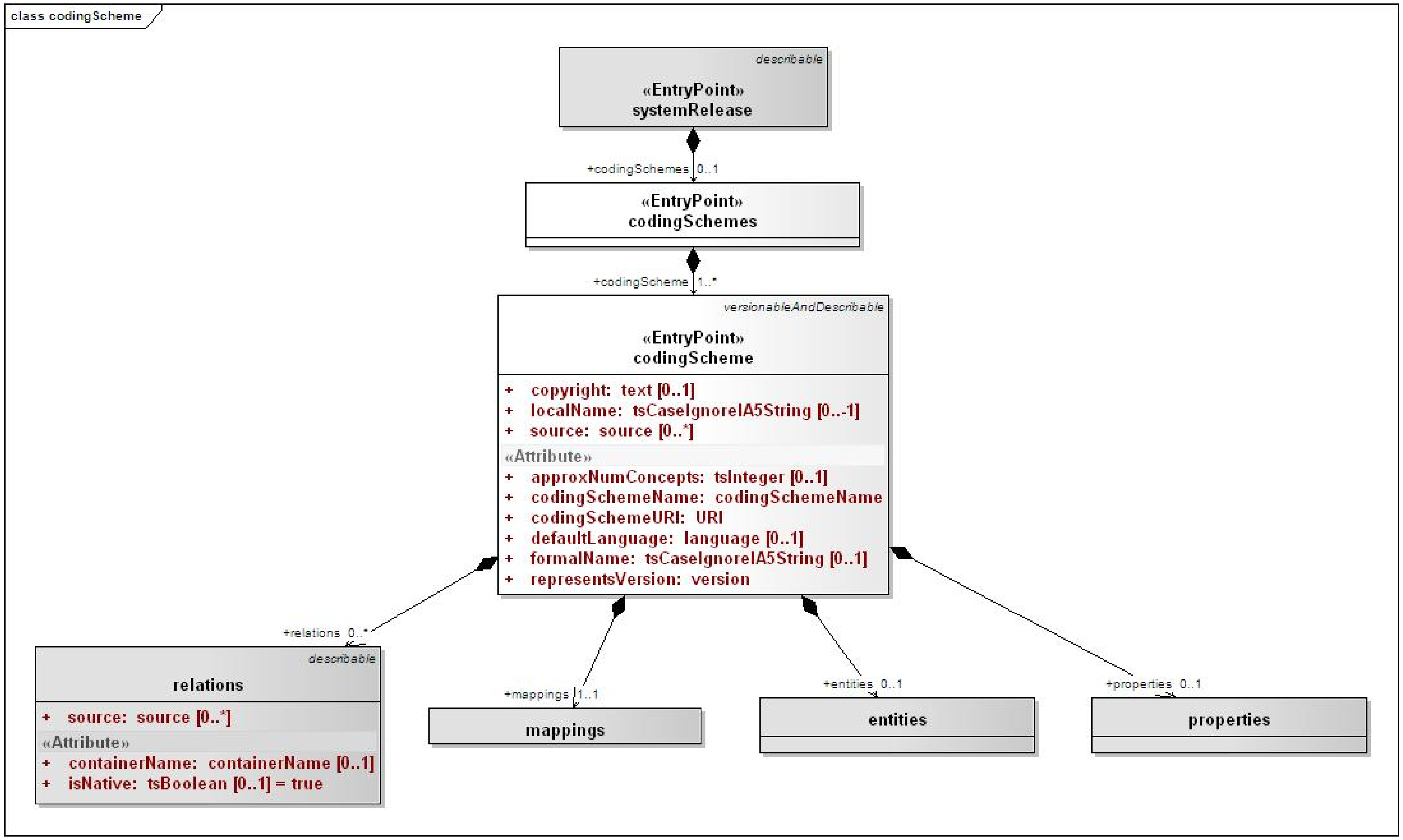
A list of past versions of the coding scheme.

mappings

A list of all of the local identifiers and defining URI's that are used in the associated resource

properties

A collection of properties.


class codingScheme
describable
«EntryPoint» systemRelease
+codingSchemes 0..1
«EntryPoint» codingSchemes
+codingScheme 1..*
versionableAndDescribable
«EntryPoint» codingScheme
+ copyright: text [0..1]
+ localName: tsCaseIgnoreIA5String [0..-1]
+ source: source [0..*]
«Attribute»
+ approxNumConcepts: tsInteger [0..1]
+ codingSchemeName: codingSchemeName
+ codingSchemeURI: URI
+ defaultLanguage: language [0..1]
+ formalName: tsCaseIgnoreIA5String [0..1]
+ representsVersion: version
+relations 0..*
describable
relations
+ source: source [0..*]
«Attribute»
+ containerName: containerName [0..1]
+ isNative: tsBoolean [0..1] = true
+mappings 1..1
mappings
+entities 0..1
entities
+properties 0..1
properties

codingSchemes

Concepts

Each concept represents a unique entity within the code system, which can be further described by properties and related to other concepts through relations.

conceptsAndInstances

codingScheme

A resource that makes assertions about a collection of terminological entities.

entities

A set of entity codes and their lexical descriptions

entity

A set of lexical assertions about the intended meaning of a particular entity code.

concept

An entity that represents a class or category. The entityType for the class concept must be "concept".

instance

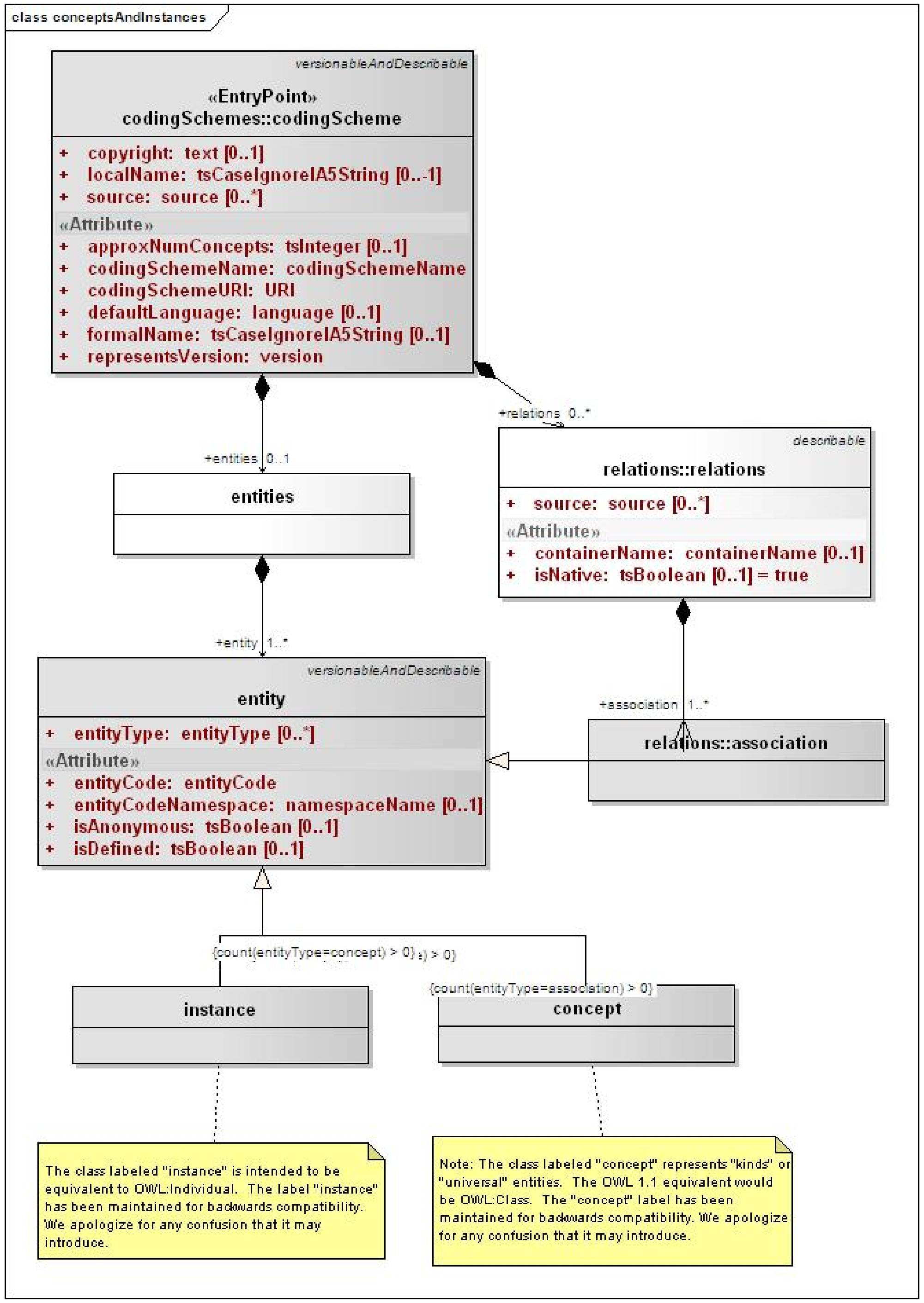
An entity that represents an instance or an individual. The entityType for the class concept must be "instance".

relations

A collection of relations that represent a particular point of view or community.

association

A binary relation from a set of entities to a set of entities and/or data. The entityType for the class concept must be "association".


class conceptsAndInstances
versionableAndDescribable
«EntryPoint»
codingSchemes::codingScheme
+ copyright: text [0..1]
+ localName: tsCaseIgnoreIA5String [0..-1]
+ source: source [0..*]
«Attribute»
+ approxNumConcepts: tsInteger [0..1]
+ codingSchemeName: codingSchemeName
+ codingSchemeURI: URI
+ defaultLanguage: language [0..1]
+ formalName: tsCaseIgnoreIA5String [0..1]
+ representsVersion: version
+relations 0..*
+entities 0..1
entities
describable
relations::relations
+ source: source [0..*]
«Attribute»
+ containerName: containerName [0..1]
+ isNative: tsBoolean [0..1] = true
+entity 1..*
versionableAndDescribable
entity
+ entityType: entityType [0..*]
«Attribute»
+ entityCode: entityCode
+ entityCodeNamespace: namespaceName [0..1]
+ isAnonymous: tsBoolean [0..1]
+ isDefined: tsBoolean [0..1]
+association 1..*
relations::association
{count(entityType=concept) > 0}
{count(entityType=association) > 0}
instance
concept
The class labeled "instance" is intended to be equivalent to OWL:Individual. The label "instance" has been maintained for backwards compatibility. We apologize for any confusion that it may introduce.
Note: The class labeled "concept" represents "kinds" or "universal" entities. The OWL 1.1 equivalent would be OWL:Class. The "concept" label has been maintained for backwards compatibility. We apologize for any confusion that it may introduce.

conceptsAndInstances

entities

codingScheme

A resource that makes assertions about a collection of terminological entities.

entities

A set of entity codes and their lexical descriptions

entity

A set of lexical assertions about the intended meaning of a particular entity code.

concept

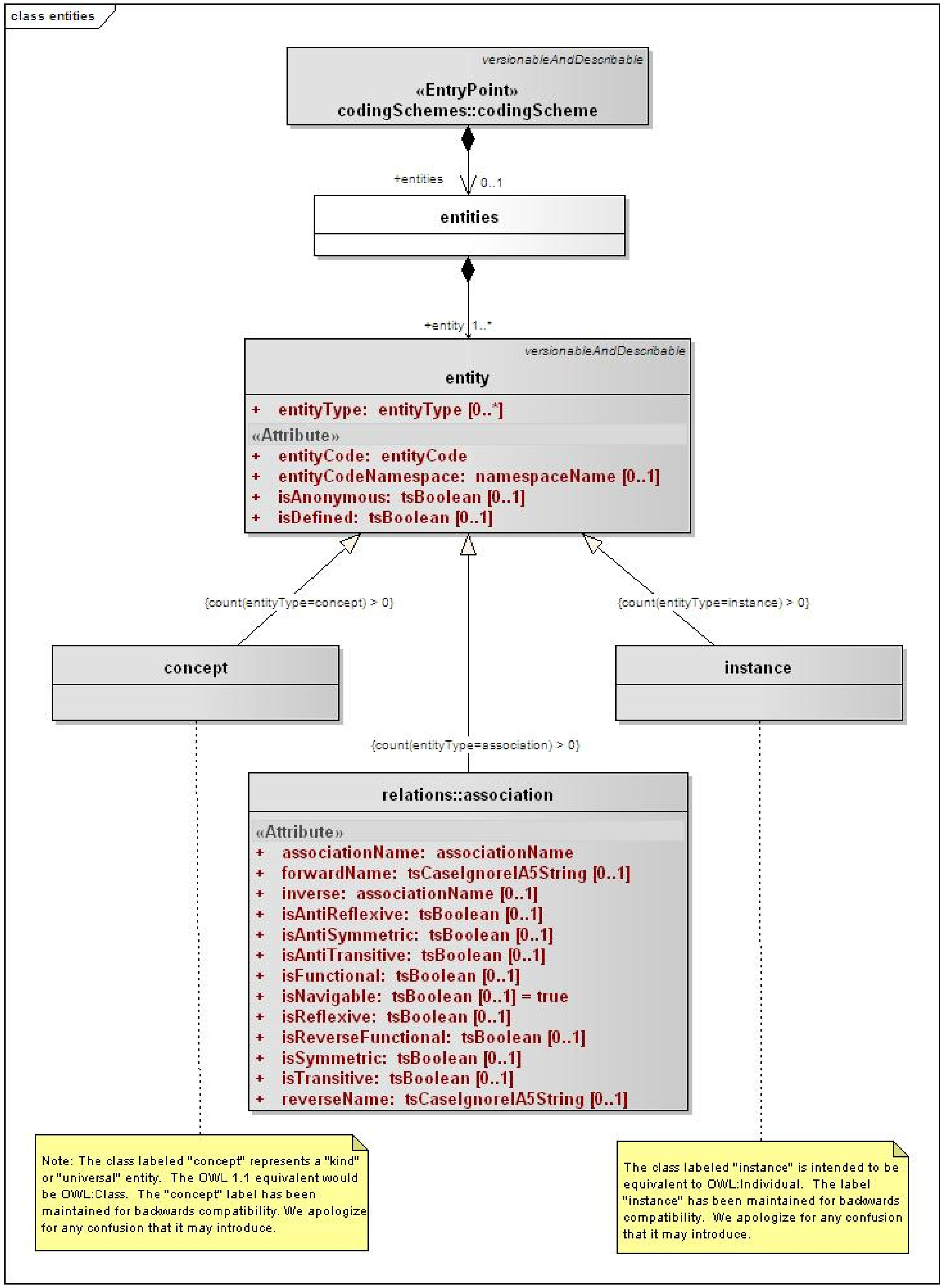
An entity that represents a class or category. The entityType for the class concept must be "concept".

instance

An entity that represents an instance or an individual. The entityType for the class concept must be "instance".

association

A binary relation from a set of entities to a set of entities and/or data. The entityType for the class concept must be "association".


class entities
versionableAndDescribable
«EntryPoint»
codingSchemes::codingScheme
+entities
0..1
entities
+entity
1..*
versionableAndDescribable
entity
+ entityType: entityType [0..*]
«Attribute»
+ entityCode: entityCode
+ entityCodeNamespace: namespaceName [0..1]
+ isAnonymous: tsBoolean [0..1]
+ isDefined: tsBoolean [0..1]
{count(entityType=concept) > 0}
{count(entityType=instance) > 0}
concept
instance
{count(entityType=association) > 0}
relations::association
«Attribute»
+ associationName: associationName
+ forwardName: tsCaseIgnoreIA5String [0..1]
+ inverse: associationName [0..1]
+ isAntiReflexive: tsBoolean [0..1]
+ isAntiSymmetric: tsBoolean [0..1]
+ isAntiTransitive: tsBoolean [0..1]
+ isFunctional: tsBoolean [0..1]
+ isNavigable: tsBoolean [0..1] = true
+ isReflexive: tsBoolean [0..1]
+ isReverseFunctional: tsBoolean [0..1]
+ isSymmetric: tsBoolean [0..1]
+ isTransitive: tsBoolean [0..1]
+ reverseName: tsCaseIgnoreIA5String [0..1]
Note: The class labeled "concept" represents a "kind" or "universal" entity. The OWL 1.1 equivalent would be OWL:Class. The "concept" label has been maintained for backwards compatibility. We apologize for any confusion that it may introduce.
The class labeled "instance" is intended to be equivalent to OWL:Individual. The label "instance" has been maintained for backwards compatibility. We apologize for any confusion that it may introduce.

entities

entity

entity

A set of lexical assertions about the intended meaning of a particular entity code.

comment

A property that is used as an annotation or other note about the state or usage of the entity. The propertyType of comment must be "comment"

definition

A property that defines the entity in a particular langage or context.. The propertyType of definition must be "definition"

presentation

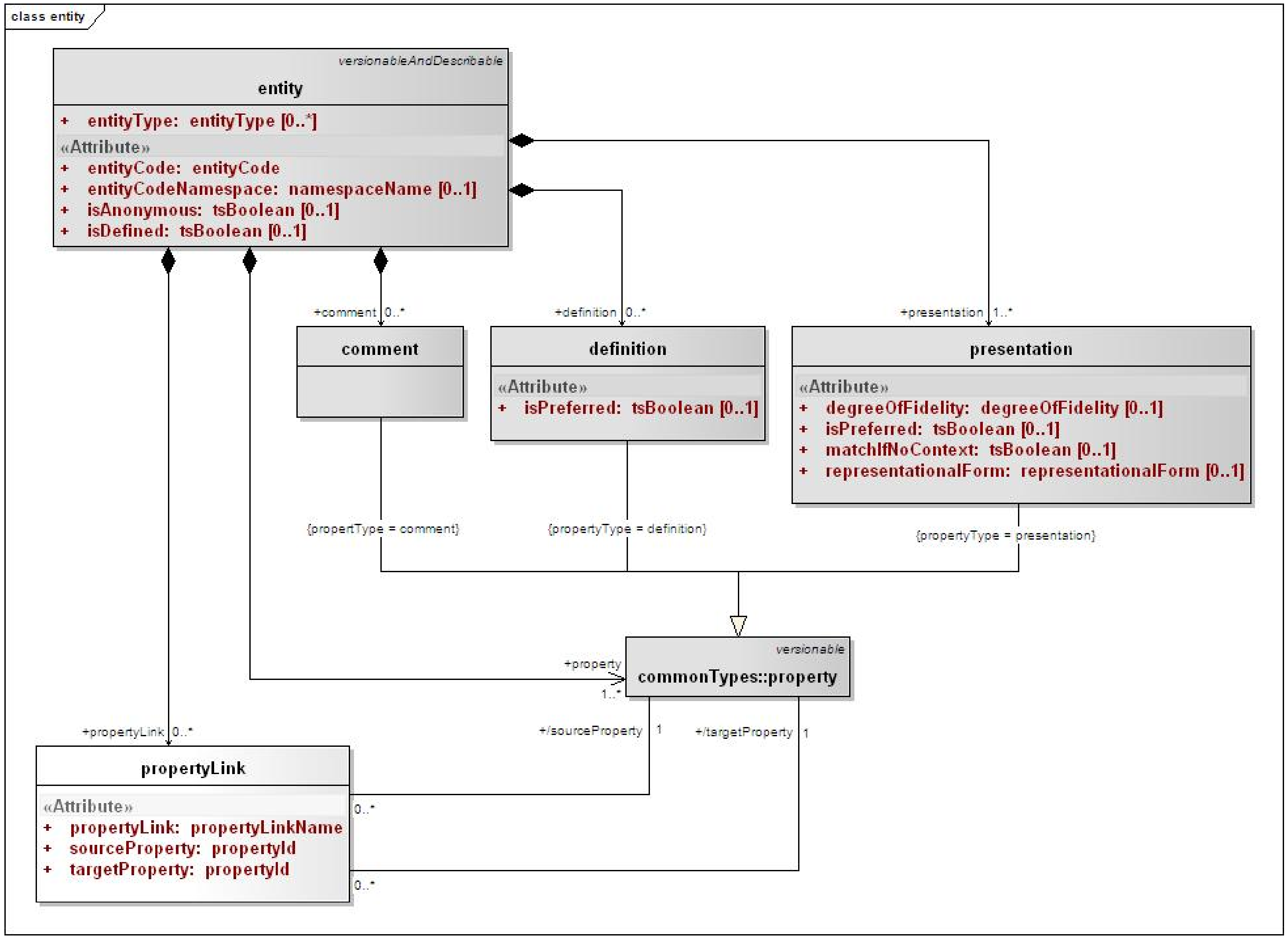
A property ths represents or designates the meaning of the entityCode. The propertyType of presentation must be "presentation"

property

A description, definition, annotation or other attribute that serves to further define or identify an resource.

propertyLink

A link between two properties for an entity.. Examples include acronymFor, abbreviationOf, spellingVariantOf, etc. Must be in supportedPropertyLink.


class entity
versionableAndDescribable
entity
+ entityType: entityType [0..*]
«Attribute»
+ entityCode: entityCode
+ entityCodeNamespace: namespaceName [0..1]
+ isAnonymous: tsBoolean [0..1]
+ isDefined: tsBoolean [0..1]
+comment 0..*
+definition 0..*
+presentation 1..*
comment
definition
«Attribute»
+ isPreferred: tsBoolean [0..1]
presentation
«Attribute»
+ degreeOfFidelity: degreeOfFidelity [0..1]
+ isPreferred: tsBoolean [0..1]
+ matchIfNoContext: tsBoolean [0..1]
+ representationalForm: representationalForm [0..1]
{propertyType = comment}
{propertyType = definition}
{propertyType = presentation}
versionable
commonTypes::property
+property
1..*
+propertyLink 0..*
+/sourceProperty 1
+/targetProperty 1
propertyLink
«Attribute»
+ propertyLink: propertyLinkName
+ sourceProperty: propertyId
+ targetProperty: propertyId
0..*
0..*

entity

Relations

Relations are used to define and qualify associations between concepts.

association

codingScheme

A resource that makes assertions about a collection of terminological entities.

relations

A collection of relations that represent a particular point of view or community.

entity

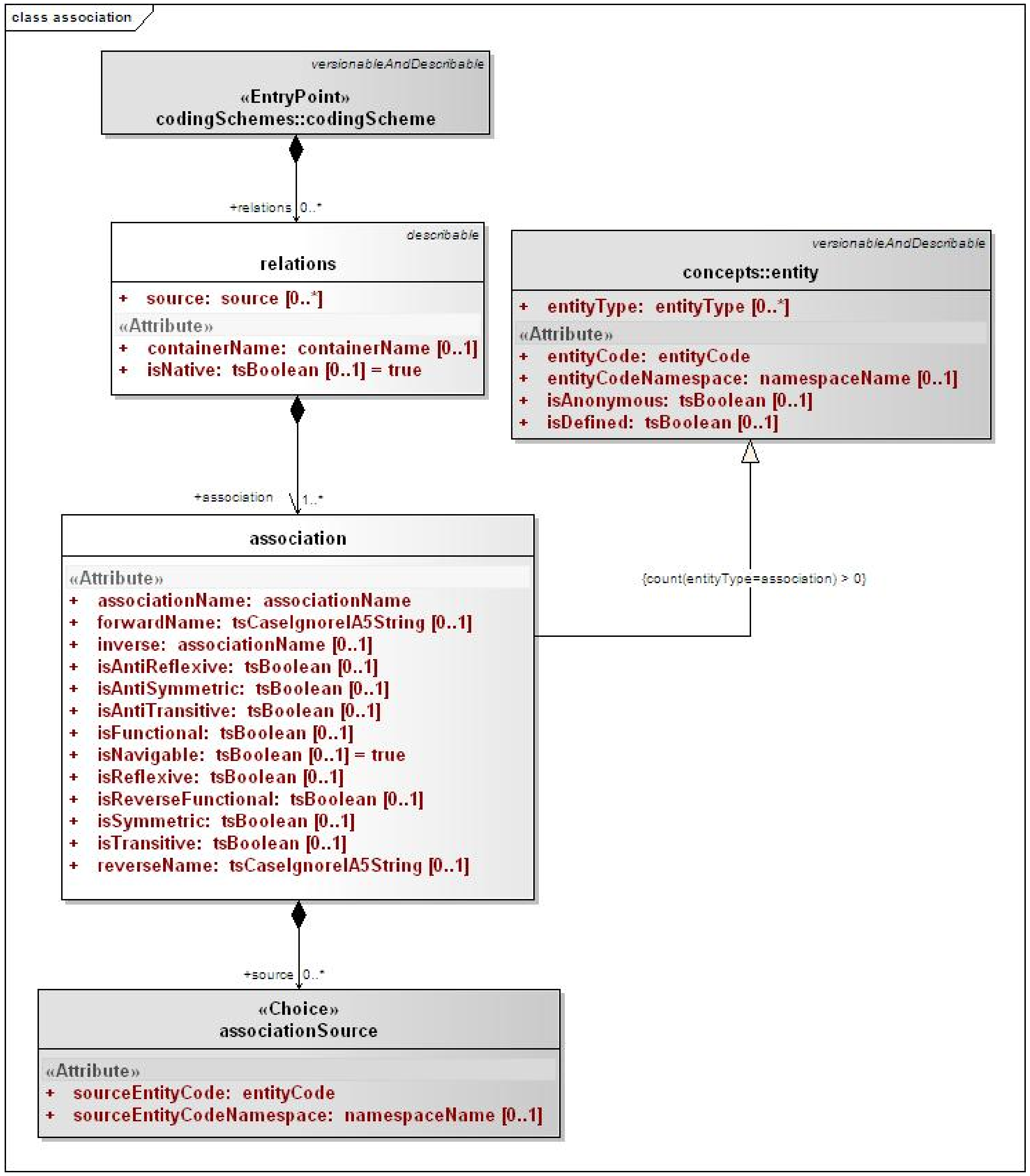
A set of lexical assertions about the intended meaning of a particular entity code.

association

A binary relation from a set of entities to a set of entities and/or data. The entityType for the class concept must be "association".

associationSource

An entity that occurs in one or more instances of a relation on the "from" (or left hand) side of a particular relation.


class association
versionableAndDescribable
«EntryPoint»
codingSchemes::codingScheme
+relations 0..*
describable
relations
+ source: source [0..*]
«Attribute»
+ containerName: containerName [0..1]
+ isNative: tsBoolean [0..1] = true
versionableAndDescribable
concepts::entity
+ entityType: entityType [0..*]
«Attribute»
+ entityCode: entityCode
+ entityCodeNamespace: namespaceName [0..1]
+ isAnonymous: tsBoolean [0..1]
+ isDefined: tsBoolean [0..1]
+association 1..*
{count(entityType=association) > 0}
association
«Attribute»
+ associationName: associationName
+ forwardName: tsCaseIgnoreIA5String [0..1]
+ inverse: associationName [0..1]
+ isAntiReflexive: tsBoolean [0..1]
+ isAntiSymmetric: tsBoolean [0..1]
+ isAntiTransitive: tsBoolean [0..1]
+ isFunctional: tsBoolean [0..1]
+ isNavigable: tsBoolean [0..1] = true
+ isReflexive: tsBoolean [0..1]
+ isReverseFunctional: tsBoolean [0..1]
+ isSymmetric: tsBoolean [0..1]
+ isTransitive: tsBoolean [0..1]
+ reverseName: tsCaseIgnoreIA5String [0..1]
+source 0..*
«Choice»
associationSource
«Attribute»
+ sourceEntityCode: entityCode
+ sourceEntityCodeNamespace: namespaceName [0..1]

association

associationInstance

association

A binary relation from a set of entities to a set of entities and/or data. The entityType for the class concept must be "association".

associationSource

An entity that occurs in one or more instances of a relation on the "from" (or left hand) side of a particular relation.

associationTarget

An entity on the "to" (or right hand) side of a relation.

associationData

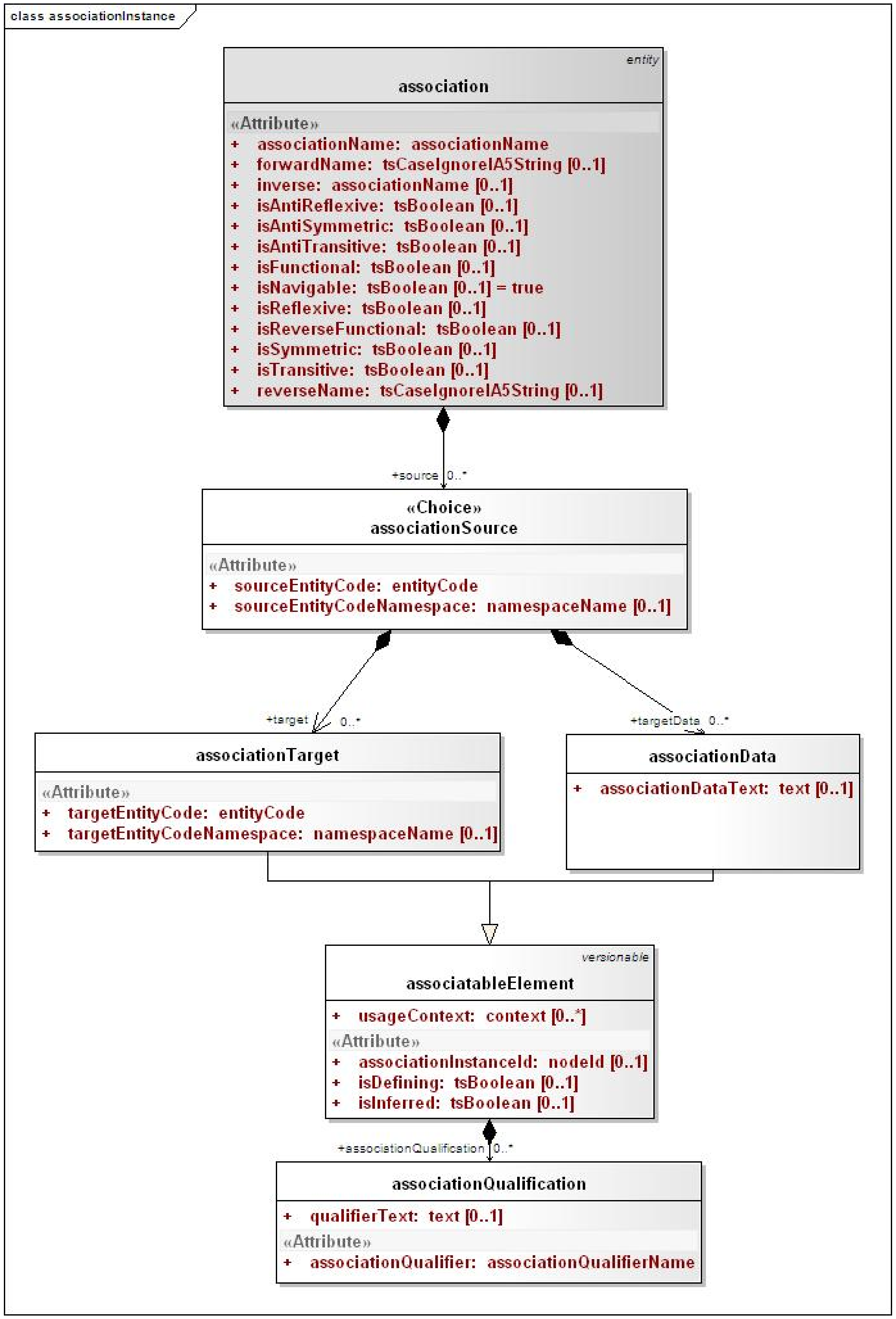
An instance of a target or RHS data value of an association.

associatableElement

Information common to both the entity and data form of the "to" (or right hand) side of an association.

associationQualification

A modifier that further qualifies the particular association instance.


class associationInstance
entity
association
«Attribute»
+ associationName: associationName
+ forwardName: tsCaseIgnoreIA5String [0..1]
+ inverse: associationName [0..1]
+ isAntiReflexive: tsBoolean [0..1]
+ isAntiSymmetric: tsBoolean [0..1]
+ isAntiTransitive: tsBoolean [0..1]
+ isFunctional: tsBoolean [0..1]
+ isNavigable: tsBoolean [0..1] = true
+ isReflexive: tsBoolean [0..1]
+ isReverseFunctional: tsBoolean [0..1]
+ isSymmetric: tsBoolean [0..1]
+ isTransitive: tsBoolean [0..1]
+ reverseName: tsCaseIgnoreIA5String [0..1]
+source 0..*
«Choice»
associationSource
«Attribute»
+ sourceEntityCode: entityCode
+ sourceEntityCodeNamespace: namespaceName [0..1]
+target 0..*
+targetData 0..*
associationTarget
«Attribute»
+ targetEntityCode: entityCode
+ targetEntityCodeNamespace: namespaceName [0..1]
associationData
+ associationDataText: text [0..1]
versionable
associatableElement
+ usageContext: context [0..*]
«Attribute»
+ associationInstanceId: nodeId [0..1]
+ isDefining: tsBoolean [0..1]
+ isInferred: tsBoolean [0..1]
+associationQualification 0..*
associationQualification
+ qualifierText: text [0..1]
«Attribute»
+ associationQualifier: associationQualifierName

associationInstance

Naming

These elements are primarily used to define metadata for a coding scheme, mapping locally used names to global references.

URIMap

A local identifier that is used in a specific context (e.g. language, property name, data type, etc) and an optional URI that can be used to find the exact definition and meaning of the local id. Note: the string portion of this entry can be used to provide additional documentation or information, especially when a URI is not supplied.

supportedAssociation

An associationName and the URI of the defining resource.

supportedAssociationQualifier

An associationQualifier and the URI of the defining resource

supportedCodingScheme

A codingSchemeName and the URI of the defining resource

supportedStatus

An entryStatus and the URI of the defining resource

supportedEntityType

An entityType and the URI of the defining resource

supportedContext

A context and the URI of the defining resource

supportedContainerName

A containerName and the URI of the defining resource

supportedDegreeOfFidelity

A degreeOfFidelity and the URI of the defining resource

supportedLanguage

A language and the URI of the defining resource

supportedProperty

A propertyName and the URI of the defining resource

supportedSortOrder

The local identifier and the URI of the defining resource

supportedHierarchy

A list of associations that can be browsed hierarchically.

supportedNamespace

A namespaceName and the corresponding URI

supportedPropertyType

A propertyType and the URI of the defining resource

supportedPropertyQualifier

A propertyQualifierName the URI of the defining resource

supportedPropertyQualifierType

A propertyQualifierType the URI of the defining resource

supportedPropertyLink

A propertyLinkName and ththe URI of the defining resource

supportedRepresentationalForm

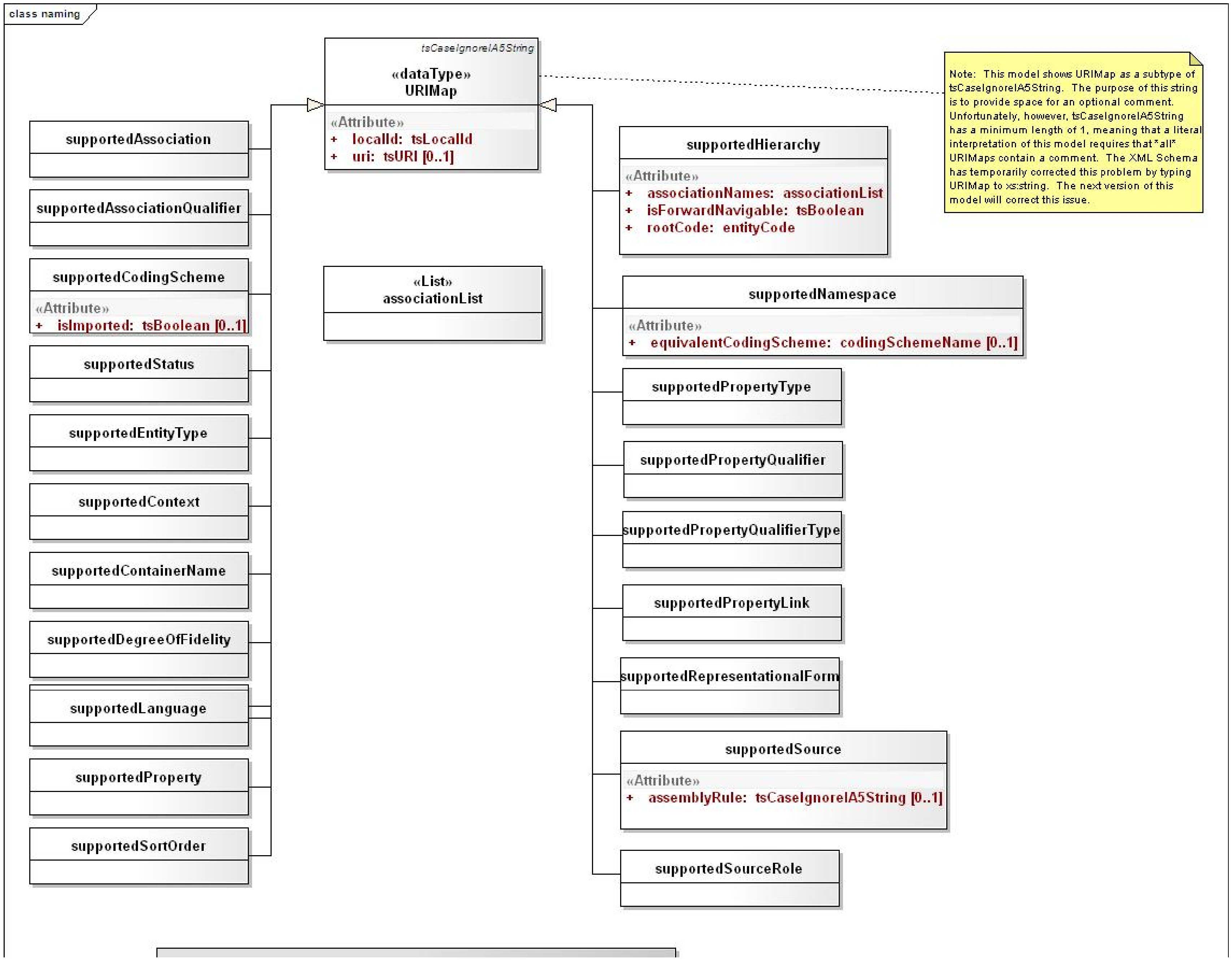
A representationalForm and the URI of the defining resource

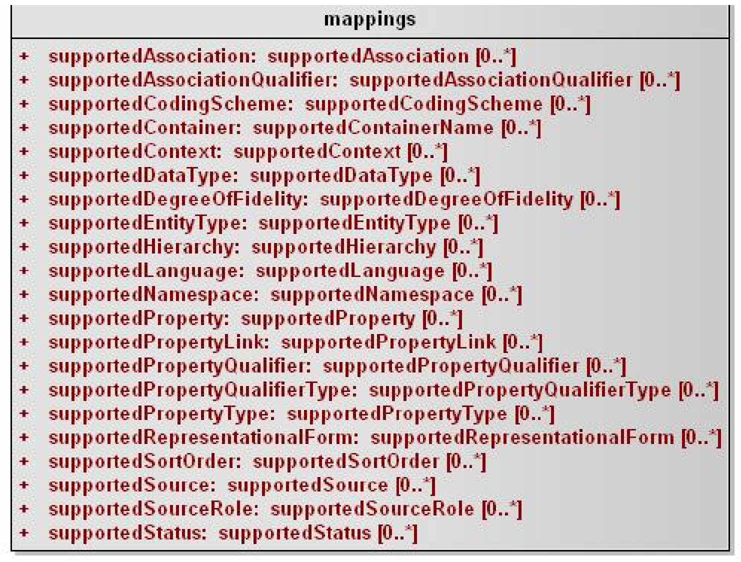
supportedSource

A source and the URI of the defining resource. Source references can also carry an additional compositional rule section that describes how to combine a subpart such as a page number, section name, etc. with the core URI in order to form a meaningful URL. An optional role can also be specified.

supportedSourceRole

A source role and athe URI of the defining resource


class naming
tsCaseIgnoreIA5String
«dataType» URIMap
«Attribute»
+ localId: tsLocalId
+ uri: tsURI [0..1]
«List» associationList
Note: This model shows URIMap as a subtype of tsCaseIgnoreIA5String. The purpose of this string is to provide space for an optional comment. Unfortunately, however, tsCaseIgnoreIA5String has a minimum length of 1, meaning that a literal interpretation of this model requires that *all* URIMaps contain a comment. The XML Schema has temporarily corrected this problem by typing URIMap to xs:string. The next version of this model will correct this issue.
supportedAssociation
supportedAssociationQualifier
supportedCodingScheme
«Attribute»
+ isImported: tsBoolean [0..1]
supportedStatus
supportedEntityType
supportedContext
supportedContainerName
supportedDegreeOfFidelity
supportedLanguage
supportedProperty
supportedSortOrder
supportedHierarchy
«Attribute»
+ associationNames: associationList
+ isForwardNavigable: tsBoolean
+ rootCode: entityCode
supportedNamespace
«Attribute»
+ equivalentCodingScheme: codingSchemeName [0..1]
supportedPropertyType
supportedPropertyQualifier
supportedPropertyQualifierType
supportedPropertyLink
supportedRepresentationalForm
supportedSource
«Attribute»
+ assemblyRule: tsCaseIgnoreIA5String [0..1]
supportedSourceRole


mappings
+ supportedAssociation: supportedAssociation [0..*]
+ supportedAssociationQualifier: supportedAssociationQualifier [0..*]
+ supportedCodingScheme: supportedCodingScheme [0..*]
+ supportedContainer: supportedContainerName [0..*]
+ supportedContext: supportedContext [0..*]
+ supportedDataType: supportedDataType [0..*]
+ supportedDegreeOfFidelity: supportedDegreeOfFidelity [0..*]
+ supportedEntityType: supportedEntityType [0..*]
+ supportedHierarchy: supportedHierarchy [0..*]
+ supportedLanguage: supportedLanguage [0..*]
+ supportedNamespace: supportedNamespace [0..*]
+ supportedProperty: supportedProperty [0..*]
+ supportedPropertyLink: supportedPropertyLink [0..*]
+ supportedPropertyQualifier: supportedPropertyQualifier [0..*]
+ supportedPropertyQualifierType: supportedPropertyQualifierType [0..*]
+ supportedPropertyType: supportedPropertyType [0..*]
+ supportedRepresentationalForm: supportedRepresentationalForm [0..*]
+ supportedSortOrder: supportedSortOrder [0..*]
+ supportedSource: supportedSource [0..*]
+ supportedSourceRole: supportedSourceRole [0..*]
+ supportedStatus: supportedStatus [0..*]

naming

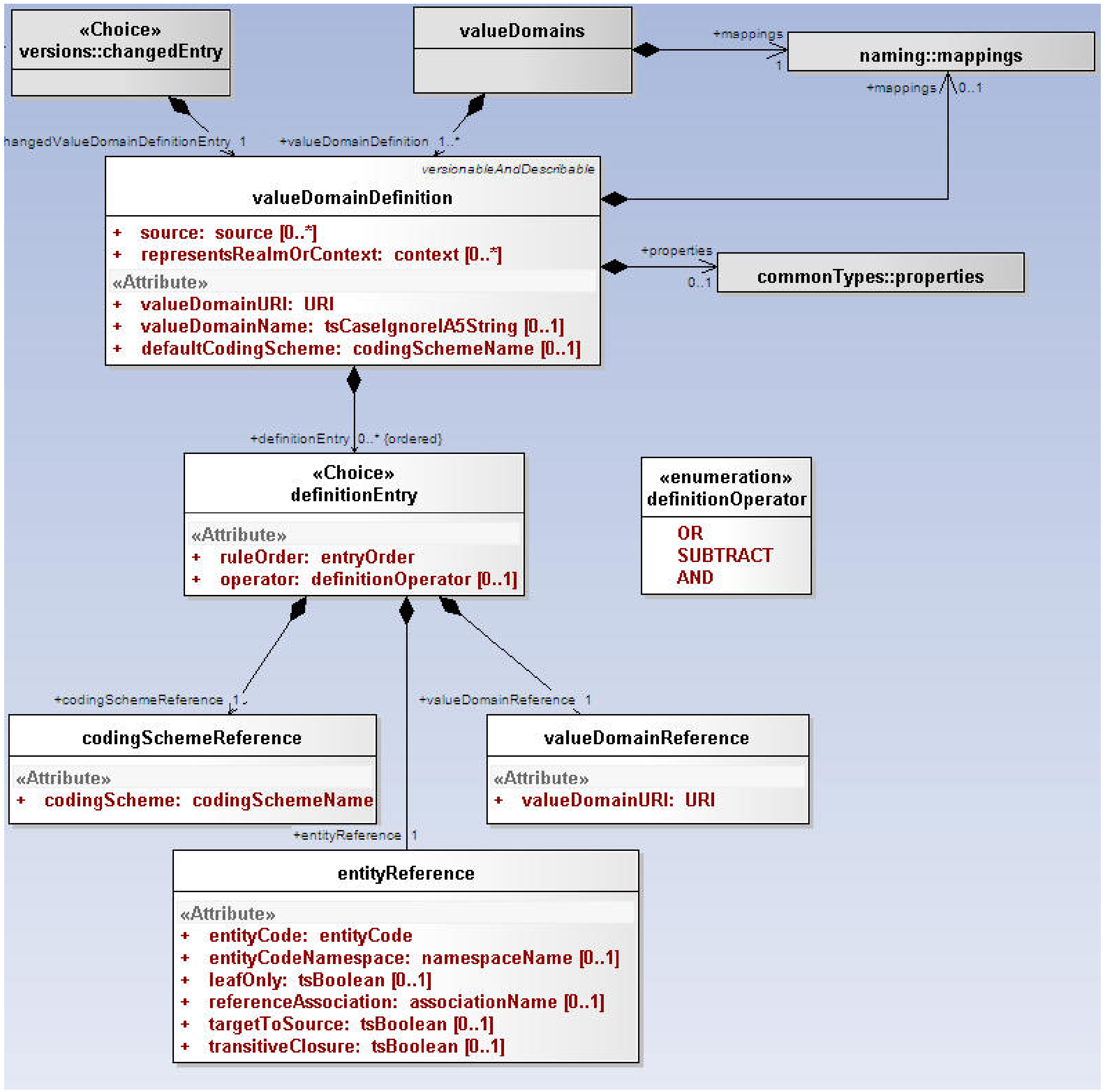
Value Domain Definition

The Value Domain Definition branch of the LexGrid model defines the contents of a value domain.

Value Domain can be defined in following forms:

- Code System : *All* concept codes in the referencing code system.
- Value Domain : *All* concept codes definied in the referencing Value Domain Definition.
- Code System + Concept Code : Individual codes.
- Code System + Concept Code + relationship + additional rules(leafOnly, targetToSource, transitiveClosure..) : Only the concept codes that matches all the definited conditions(rules).
- Combination of any of the above with or/and/difference operators

Here is a UML representation of Value Domain Definition in LexGrid 200901 model:


«Choice»
versions::changedEntry
valueDomains
+mappings
1
naming::mappings
+mappings
0..1
hangedValueDomainDefinitionEntry 1
+valueDomainDefinition 1..*
versionableAndDescribable
valueDomainDefinition
+ source: source [0..*]
+ representsRealmOrContext: context [0..*]
«Attribute»
+ valueDomainURI: URI
+ valueDomainName: tsCaseIgnoreIA5String [0..1]
+ defaultCodingScheme: codingSchemeName [0..1]
+properties
0..1
commonTypes::properties
+definitionEntry 0..* {ordered}
«Choice»
definitionEntry
«Attribute»
+ ruleOrder: entryOrder
+ operator: definitionOperator [0..1]
«enumeration»
definitionOperator
OR
SUBTRACT
AND
+codingSchemeReference 1
+valueDomainReference 1
codingSchemeReference
«Attribute»
+ codingScheme: codingSchemeName
valueDomainReference
«Attribute»
+ valueDomainURI: URI
+entityReference 1
entityReference
«Attribute»
+ entityCode: entityCode
+ entityCodeNamespace: namespaceName [0..1]
+ leafOnly: tsBoolean [0..1]
+ referenceAssociation: associationName [0..1]
+ targetToSource: tsBoolean [0..1]
+ transitiveClosure: tsBoolean [0..1]

Model components

valueDomains

A collection of value domain definitions.

mappings

A list of all of the local identifiers and defining URI's that are used in the associated value domains.

properties

A collection of value domain properties.

changedEntry

A top level versionable entry.

valueDomainDefinition

A definition of a given value domain. A value domain can be a simple description with no associated value domain entries, or it can consist of one or more definitionEntries that resolve to an enumerated list of entityCodes when applied to one or more codingScheme versions.

Attributes of Value Domain Definition:

Source: The local identifiers of the source(s) of this property. Must match a local id of a supportedSource in the corresponding mappings section.

representsRealmOrContext: The local identifiers of the context(s) in which this value domain applies. Must match a local id of a supportedContext in the corresponding mappings section.

valueDomainURI: The URI of this value domain.

valueDomainName: The name of this domain, if any.

defaultCodingScheme: Local name of the primary coding scheme from which the domain is drawn. defaultCodingScheme must match a local id of a supportedCodingScheme in the mappings section.

definitionEntry

A reference to an entry code, a coding scheme or another value domain along with the instructions about how the reference is applied. Definition entrys are applied in entryOrder, with each successive entry either adding to or subtracting from the final set of entity codes.

Attributes of Value Domain Definition Entry:

ruleOrder: The unique identifier of the definition entry within the definition as well as the relative order in which this entry should be applied.

operator: How this entry is to be applied to the value domain.

codingSchemeReference

A reference to all of the entity codes in a given coding scheme.

Attributes of Coding Scheme Reference:

codingScheme: The local identifier of the coding scheme from which the entity codes are drawn. codingSchemeName must match a local ID of a supportedCodingScheme in the mappings section.

valueDomainReference

A reference to the set of codes defined in another value domain.

Attributes of Value Domain Reference:

valueDomainURI: The URI of the value domain to which to apply the operator. This value domain may be contained within the local service or may need to be resolved externally.

entityReference

A reference to an entityCode and/or one or more entityCodes that have a relationship to the specified entity code plus the rules(leafOnly, targetToSource..) to be applied.

Attributes of Entity Reference:

entityCode: The entity code being referenced.

entityCodeNamespace: Local identifier of the namespace of the entityCode. entityCodeNamespace must match a local ID of a supportedNamespace in the corresponding mappings section. If omitted, the URI of the defaultCodingScheme will be used as the URI of the entity code.

leafOnly: If true and referenceAssociation is supplied and referenceAssociation is defined as transitive, include all entity codes that are "leaves" in transitive closure of referenceAssociation as applied to entity code. Default=false.

referenceAssociation: The local identifier of an association that appears in the native relations collection in the default coding scheme. This association is used to describe a set of entity codes. If absent, only the entityCode itself is included in this definition.

targetToSource: If true and referenceAssociation is supplied, navigate from entityCode as the association target to the corresponding sources. If transitiveClosure is true and the referenceAssociation is transitive, include all the ancestors in the list rather than just the direct "parents" (sources).

transitiveClosure: If true and referenceAssociation is supplied and referenceAssociation is defined as transitive, include all entity codes that belong to transitive closure of referenceAssociation as applied to entity code. Default=false.

definitionOperator

The description of how a given definition entry is applied.

Attributes of Definition Operator:

OR: Add the set of entityCodes described by the currentEntity to the value domain; logical OR.

SUBTRACT: Subtract (remove) the set of entityCodes described by the currentEntity to the value domain; logical NAND.

AND: Only include the entity codes that are both in the value domain and the definition entry; logical AND.

Value Domain services

This feature is new in LexEVS 5.1. For details on using value domain services, see the Programmer's Guide, Value Domain Services section.

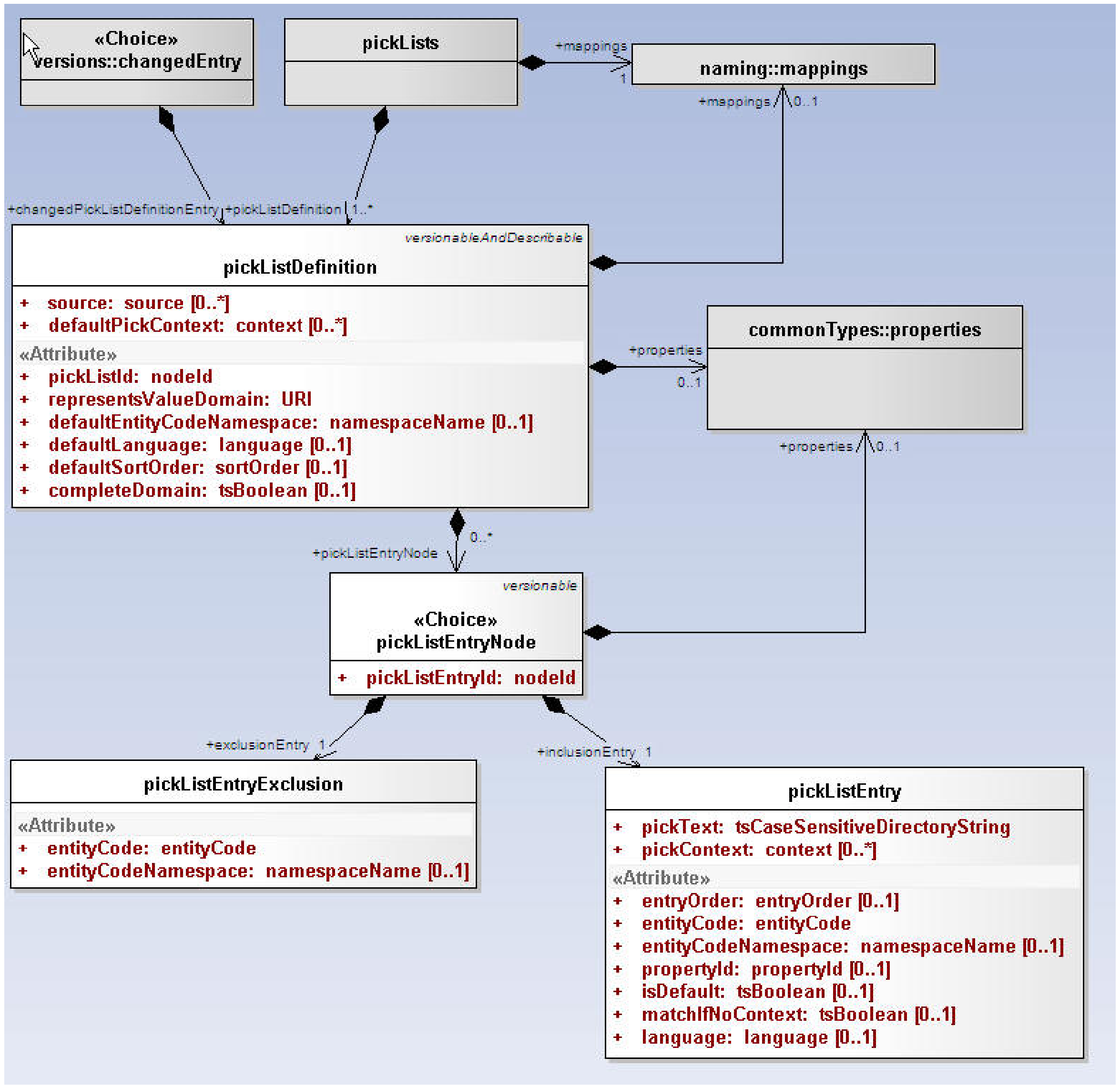
Pick List Definition

Pick List Definition branch of LexGrid model definies the contents of a pick list.

Pick List can be defined in following forms:

- Value Domain: *all* concept codes definied in referenced value domain
- Code System + Concept Code: individual codes (inclusion and exclusion)

Here is a UML representation of Pick List in LexGrid 200901 model:


«Choice» versions::changedEntry
pickLists
naming::mappings
+mappings
1
+mappings
0..1
+changedPickListDefinitionEntry
+pickListDefinition
1..*
versionableAndDescribable
pickListDefinition
+ source: source [0..*]
+ defaultPickContext: context [0..*]
«Attribute»
+ pickListId: nodeId
+ representsValueDomain: URI
+ defaultEntityCodeNamespace: namespaceName [0..1]
+ defaultLanguage: language [0..1]
+ defaultSortOrder: sortOrder [0..1]
+ completeDomain: tsBoolean [0..1]
commonTypes::properties
+properties
0..1
+properties
0..1
+pickListEntryNode
0..*
versionable
«Choice» pickListEntryNode
+ pickListEntryId: nodeId
+exclusionEntry 1
+inclusionEntry 1
pickListEntryExclusion
«Attribute»
+ entityCode: entityCode
+ entityCodeNamespace: namespaceName [0..1]
pickListEntry
+ pickText: tsCaseSensitiveDirectoryString
+ pickContext: context [0..*]
«Attribute»
+ entryOrder: entryOrder [0..1]
+ entityCode: entityCode
+ entityCodeNamespace: namespaceName [0..1]
+ propertyId: propertyId [0..1]
+ isDefault: tsBoolean [0..1]
+ matchIfNoContext: tsBoolean [0..1]
+ language: language [0..1]

Model components

pickLists

A collection of pick list definitions.

mappings

A list of all of the local identifiers and defining URI's that are used in the associated pick list definitions.

properties

A collection of properties.

changedEntry

A top level versionable entry.

pickListDefinition

An ordered list of entity codes and corresponding presentations drawn from a value domain.

Attributes of Pick List Definition:

Source: The local identifiers of the source(s) of this pick list definition. Must match a local ID of a supportedSource in the corresponding mappings section.

pickListId: An identifier that uniquely names this list within the context of the collection.

representsValueDomain: The URI of the value domain definition that is represented by this pick list.

defaultEntityCodeNamespace: Local name of the namespace to which the entry codes in this list belong. defaultEntityCodeNamespace must match a local ID of a supportedNamespace in the mappings section.

defaultLanguage: The local identifier of the language that is used to generate the text of this pick list if not otherwise specified. Note that this language does NOT necessarily have any coorelation with the language of a pickListEntry itself or the language of the target user. defaultLanguage must match a local ID of a supportedLanguage in the mappings section.

defaultSortOrder: The local identifier of a sort order that is used as the default in the definition of the pick list.

defaultPickContext: The local identifiers of the context used in the definition of the pick list.

completeDomain: True means that this pick list should represent all of the entries in the domain. Any active entity codes that are not in the specific pick list entries are added to the end, using the designations identified by the defaultLanguage, defaultSortOrder, and defaultPickContext; default=false.

pickListEntryNode

An inclusion (pickListEntry) or exclusion (pickListEntryExclusion) in a pick list definition

Attributes of Pick List Entry Node:

pickListEntryId: Unique identifier of this node within the list.

pickListEntry

An entity code and corresponding textual representation.

Attributes of Pick List Entry:

pickText: The text that represents this node in the pick list. Some business rules may require that this string match a presentation associated with the entityCode.

pickContext: The local identifiers of the context(s) in which this entry applies. pickContext must match a local ID of a supportedContext in the mappings section.

entryOrder: Relative order of this entry in the list. pickListEntries without a supplied order follow the all entries with an order, and the order is not defined.

entityCode: Entity code associated with this entry.

entityCodeNamespace: Local identifier of the namespace of the entity code if different than the pickListDefinition defaultEntityCodeNamespace. entityCodeNamespace must match a local ID of a supportedNamespace in the mappings section.

propertyId: The property identifier associated with the entityCode and entityCodeNamespace from which the pickText was derived. If absent, the pick text can be anything. Some terminologies may have business rules requiring this attribute to be present.

isDefault: True means that this is the default entry for the supplied language and context.

matchIfNoContext: True means that this entry can be used if no contexts are supplied, even though pickContext is present.

Language: The local name of the language to be used when the application/user supplies a selection language matches. If absent, this matches all languages. language must match a local ID of a supportedLanguage in the mappings section.

pickListEntryExclusion

An entity code that is explicitly excluded from a pick list.

Attributes of Pick List Entry Exclusion:

entityCode: Entity code associated with this entry.

entityCodeNamespace: Local identifier of the namespace of the entity code if different than the pickListDefinition defaultEntityCodeNamespace. entityCodeNamespace must match a local ID of a supportedNamespace in the mappings section.

Pick List services

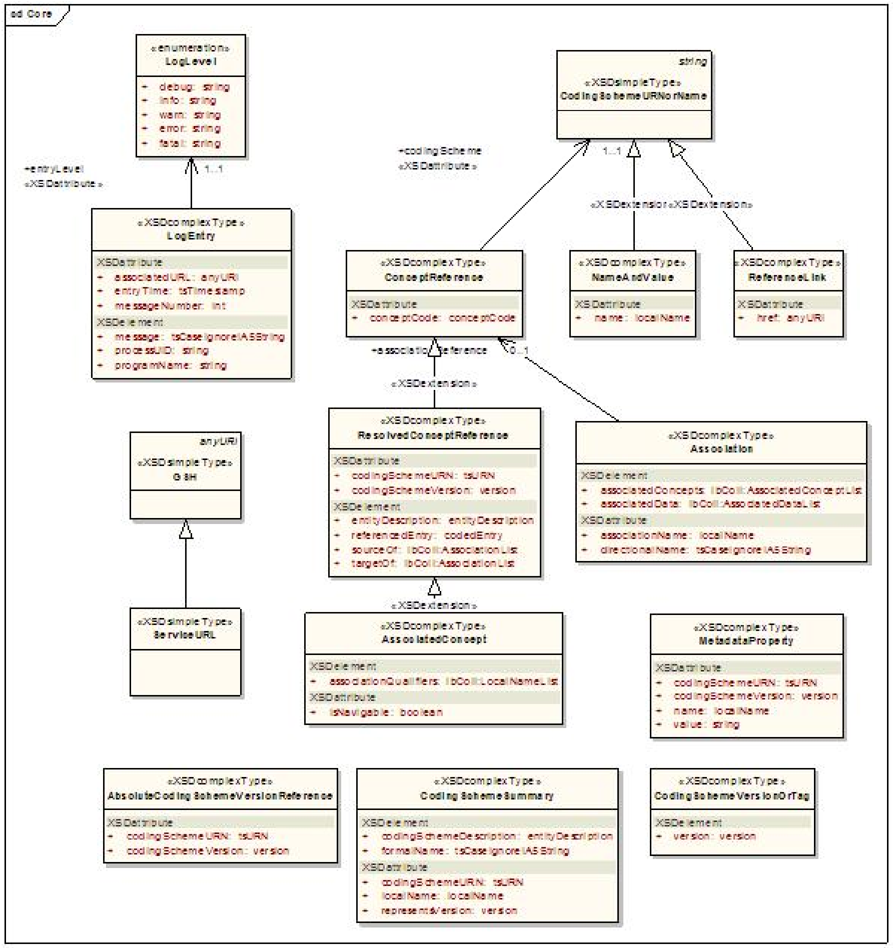
This feature is new in LexEVS 5.1. For details on using pick list services, see the Programmer's Guide, Pick List Services section.

LexBIG model

The following extensions to the LexGrid model were introduced in support of caBIG® requirements. As with the LexGrid model, this document provides a summary of the most significant elements for consideration by LexBIG programmers. The complete and current version of the model is available online at http://informatics.mayo.edu?page=lexex .

Core

LexBIG core elements provide enhanced referencing and controlled resolution of LexGrid model objects.



Core components of interest include:

AbsoluteCodingSchemeVersionReference

An absolute reference to a coding scheme. This form of reference is service independent, as it doesn't depend on local coding schemes names or virtual tags.

AssociatedConcept

A concept reference that is the source or target of an association.

Association

The representation of a particular association as it appears in a CodedNode.

CodingSchemeSummary

Abbreviated list of information about a coding scheme.

CodingSchemeURNorName

Either a local name or the URN of a coding scheme. These two are differentiated syntactically - if the entity includes a colon (:) or a hash "#" it is assumed to be a URN. Otherwise it is assumed to be a local name.

CodingSchemeVersionOrTag

A named coding scheme version or a virtual tag (e.g. latest, production, etc). Note that the tagged form of identifier is only applicable in the context of a given service, as one service may identify the scheme as "production" and another as "staging".

ConceptReference

A reference to a coding scheme and a concept code.

LogEntry

A single recorded log entry.

LogLevel

Indicates severity of the log entry.

MetadataProperty

Reference to a property name and value stored in the coding scheme metadata.

NameAndValue

A simple name/value pair.

ReferenceLink

Any reference to another document element. Used by the REST architecture to embed links.

ResolvedConceptReference

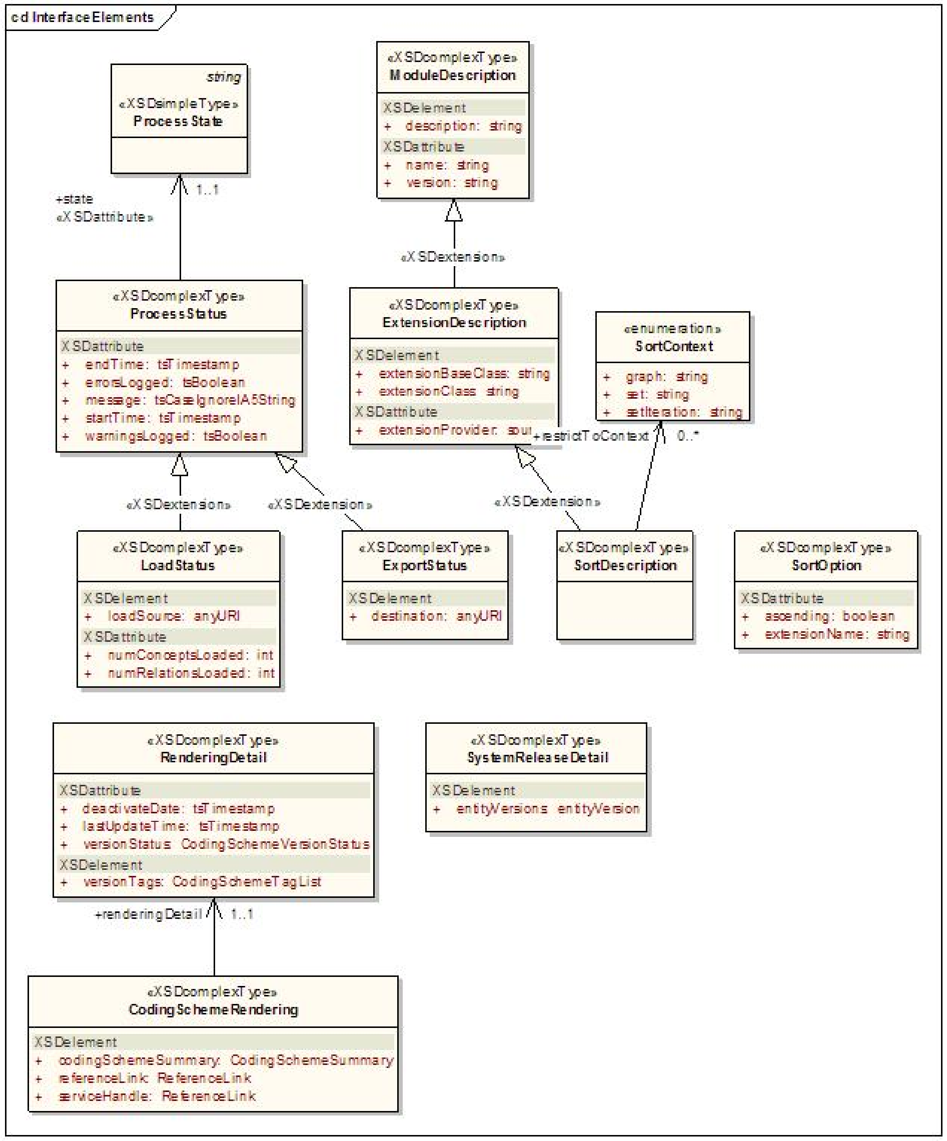
A resolvable concept reference.

ServiceURL

References a service in the Globus environment, this will be a global service handle (GSH).

InterfaceElements

Defines metadata related to model objects required by the runtime.


cd InterfaceElements
string
«XSDsimpleType» ProcessState
+state «XSDattribute» 1..1
«XSDcomplexType» ModuleDescription
XSDelement
+ description: string
XSDattribute
+ name: string
+ version: string
«XSDextension»
«XSDcomplexType» ProcessStatus
XSDattribute
+ endTime: tsTimestamp
+ errorsLogged: tsBoolean
+ message: tsCaseIgnoreIA5String
+ startTime: tsTimestamp
+ warningsLogged: tsBoolean
«XSDcomplexType» ExtensionDescription
XSDelement
+ extensionBaseClass: string
+ extensionClass: string
XSDattribute
+ extensionProvider: sour
«enumeration» SortContext
+ graph: string
+ set: string
+ setIteration: string
+restrictToContext 0..*
«XSDextension»
«XSDextension»
«XSDextension»
«XSDcomplexType» LoadStatus
XSDelement
+ loadSource: anyURI
XSDattribute
+ numConceptsLoaded: int
+ numRelationsLoaded: int
«XSDcomplexType» ExportStatus
XSDelement
+ destination: anyURI
«XSDcomplexType» SortDescription
«XSDcomplexType» SortOption
XSDattribute
+ ascending: boolean
+ extensionName: string
«XSDcomplexType» RenderingDetail
XSDattribute
+ deactivateDate: tsTimestamp
+ lastUpdateTime: tsTimestamp
+ versionStatus: CodingSchemeVersionStatus
XSDelement
+ versionTags: CodingSchemeTagList
«XSDcomplexType» SystemReleaseDetail
XSDelement
+ entityVersions: entityVersion
+renderingDetail 1..1
«XSDcomplexType» CodingSchemeRendering
XSDelement
+ codingSchemeSummary: CodingSchemeSummary
+ referenceLink: ReferenceLink
+ serviceHandle: ReferenceLink

InterfaceElements components of interest include:

CodingSchemeRendering

Information about a coding scheme as it appears in a particular service.

ExportStatus

Reports the state of LexBIG export operations.

ExtensionDescription

Describes an add-on module registered to the LexBIG environment.

LoadStatus

Reports the state of LexBIG load operations.

ModuleDescription

Describes a LexBIG integrated software module.

ProcessState

Enumerates possible status reported for LexBIG runtime operations.

ProcessStatus

Reports the state of LexBIG runtime operations.

RenderingDetail

The details of how a coding scheme is rendered in a given service.

SortContext

Describes a LexBIG sort module.

SortDescription

A description of a LexBIG extension module.

SortOption

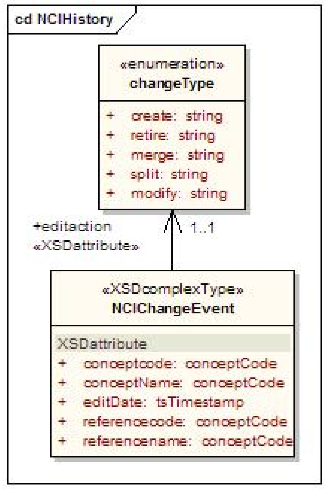
Represents a pairing of sort algorithm and order.

SystemReleaseDetail

The combination of a system release and all of the entityVersions that accompanied that release.

NCIHistory

Maintains a record of modifications made to a code system.



NCIHistory components of interest include:

changeType

Atomic modification actions. Currently populated from a combination of Concordia, SNOMED-CT list and NCI's action list.

NCIChangeEvent

A change event as documented in ftp://ftp1.nci.nih.gov/pub/cacore/EVS/ReadMe_history.txt. Note that date and time of the change event is recorded in the containing version. All change events for the same/date and time a recorded in the same version.

Categories: VKC Contents | Documentation | LexEVS

This page was last modified on 31 January 2010, at 01:33. This page has been accessed 32 times.

LexEVS 5.x Design and Architecture Guide LexEVS Architecture

LexEVS 5.x Design and Architecture Guide > LexEVS 5.x Design and Architecture Guide LexGrid Model > LexEVS 5.x Design and Architecture Guide LexBIG Extensions > LexEVS 5.x Design and Architecture Guide LexEVS Information Models > LexEVS 5.x Design and Architecture Guide LexEVS Architecture

Contents [hide]

Introduction

This document is a section of the Design and Architecture Guide.

LexEVS architecture overview

The LexEVS v5.1 infrastructure exhibits an n-tiered architecture with client interfaces, server components, domain objects, data sources, and back-end systems (illustrated below). This n-tiered architecture divides tasks or requests among different servers and data stores; it isolates the client from the details of where and how data is retrieved from different data stores.

LexEVS also performs common tasks such as logging and provides a level of security for protected content. Clients (browsers, applications) receive information through designated application programming interfaces (APIs). Java applications communicate with back-end objects via domain objects packaged within the client.jar. Non-Java applications can communicate via SOAP (Simple Object Access Protocol) or REST (Representational State Transfer) services.

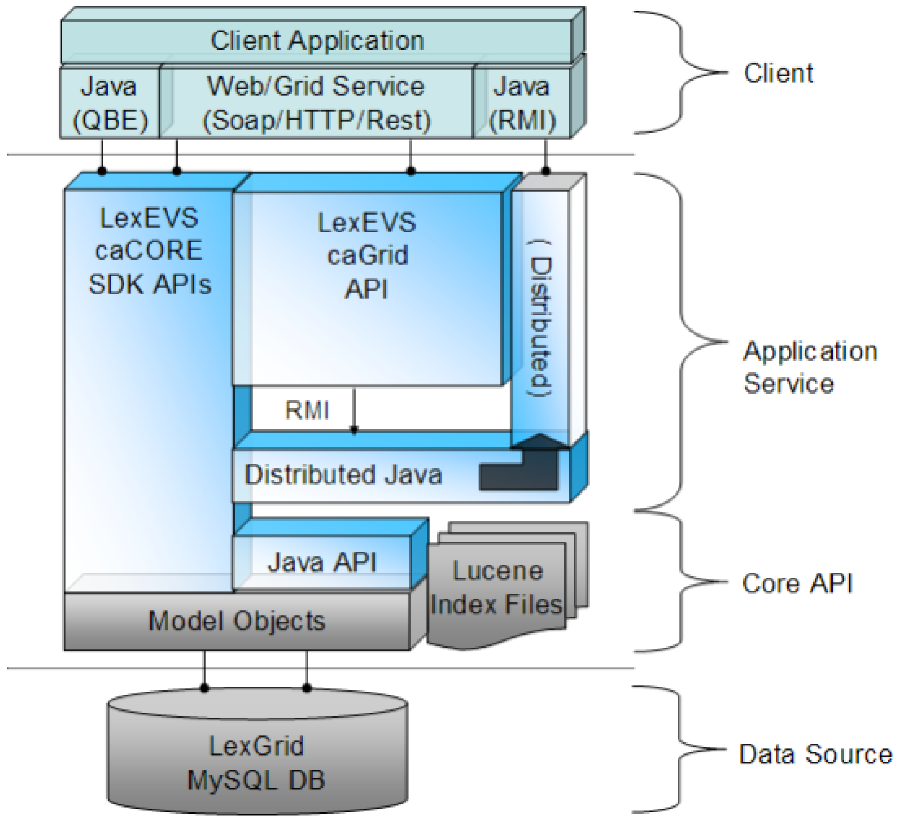
Most of the LexEVS API infrastructure is written in the Java programming language and leverages reusable, third-party components. The service infrastructure is composed of the following layers:

> Application Service layer accepts incoming requests from all public interfaces and translates them, as required, to Java calls in terms of the native LexEVS API. Non-SDK queries are invoked against the Distributed LexEVS API, which handles client authentication and acts as proxy to invoke the equivalent function against the LexEVS core Java API. The caGrid and SDK-generated services are optionally run in an application server separate from the Distributed LexEVS API.
>
> The LexEVS caCORE SDK services work directly against the database, via Hibernate bindings, to resolve stored objects without intermediate translation of calls in terms of the LexEVS API. However, the LexEVS SDK services do still require access to metadata and security information stored by the Distributed and Core LexEVS API environment to resolve the specific database location for requested objects and to verify access to protected resources, respectively.
>
> From the client prospective, the LexEVS services function as "ports" accessible through the caGrid 1.3 service architectural model. LexEVS services follow the caGrid architecture for analytical and data services. See the caGrid 1.3 documentation for architectural details: [1]
>
> Core API layer underpins all LexEVS API requests. Search of pre-populated Lucene index files is used to evaluate query results before incurring cost of database access. Access to the LexGrid database is performed as required to populate returned objects using pooled connections.
>
> Data Source layer is responsible for storage and access to all data required to represent the objects returned through API

vocabkc contents
Main Page
What's New
Forums
Bugzilla
Code Repository
Feedback
Contact Us
tools
LexBIG/LexEVS
LexWiki
NCI Protégé
Related Tools and Models
projects
LexAjax
LexGrid
Cancer Data Standards Repository (caDSR)
Common Terminology Criteria for Adverse Events (CTCAE)
Open Health Natural Language Processing (OHNLP) Consortium
Ontology Development and Information Extraction (ODIE)
semantic infrastructure
SI Main Page
Initiatives
Requirements
other resources
Library of Documents
Documentation and Training for Tools
Index of Terminologies
Standards and Standards Influencing Organizations
Outreach
external links
VCDE Workspace
caBIG® Community Website
caBIG® Support Service Providers
help
Editing Wiki Pages

invocation.

High-level design diagram



Cross-product dependencies

No new dependencies since LexEVS v5.0. See the Core Product Dependency Matrix.

Changes in technology

No changes in the technology stack since LexEVS v5.0.

Assumptions

None.

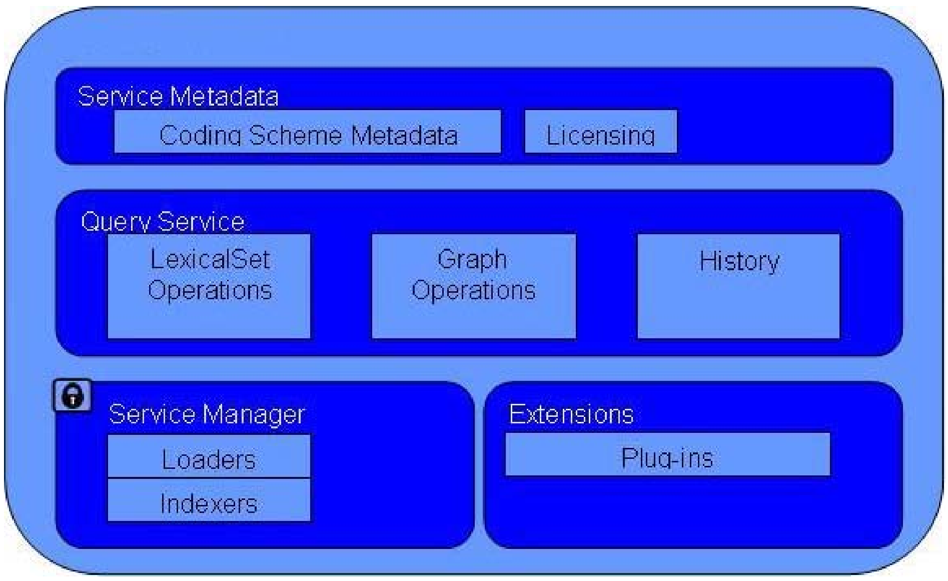
Risks

Due to the additional content to load from RRF files in LexEVS v5.1, there is a risk of increased loading times and increased storage requirements. However, the new loader framework should mitigate the increased loading times: it provides a faster load while increasing the capacity for content to be loaded.

LexBIG

LexBIG Services

This section describes architectural detail for services provided by the LexBIG system. These services are geared toward the administration, management, and serving of vocabularies defined to the LexGrid/LexBIG information model. A system overview is provided, followed by a description of key subsystems and components. Each subsystem is described in terms of its overall structure, formal model, and specification of key public interfaces.

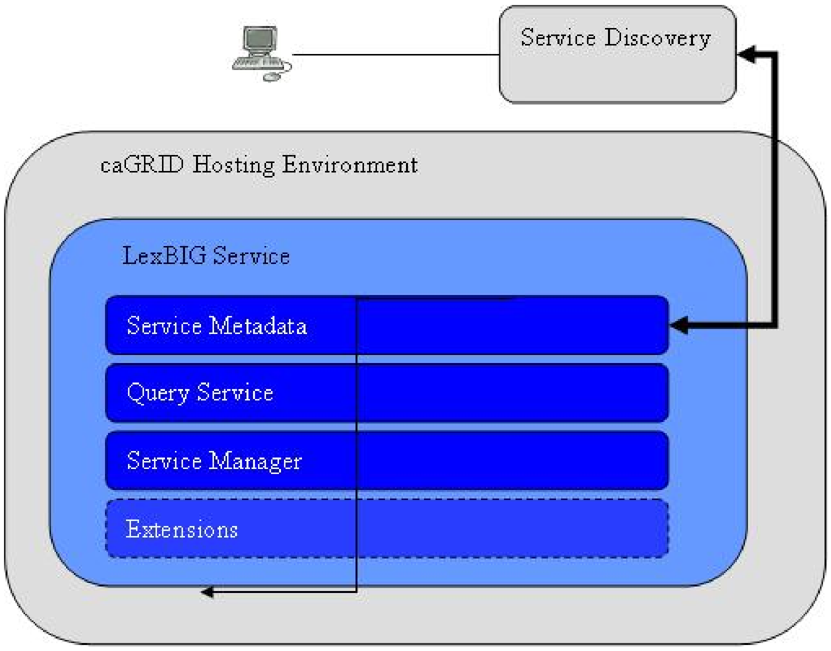


The LexBIG Service is designed to run standalone or as part of a larger network of services. It is comprised of four primary subsystems: Service Management, Service Metadata, Query Operations, and Extensions. The Service Manager provides administration control for loading a vocabulary and activating a service. The Service Metadata provides external clients with information about the vocabulary content (e.g. NCI Thesaurus) and appropriate licensing information. The Query Operations provide numerous functions for querying and traversing vocabulary content. Finally, the Extensions component provides a mechanism to extend the specific service functions, such as Loaders, or re-wrap specific query operations into convenience methods. Primary points of interaction for programming include the following classes:

`LexBIGService` – This interface provides centralized access to all LexBIG services.

`LexBIGServiceManager` – The service manager provides a centralized access point for administrative functions, including write and update access for a service's content. For example, the service manager allows new coding schemes to be validated and loaded, existing coding schemes to be retired and removed, and the status of various coding schemes to be updated and changed.

caGRID Hosting

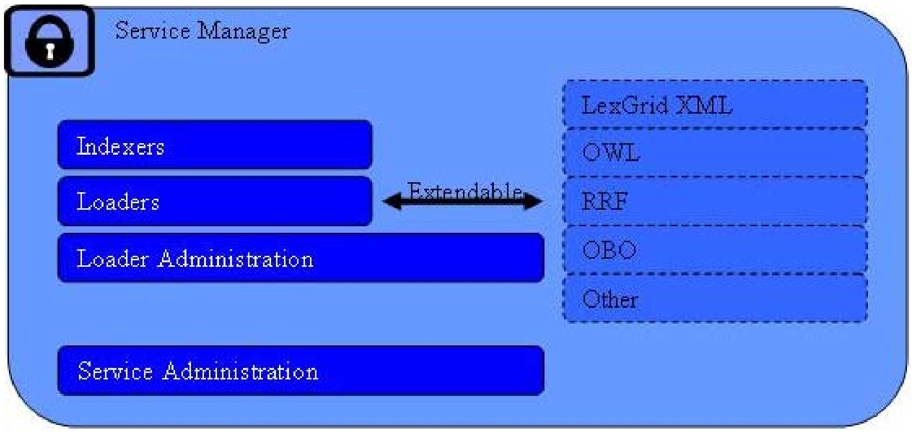


The LexBIG architecture provides the underpinnings LexBIG services to be made accessible through the caGRID environment in the future, where LexBIG services might optionally be deployed in a caGRID Globus container. caGrid provides a Globus service for service registration and discovery. LexBIG services deployed to the grid would be registered in the NCICB registry and be searchable through the NCICB index service.

Specification

Additional specifications related to the registration and discovery of LexBIG services in the caGRID environment will be included later phases of work in concordance with caGRID 1.0. This is will be coordinated with caBIG® Architecture workspace designees.

Service Management Subsystem



This subsystem provides administrative access to functions related to management and publication of LexBIG vocabularies. These functions are generally considered to be reserved for LexBIG administrators, with detailed instructions on how to secure and carry out related tasks described by the *LexBIG Administrator's Guide*.

This subsystem is further broken down into the following components:

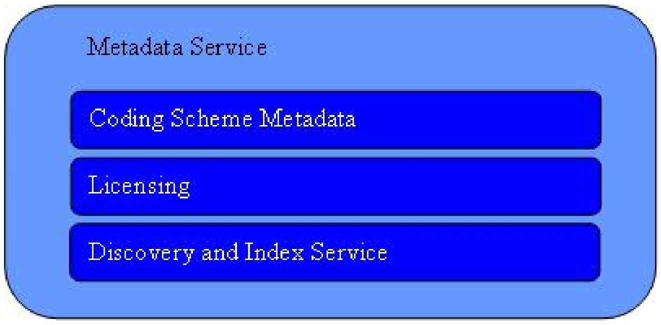
- Indexers
  Vocabularies may be indexed to provide enhanced performance or query capabilities. Types of indexes incorporated into the LexBIG system include but are not limited to the following:
  - Lexical Match – for example, "begins-with" and "contains"
  - Phonetic – allows for the ability to query based on "sounds-like" entry of search criteria.
  - Stemming – allows for the ability to find lexical variations of search terms.

Index creation is typically bundled into the load process. Architecturally speaking, however, this capability is decoupled and extensible.

- Loaders
  Vocabularies may be imported to the system from a variety of accepted formats, including but not limited to:
  - LexGrid XML (LexBIG canonical format)
  - NCI Thesaurus, provided in Web Ontology Language format (OWL)
  - UMLS Rich Release format (RRF)
  - Open Biomedical Ontologies format (OBO)

As with indexers, the load mechanism is designed to be extensible from an architectural standpoint. Additional loaders can be supported by the introduction of pluggable modules. Each module is implemented in the Java programming language according to a LexBIG-provided interface, and registered to the loader runtime environment.

Metadata and Discovery Subsystem



This subsystem provides information about accessible vocabularies, related licensing/copyright information, and registration/discovery of LexBIG services.

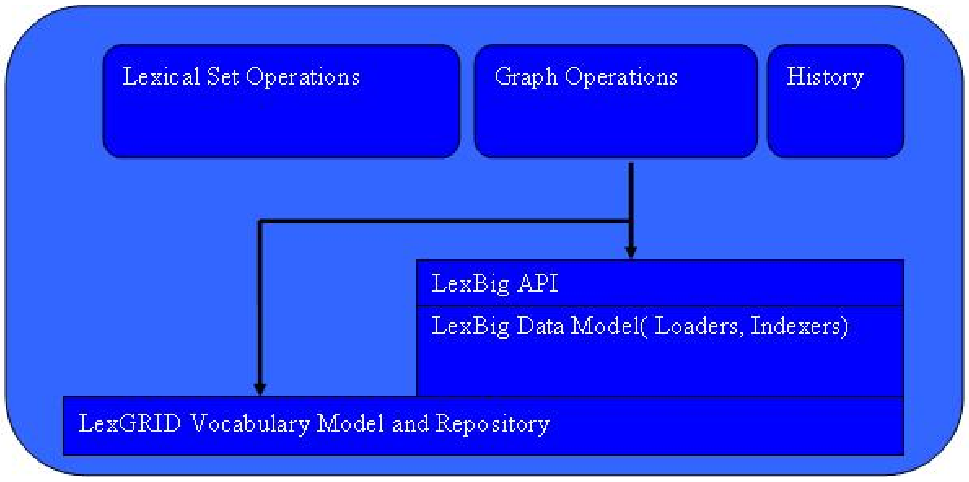
The ability to locate and resolve vocabulary metadata is fulfilled through the LexBIGService class. Metadata defined by the LexGrid information model is resolved with each CodingScheme instance. Available metadata on each resolved scheme includes, but is not necessarily limited to, the following:

- License or copyright information
- Supported values (e.g. supported concept status, language, property names, etc)
- Mappings from names used locally to globally unique URNs

In addition, each LexBIGService provides a centralized metadata index that allows registration and query of code system metadata without requiring resolution of individual CodingSchemes. This metadata index is optionally populated, typically during the vocabulary load process. The metadata index allows for the metadata of multiple code systems to be cross-indexed and searched as part of the query subsystem.

Finally, the LexBIG architecture provides the underpinnings for LexBIG services to be made accessible through the caGRID environment in the future, where vocabulary services might be deployed and discovered within a caGRID Globus container. However, this portion of the API is preliminary and awaits coordination with caBIG® Architecture WS designees to determine exact recommendations and nature of LexBIG services on the grid.

Query Subsystem


Lexical Set Operations
Graph Operations
History
LexBig API
LexBig Data Model(Loaders, Indexers)
LexGRID Vocabulary Model and Repository

This subsystem provides the functionality required to fulfill caCORE/EVS and other vocabulary requests. The Query Service is comprised of Lexical Operations, Graph Operations, Metadata, and History Operations.

Lexical Set Operations

Lexical Set Operations provides methods to return a lists or iterators of coded entries. Supported query criteria include the application of match/filter algorithms, sorting algorithms, and property restrictions. Support is also provided to resolve the union, intersection or difference of two node sets.

Graph Set Operations

Graph Operations support the subsetting of concepts according to relationship and distance, identification of relation source and target concepts, and graph traversal. Additional operations include enumeration and traversal of concepts by relation, walking of directed acyclic graphs (DAGs), enumeration of source and target concepts for a relation, and enumeration of relations for a concept.

Metadata Operations

Metadata Operations allows for the query and resolution of registered code system metadata according to specified coding scheme references, property names, or values.

History Operations

History provides vocabulary-specific information about concept insertions, modifications, splits, merges, and retirements when supplied by the content provider.

LexEVS API/Grid service interaction

See the LexEVS Grid Service Design and Implementation Guide.

Categories: VKC Contents | Documentation | LexEVS