Common Terminology Services CTS_2
Core Model Elements
0.96

Mayo Clinic

May 15, 2012

Contents

Chapter 1

Introduction

The CTS_2 specification is divided into a number of separably implementable profiles. This section on *Core Information Model Components* describes the data types, identifiers and common structures that are shared between two or more models.

Conventions

The CTS_2 specification was developed using a combination of UML and Z notation. The classes, attributes, method signatures and relationships were modeled using UML class diagrams, while the invariants, preconditions and post-conditions are written in Z. The specification can be published using UML notation with the invariants expressed informally or in Z notation, where the model can be formally validated using the FUZZ type checker. This, along with additional issues that may well reflect the authors' lack of understanding of some of the subtleties of UML modeling have led to a number of conventions.

Notation

Model elements are referenced using `Typewriter font`. Implementation profiles are referenced using **BOLD**

Identifiers

- Class names are capitalized camel case (`CodeSystemCatalog`, `String`)
- Attribute and role names are lower camel case (`version`, `sourceAndRole`)
- Enumeration values are all capital letters with undescores ('_') introduced as needed for clarity. (`EMPTY`, `STOP_ON_ERROR`)
- Method names are `lowerCamelCase`

Stereotypes

The following stereotypes are used throughout the model:

- **dataType** - UML dataType semantics
- **enumeration** - UML enumeration semantics
- **interface** - UML interface semantics
- **mixin** - a class that provides a certain functionality to be inherited by a subclass, while not meant for instantiation
- **opt** - an optional parameter in a method call. This stereotype should always be accompanied by a 0..1 cardinality and is provided to make optionality visible in the modeling tool
- **optparam** - an optional attribute in a class instance. This indicates that an implementation must be able to distinguish three alternatives: (1) add or change the attribute to whatever is in the class, (2) remove the attribute if its minimum cardinality is zero and (3) leave the attribute unchanged.

- **delta** - a method call that changes the state of the service. Note that the semantics of this stereotype is the inverse of the *is query* model attribute.
- **exception** - an exception
- **exceptionSet** - a set of exceptions, the members of which are represented by aggregation

Read Only Attributes

The CTS_2 PIM is based on a RESTful architectural style. Resources can be created, updated, read and/or deleted, and every resource has one or more immutable characteristics - characteristics that, if changed, would change the identity of the resource itself. In the information model we distinguish these characteristics from those that can be modified by declaring the identifying characteristics as *ReadOnly*. The computational model utilizes a pattern based on this information where the *create* operation(s) supply the identifying characteristics plus any of the non-optional mutable characteristics. The *update* operations then supply a unique identifier and a set of one or more mutable characteristics to be changed.

Chapter 2

Core Information Model Elements

Data Types

Axiomatic Data Types

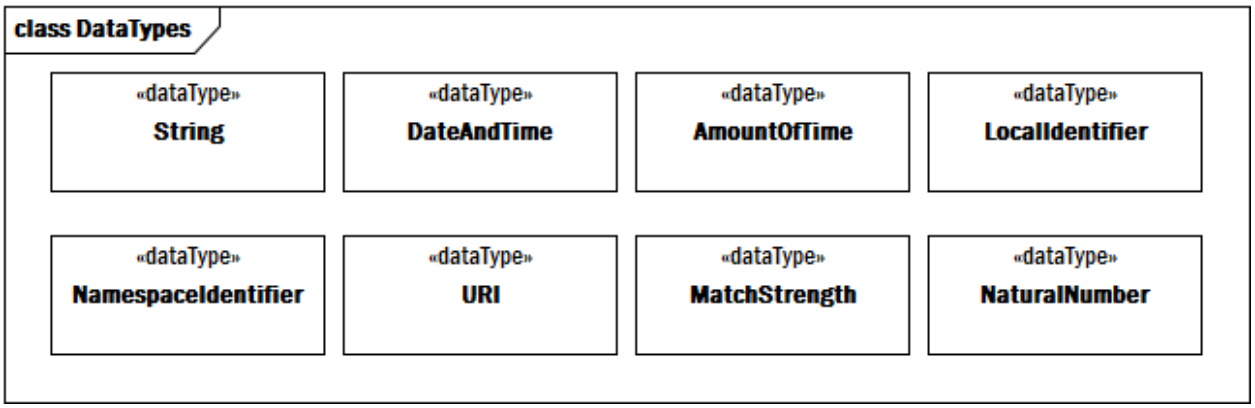


Figure 2.1: Axiomatic Data Types

This section identifies the core data types that are used throughout the remainder of the specification. The term“data type" refers to “a type whose instances are identified only by their value" [1]

Elements:

- **AmountOfTime** - a quantity of time. `AmountOfTime` is used exclusively to represent service timeout limits.
- **DateAndTime** - represents an “Instant" as defined in the OWL Time Specification [2]. CTS_2 implementations must be able to support temporal units of `second` , `minute` , `hour` , `day` , `month` , and `year` , and be able to represent and compare instances represented in any of these units. `DateAndTime` can only provide a partial ordering and, as a consequence, is never used as an index, unique identifier or to sequence data or events.
- **LocalIdentifier** - an identifier that uniquely references a class, individual, property or other resource within the context of a specific CTS_2 service implementation. `LocalIdentifier` syntax must match the PNAME_LN [3] production as defined in the SPARQL Query Specification [4] . `LocalIdentifiers` may begin with leading digits, where XML Local Identifiers and `NameSpaceIdentifier` s may not
- **MatchStrength** - represents the relative strength of the result of a search. Represented as a real number $\mathbb{R}$ such that $0.0 < MatchStrength \leq 1.0$.

[1] UML Infrastructure 2.3, p. 100
[2] http://www.w3.org/TR/owl-time/
[3] http://www.w3.org/TR/rdf-sparql-query/#rPNAME_LN
[4] http://www.w3.org/TR/rdf-sparql-query/

- **NamespaceIdentifier** - an identifier that uniquely references the scoping namespace of an `Entity` (class, role or individual) within a the context of a CTS_2 service. `NamespaceIdentifier` syntax must match the PNAME_NS [5] production as defined in the SPARQL Query Specification - meaning that it must begin alphabetic character. [6]
- **NaturalNumber** - a non-negative integer ($\mathbb{N}$). `NaturalNumber` is used exclusively for representing quantities in this specification.
- **String** - a non-empty sequence of characters. As terminological resources are often multilingual, it is expected that most CTS_2 Platform Specific Models (PSMs) will require that the `String` implementation support international character sets.
- **URI** - a Universal Resource Identifier (URI) as defined in IETF RFC 3986 [7]. CTS_2 implementations are encouraged to consider implementing this data type using the IRI (RFC3987 [8]) specification.

[5] http://www.w3.org/TR/rdf-sparql-query/#rPNAME_NS
[6] http://www.w3.org/TR/rdf-sparql-query/
[7] http://www.ietf.org/rfc/rfc3986.txt
[8] http://www.ietf.org/rfc/rfc3987.txt

EntryDescription and OpaqueData Types

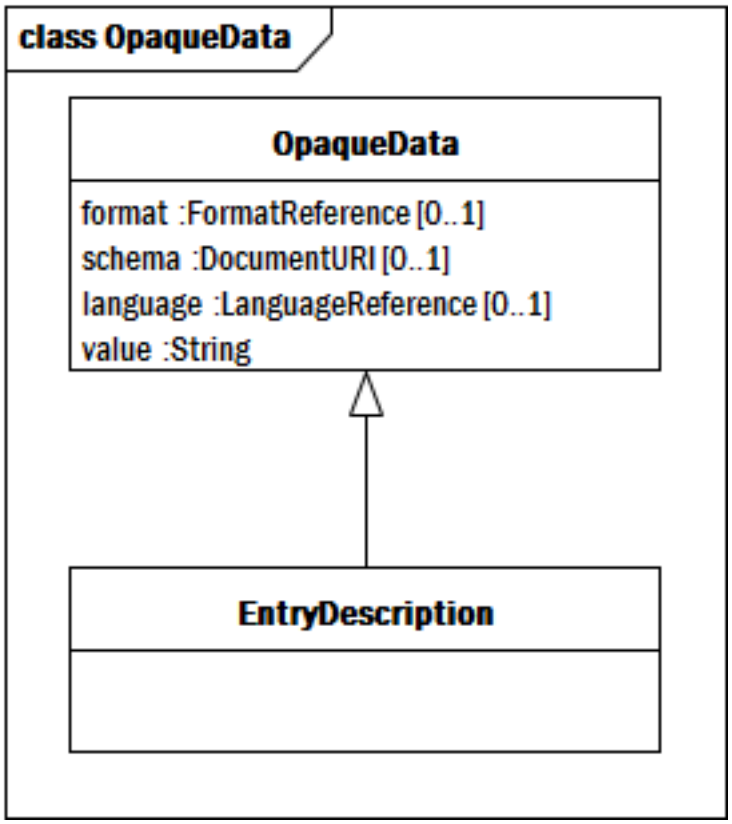


Figure 2.2: Opaque Data

Class EntryDescription

`EntryDescription` is a subclass of `OpaqueData` . The purpose behind this is that there are certain textual fields that some CTS_2 service implementations may want constrain. As an example, `Designation` text is of type `EntryDescription` , but implementations may want to restrict the `OpaqueData value` attribute to a simple string rather than xs:anyType. When `OpaqueData` appears directly as a model element, implementations must be able to support the full `OpaqueData` model. `EntryDescription` , however, may be constrained by implementations or specialized PSMs.

Superclasses:

- Every instance of `EntryDescription` is also an instance of `OpaqueData` .

Class OpaqueData

Opaque data is the equivalent of an ASN.1 External Type [9] or the XML Schema anyType [10]. An `OpaqueData` instance may represent text with an optional spoken or written language code or a formal structure such as embedded HTML, XML or MIME encoded data. When a formal structure is included, its type should be specified in the `format` attribute and, when the type is an XML variant, the corresponding schema (or DTD) should be included in the `schema` parameter.

The `OpaqueData` data type must be encoded in such a way that the content can be represented by a character string. Binary data is not permitted, although hyperlinks *to* binary data are.

[9] ISO/IEC 8824-1:2008 Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notion. Clause 37 - Notation for the external type

[10] http://www.w3.org/TR/xmlschema-0/#anyType

Attributes:

- **format** - the format or encoding for `value`. This is typically recorded as the URI of a Mime Type [11].
- **schema** - if the format of the document involves an XML encoding, this contains the URI of a document that carries the corresponding XML Schema or DTD.
- **language** - a reference to the written or spoken language used in `value` .
- **value** - the instance value. Note that instance value should be encoded in such a way that it allows embedded structures. As an example, in XML Schema, this encoding should be to xs:anyType or an equivalent.

[11] http://www.ietf.org/rfc/rfc2046.txt

Types of URI

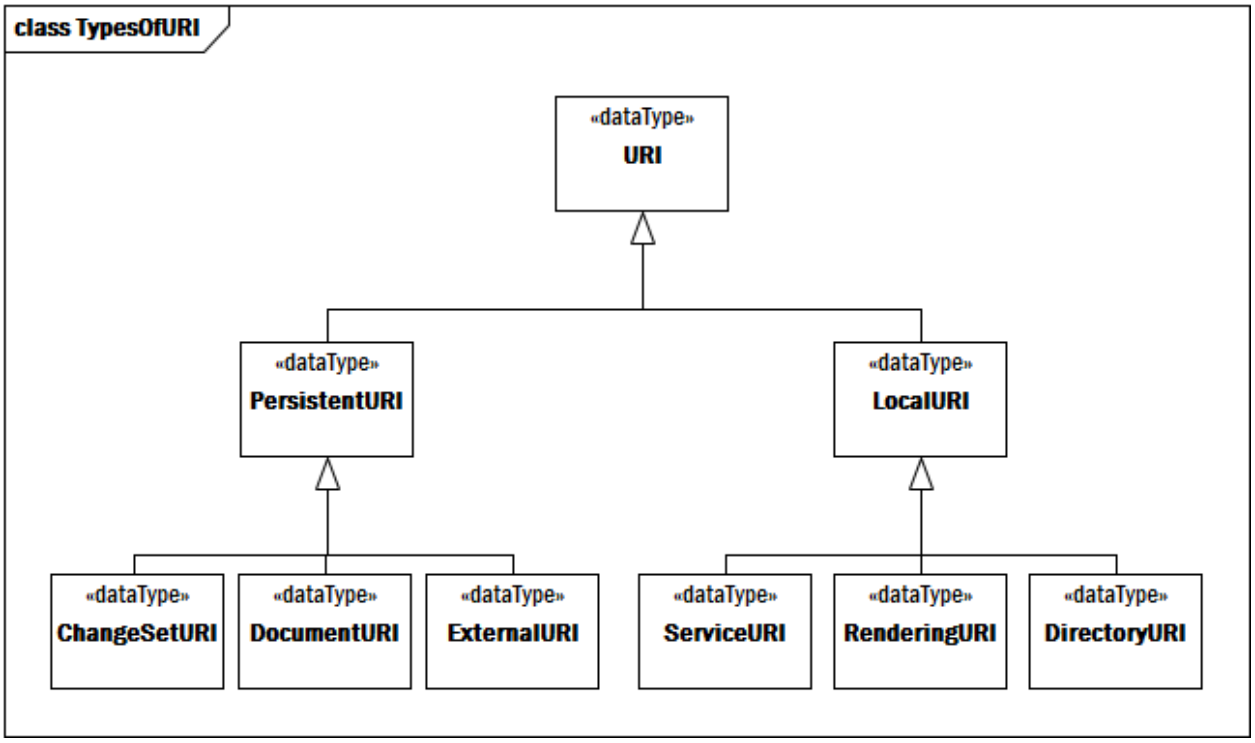


Figure 2.3: Types Of URI

URI's serve a variety of purposes, and separate labels are used to differentiate them:

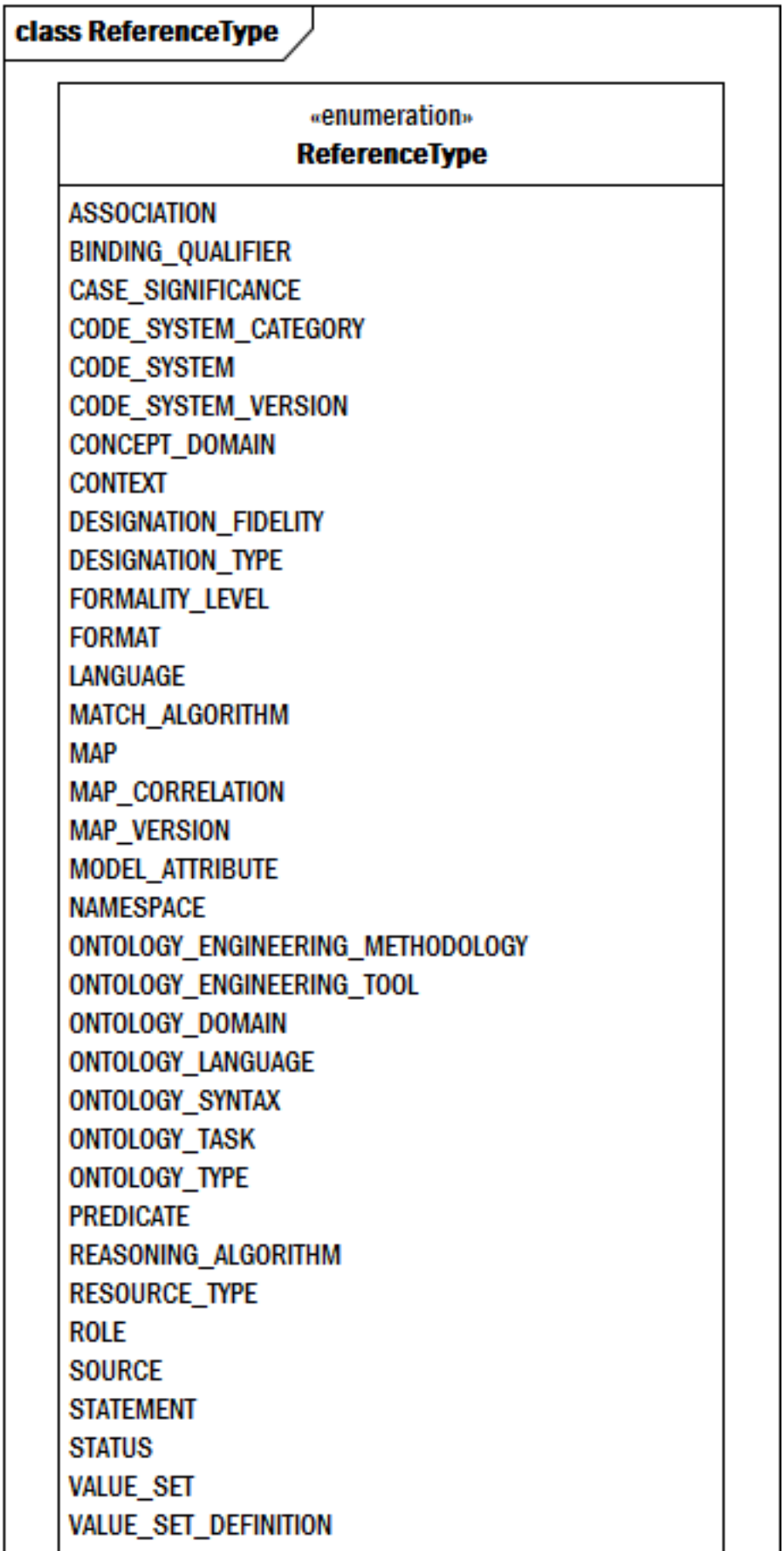
Elements:

- **ChangeSetURI** - the unique identifier of a set of change instructions that can potentially transform the contents of CTS_2 service instance from one state to another.
- **DirectoryURI** - the unique name of a query that, when executed results in a list of resources that, in the context of a given service, satisfy the query.
- **DocumentURI** - a reference to a "work" in the bibliographic sense. It is not necessary that a `DocumentURI` be directly or indirectly resolvable to a digital resource - it may simply be the name of a book, publication or other abstraction.
- **ExternalURI** - a URI that names a unique resource. CTS_2 implementations should never assume that `ExternalURI` is resolvable via an http: GET operation - `ExternalURI` s should always be passed as parameters to service implementations to get the sanctioned equivalent in a given service context
- **LocalURI** - a URI or handle whose scope is local to the implementing service. `LocalURI` cannot be used as a permanent identifier in a message or a data record.
- **PersistentURI** - a Universal Resource Identifier (URI) that persists across service instances. Persistent URI's have enduring reference and meaning.
- **RenderingURI** - a URI or handle that is directly readable by a specific instance of a CTS_2 service implementation. `RenderingURI` must resolve to `Changeable` CTS_2 element.
- **ServiceURI** - the URI or handle of a service implementation.
- **URI** - a Universal Resource Identifier (URI) as defined in IETF RFC 3986 [12]. CTS_2 implementations are encouraged to consider implementing this data type using the IRI (RFC3987 [13]) specification.

[12] http://www.ietf.org/rfc/rfc3986.txt
[13] http://www.ietf.org/rfc/rfc3987.txt

Resources, Local Identifiers and Entity References

Resource References



`ReferenceType` contains a list of all of the element types that appear in the CTS_2 specification. It includes CTS_2 resources (`CODE_SYSTEM`, `CODE_SYSTEM_VERSION`, `CONCEPT_DOMAIN`, `MAP_VERSION`, `VALUE_SET`, `VALUE_SET_DEFINITION`), changeable elements (`ASSOCIATION` , `ENTITY` , and `STATEMENT`) and external concept domains that are used in the model.

Enum ReferenceType

Attributes:

- **ASSOCIATION** - an formal "semantic" assertion about a named entity, in the form of subject, predicated and object including any provenance, qualifiers or internal BNODEs
- **BINDING_QUALIFIER** - an assertion about the semantics of a concept domain / value set binding. This model element exists specifically to address section 2.4.2.23 of the HL7 SFM [14], which needs a qualifier that "indicates whether the binding is "overall", "minimal" or "maximum.

  The CTS_2 specification does not formally define the semantics of the various possible `BINDING_QUALIFIER` elements - it is up to specific implementations and service clients to interpret the meaning of the specific binding qualifiers that may be represented in references of this type.
- **CASE_SIGNIFICANCE** - identifies the significance of case in a term or designation
- **CODE_SYSTEM_CATEGORY** - the general category of a code system (flat list, subject heading system, taxonomy, thesaurus, classification, terminology, description logic ontology, first order predicate logic, etc.) (same as KnowledgeRepresentationParadigm - OMV 5.8)
- **CODE_SYSTEM** - a collection of metadata about the provenance, use and distribution of a code system or ontology
- **CODE_SYSTEM_VERSION** - a collection of metadata about content and distribution format of a particular version or release of a code system
- **CONCEPT_DOMAIN** - the description of the conceptual domain of a field in a message, column in a database, field on a form, etc. Equivalent to the ISO 11179-3 "Data Element Concept".
- **CONTEXT** - external and environmental factors that serve to discriminate among multiple possible selections. While it is assumed that the specific contexts referenced by `CONTEXT` are represented by entity descriptions contained in some ontology or coding scheme, the CTS_2 specification does not recommend any targets. Note, however, the CTS_2 context is intended to represent the notion of "jurisdictional domain" or "realm" as described in the HL7 CTS2 SFM [15]
- **DESIGNATION_FIDELITY** - identifies how well a particular designation represents the intended meaning of a the referenced entity. CTS_2 implementations may consider using the SKOS [16] semantic relations to represent this relationship
- **DESIGNATION_TYPE** - the particular form or type of a given designation - can be "short name", "long name", "abbreviation", "eponym", ...
- **FORMALITY_LEVEL** - the level of formality of an ontology (OMV 5.9)
- **FORMAT** - a particular way that information is encoded for storage in a computer file [17]
- **LANGUAGE** - a spoken or written language intended for human consumption
- **MATCH_ALGORITHM** - a predicate that determines whether an entity resource qualities for membership in a set based on supplied matching criteria
- **MAP** - a set of rules that associate a set of entity references from one domain into those in another
- **MAP_CORRELATION** - an assertion about the strength or significance of a specific rule in a `Map`

[14] http://www.hl7.org/documentcenter/ballots/2009may/downloads/V3_CTS_R2_DSTU_2009OCT.pdf
[15] http://www.hl7.org/documentcenter/ballots/2009may/downloads/V3_CTS_R2_DSTU_2009OCT.pdf
[16] http://www.w3.org/TR/2009/REC-skos-reference-20090818/
[17] http://en.wikipedia.org/wiki/File_format

- **MAP_VERSION** - the state of a `Map` at a given point in time
- **MODEL_ATTRIBUTE** - an attribute defined in CTS_2 information model
- **NAMESPACE** - a reference to a conceptual space that groups identifiers to avoid conflict with items that have the same name but different meanings
- **ONTOLOGY_ENGINEERING_METHODOLOGY** - information about the ontology engineering methodology (OMV 5.4) (sic)
- **ONTOLOGY_ENGINEERING_TOOL** - a tool used to create the ontology (OMV 5.5)
- **ONTOLOGY_DOMAIN** - while the domain can refer to any topic ontology it is advised to use one of the established general purpose topic hierarchy like DMOZ or domain specific topic like ACM for the computer science domain. Only this way it can be ensured that meaningful information about the relation of the domains of two separate ontologies can be deduced (OMV 5.11)(sic)
- **ONTOLOGY_LANGUAGE** - information about the language in which the ontology is implemented (OMV 5.7)
- **ONTOLOGY_SYNTAX** - information about the syntax used by an ontology (OMV 5.6)
- **ONTOLOGY_TASK** - information about the task the ontology was intended to be used for (OMV 5.10)
- **ONTOLOGY_TYPE** - categorizes ontologies (OMV 5.2)
- **PREDICATE** - a property or relation between entities
- **REASONING_ALGORITHM** - a set of formal rules that allow the deduction of additional assertions from a supplied list of axioms
- **RESOURCE_TYPE** - a class of which a referencing resource is an instance of
- **ROLE** - a role that a `SOURCE` can play in the construction or dissemination of an terminological resource
- **SOURCE** - an individual, organization or bibliographic reference
- **STATEMENT** - an atomic assertion about a CTS_2 resource
- **STATUS** - the state of a resource or other entry in an external workflow
- **VALUE_SET** - a set of entity references
- **VALUE_SET_DEFINITION** - a set of rules that can be applied to specified versions or one or more code systems to yield a set of entity references
- **VERSION_TAG** - an identifier that can be assigned to resource versions by a service implementation to identify their state in the service workflow. Examples might include "development", "test", "production", etc.

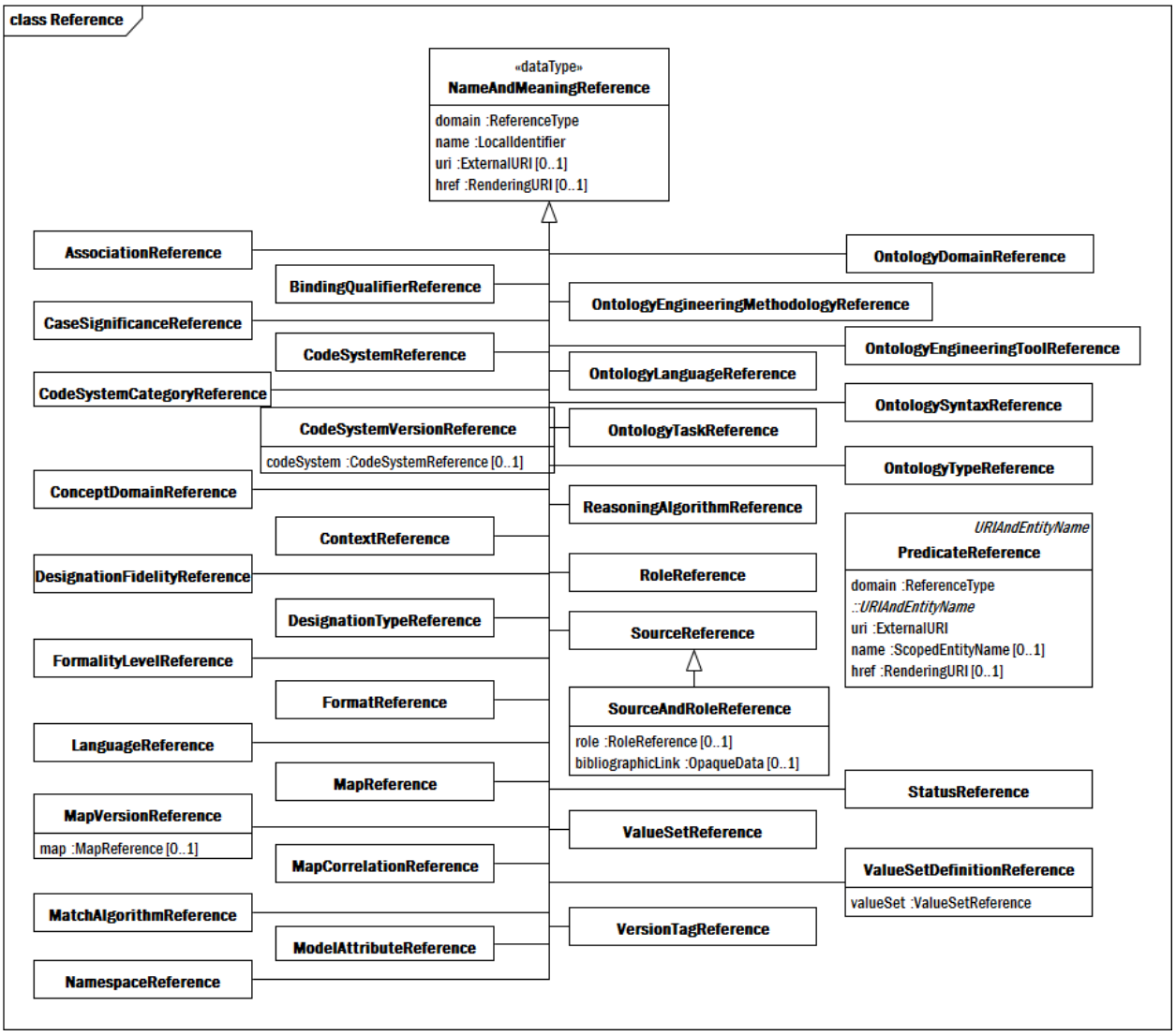


Figure 2.5: CTS_2 References

This diagram identifies all of the value domains that are used within the CTS_2 service itself. Each reference has a `domain` , which frequently isn't necessary to carry through to PSMs with strong type systems, a `name` that serves as a unique permissible value within the context of the `domain` , a `uri` that references the intended meaning of `name` and an optional `href` that, if present, provides a link to a CTS_2 `EntityDescription` that provides definitions, descriptions, etc. of the meaning.

The `NameAndMeaningReference` pattern represents, at the meta-level, one of the key purposes for a CTS_2 based service - to establish lists of value domains (identified here by `ReferenceType`) from which sets of permissible values and meanings can be drawn.

For each specific reference below we attempt to identify an ontology or value set from which the set of possible values could be drawn. We then state whether this set is *mandatory* - meaning that it must be used in a compliant CTS_2 implementation, *recommended* - meaning that, while not required, we anticipate that significant gains in interoperability would result were it used or simply *exemplar*, meaning that the set carries examples of what might be used.

Elements:

- **AssociationReference** - a name or identifier that uniquely names an association instance in a code system
- **BindingQualifierReference** - a reference to an entity that describes the role that a given value set binding plays for a concept domain. Typical values represent "overall", "minimum" or "maximum", the significance of which can be found in HL7 Version 3 documentation.

- **CaseSignificanceReference** - a reference to an entity that describes significance of the case in term or designation
- **CodeSystemCategoryReference** - a reference to information about a paradigm model used to create an ontology (aka. knowledge representation paradigm
- **CodeSystemReference** - a reference to a code system or ontology
- **CodeSystemVersionReference** - a reference to a specific version of code system and, if known, the code system which it is a version of
- **ConceptDomainReference** - a reference to a concept domain
- **ContextReference** - a reference to a realm or context
- **DesignationFidelityReference** - a reference to a description about designation faithfulness or accuracy
- **DesignationTypeReference** - a reference to a designation type or form such as "short name", "abbreviation", "eponym"
- **FormalityLevelReference** - a reference to a description of the relative formality an ontology
- **FormatReference** - a reference to a particular way that information is encoded for storage or transmission
- **LanguageReference** - a reference to a spoken or written human language
- **MapCorrelationReference** - a reference to a way that the source and target in a map can be related or assessed
- **MapReference** - a reference to an abstract map
- **MapVersionReference** - a reference to a map version and the corresponding map, if known
- **MatchAlgorithmReference** - a reference to an algorithm used for selecting and filtering data
- **ModelAttributeReference** - a reference to an attribute defined in the CTS_2 specification
- **NameAndMeaningReference** - A `NameAndMeaningReference` consists of a local identifier that references a unique meaning within the context of a given domain in a CTS_2 service instance and a globally unique URI that identifies the intended meaning of the identifier.
- **NamespaceReference** - a reference to a conceptual space that groups identifiers to avoid conflict with items that have the same name but different meanings
- **OntologyDomainReference** - a reference to a subject domain for an ontology
- **OntologyEngineeringMethodologyReference** - a reference to a method model that can be used to create an ontology
- **OntologyEngineeringToolReference** - a reference to a tool that can be used to create an ontology
- **OntologyLanguageReference** - a reference to a language in which an ontology may be implemented
- **OntologySyntaxReference** - a reference to a syntax in which an ontology may be represented
- **OntologyTaskReference** - a reference to a purpose for which an ontology can be designed
- **OntologyTypeReference** - a reference to the nature of the content of an ontology
- **PredicateReference** - An `EntityReference` that serves the role of predicate. Note that this varies slightly from the base class of `NameAndMeaningReference` because the `name` attribute is a namespace/name combination rather than a simple name scoped exclusively by the domain.
- **ReasoningAlgorithmReference** - a reference to a formal algorithm for making inferences about an ontology
- **RoleReference** - a reference to a role that an individual, organization or bibliographic reference can play in the construction of a resource or resource component
- **SourceAndRoleReference** - a reference to a source that also includes the role that the source played and/or fixes the particular chapter, page or other element within the reference

- **SourceReference** - a reference to an individual, organization of bibliographic reference
- **StatusReference** - a reference to a state in an external ontology authoring workflow
- **ValueSetDefinitionReference** - a reference to a set of rules for constructing a value set along with the corresponding value set if known
- **ValueSetReference** - a reference to a named set of entity references
- **VersionTagReference** - a reference to a tag that can be assigned to versionable resources within the context of a service implementation

Local Identifiers

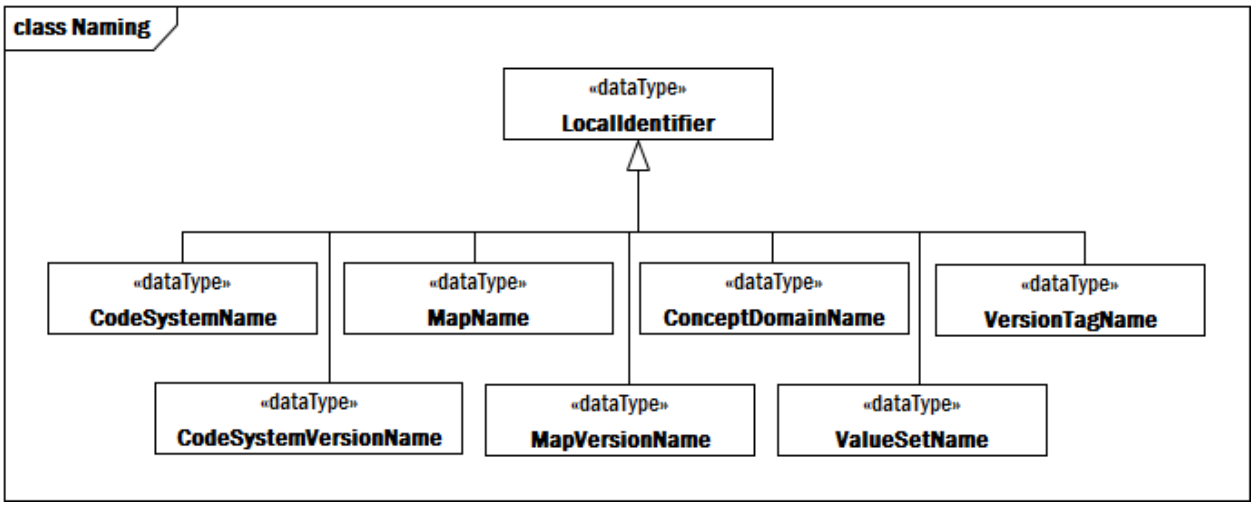


Figure 2.6: Local Identifiers

This section lists the specific types of local identifiers that are used within the CTS_2 specification. Instances of each type of local identifier must be unique within the context of the service instance. As an example, "SCT" might uniquely name the SNOMED-CT code system within the context of one service, while another service might use "SMD-CT". As a consequence, local identifiers can never be used in interchanges between services - URI's must be used instead. Note, also, that it is ok to have the same local identifier for different types of resource. As an example, the identifier "SCT" could be a `CodeSysteName` for the SNOMED-CT code system and a `ValueSetName` for the "Standardized Category Terms" value set.

Elements:

- **CodeSystemName** - a local identifier for a `CodeSystem`
- **CodeSystemVersionName** - a local identifier for a `CodeSystemVersion`
- **ConceptDomainName** - a local identifier for a `ConceptDomain`.
- **LocalIdentifier** - an identifier that uniquely references a class, individual, property or other resource within the context of a specific CTS_2 service implementation. `LocalIdentifier` syntax must match the PNAME_LN [18] production as defined in the SPARQL Query Specification [19] . `LocalIdentifiers` may begin with leading digits, where XML Local Identifiers and `NameSpaceIdentifier` s may not
- **MapName** - a local identifier for a `Map`
- **MapVersionName** - a local identifier for a `MapVersion`

[18] http://www.w3.org/TR/rdf-sparql-query/#rPNAME_LN
[19] http://www.w3.org/TR/rdf-sparql-query/

- **ValueSetName** - a local identifier for a `ValueSet`
- **VersionTagName** - a local identifier for a `VersionTag`

Entity References

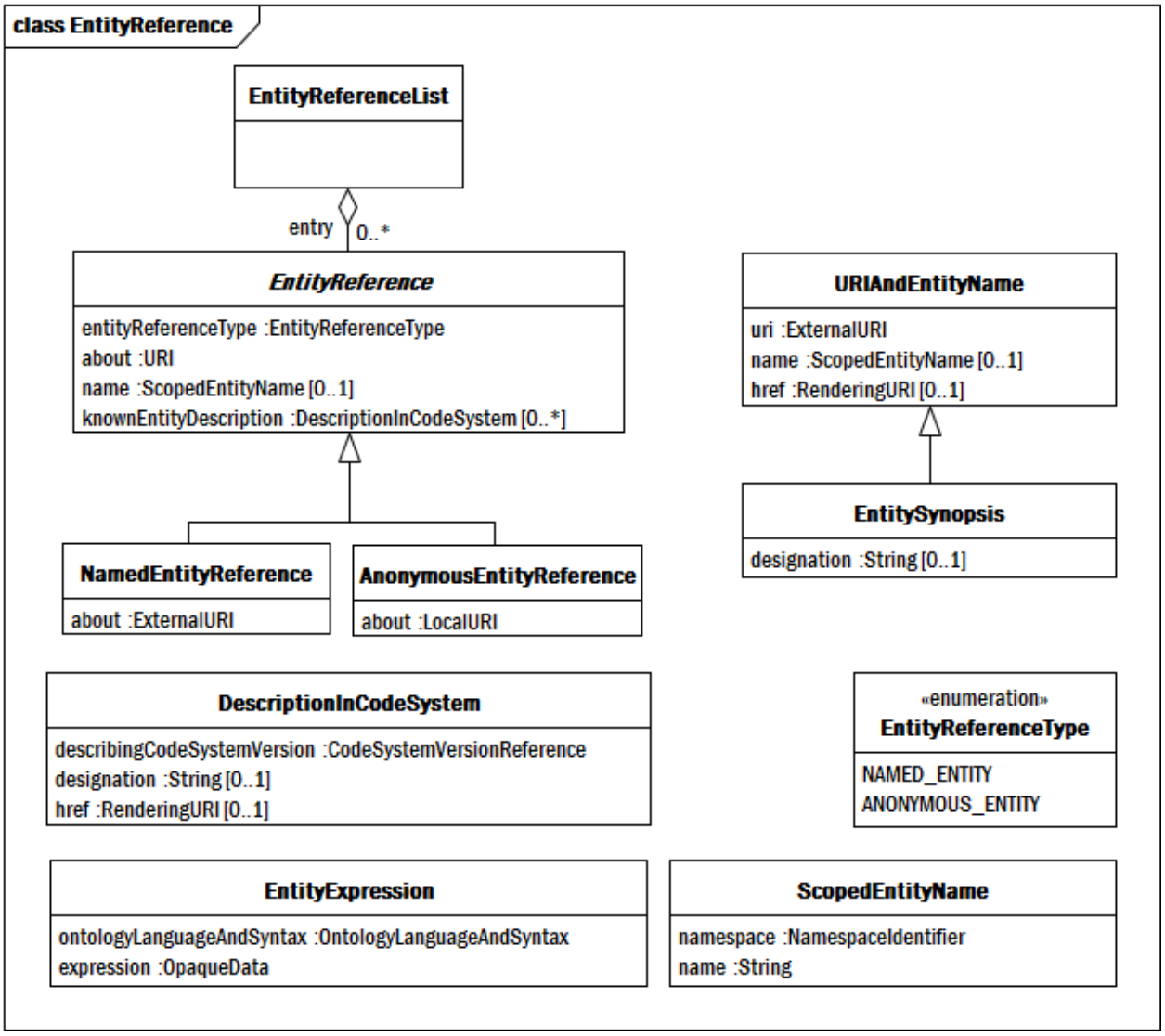


Figure 2.7: Entity Reference

CTS_2 differentiates between a simple resource reference, such as a code system, code system version, value set, etc. and a reference to an `Entity` - a class, predicate or individual. Simple resource references are identified by a single `URI`. Entity references, however, are subdivided into two parts - an scoping `namespace` and a `name` that is unique within the context of the namespace.

The diagram above contains three building blocks that are used for referencing entities throughout the specification. The first form, `URIAndEntityName` provides the URI and local name by which the entity is known within the context of the service. An optional `href` may also be supplied that resolves to the `EntityDescription` that is contextually appropriate.

The second form, `EntityReference` , supplies the uri and name but also includes a list of code system versions that make one or more assertions about (or using) the referenced entity. There will be at most one version of any given code system in this list, the choice of which will depend on the context of the query.

The third form, `EntityExpression` , is a description of an `Entity` in an external language and syntax such as RDF/OWL, Manchester OWL or SNOMED CT Compositional Grammar.

Class AnonymousEntityReference

A reference to an entity whose name and description is local to the containing code system and service. Information about anonymous entities cannot be shared between services or across code systems

Superclasses:

- Every instance of `AnonymousEntityReference` is also an instance of `EntityReference.`

Attributes:

- **about** - the local URI that identifies this entity within the context of the particular service

Invariants:

1. The `entityReferenceType` of an `AnonymousEntityReference` must be `ANONYMOUS_ENTITY` .
2. Anonymous entity references must have a local entity name, where the namespace is equal to the containing code system version namespace .
3. An `AnonymousEntityReference` must be described by exactly one code system version. Anonymous entities cannot be shared across code systems or services. .

Class DescriptionInCodeSystem

A reference to specific version of a code system that contains assertions about a given entity, including the namespace and name by which the entity is referenced, an optional designation appropriate to the given usage context and an optional `RenderingURI` that references the full `EntityDescription` contained in the specific code system version.

Attributes:

- **describingCodeSystemVersion** - a reference to the code system version that describes the entity
- **designation** - a contextually appropriate designation for the entity derived from the `describingCodeSystemVersion`
- **href** - a `RenderingURI` that, if followed, will provide a full CTS_2 `EntityDescription` derived from the corresponding code system version

Class EntityExpression

An expression in a given ontology language and syntax that describes or defines an entity. Examples might include descriptions of entities in Manchester OWL Syntax, RDF, SNOMED Concept Expression, etc.

Attributes:

- **ontologyLanguageAndSyntax** - the ontology language and syntax of the expression
- **expression** - the actual expression

Class EntityReference

The URI, namespace/name (if known) and a list of code systems that make assertions about the entity.

Attributes:

- **entityReferenceType** - the discriminant of the entity reference - whether it is named or anonymous
- **about** - the entity URI. This is an `ExternalURI` if the `entityReferenceType` is `NAMED_ENTITY` and a `LocalURI` is the `entityReferenceType` is `ANONYMOUS_ENTITY`.
- **name** - the namespace and name by which this entity is known within the context of the service implementation
- **knownEntityDescription** - a reference to a version of a code system that makes one or more assertions about the referenced entity. Note that only one version of a given code system is allowed in the `describingCodeSystem` list. Unless specified otherwise in a specific call, the code system version with the tag "CURRENT" must be used.

Invariants:

1. There can be at most one version of any code system in an entity reference .

Class EntityReferenceList

A list (set) of zero or more entity references

Attributes:

- **entry** - An entry in a list of EntityReferences

Class EntitySynopsis

The URI, local namespace and name and optional designation of an `EntityDescription` . `EntitySynopsis` represents entities when they are referenced from the context of a single code system, such as the resolution of value sets and association graphs.

Superclasses:

- Every instance of `EntitySynopsis` is also an instance of `URIAndEntityName`.

Attributes:

- **designation** - a designation considered appropriate for the entity in the specific context of use

Class NamedEntityReference

A reference to an entity that is identified through a globally unique URI.

Superclasses:

- Every instance of `NamedEntityReference` is also an instance of `EntityReference`.

Attributes:

- **about** - a URI that identifies a unique entity in a global context

Invariants:

1. The `entityReferenceType` of a `NamedEntityReference` must be `NAMED_ENTITY` .

Class ScopedEntityName

The combination of a namespace identifier and a local name. Scoped entity names are not portable - they only work within the context of a single service instance, as different services may assign different namespace identifiers to the same namespace and different services may make different choices of the appropriate local identifier to use to represent an entity. As an example, one service may choose to use the entity code while a second may use another designation that is known to be unique.

Attributes:

- **namespace** - an identifier that references a unique namespace URI within the context of the service
- **name** - the local entity name within the context of the namespace. What is chosen for the entity name is service specific

Class URIAndEntityName

The combination of a URI and/or and ScopedEntityName.

Attributes:

- **uri** - a URI that uniquely references the target entity
- **name** - a namespace/name combination that uniquely represents the entity. This can be the primary `entityID` , as determined by the service or any valid `alternateId` . Service implementers are encouraged to develop mechanisms that will allow clients to choose an appropriate namespace for rendering `URIAndEntityName` instances. As an example, it should be possible to view SNOMED-CT entity references by either the SctId, the "fully specified name" or, where appropriate, the CTV3ID or SNOMED-3 identifier. Similar mechanisms would apply to ontologies that have both id and label fields.
- **href** - a URI that resolves to the full `EntityDescription` represented by this resource

Enum EntityReferenceType

The discriminant for the type of entity reference.

Attributes:

- **NAMED_ENTITY** - the entity reference is named
- **ANONYMOUS_ENTITY** - the entity reference is anonymous

Building Blocks

Annotations

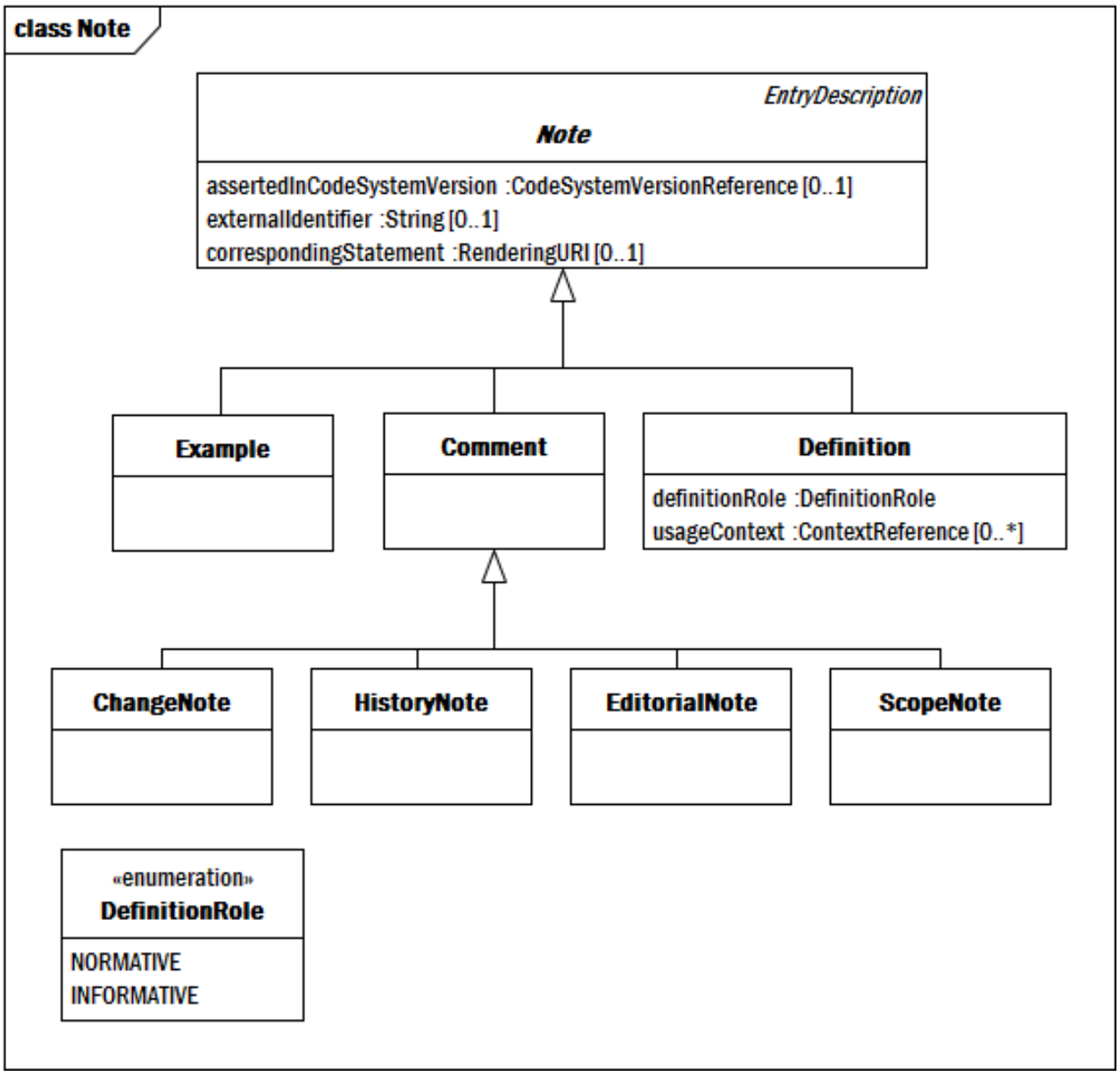


Figure 2.8: Note

The `Note` model is derived from the model defined in skos:note [20]. It differs, however, on a couple of key points:

1. The CTS_2 model does not group examples and definitions in the same category as comments. There are many circumstances where comments are appropriate but definitions and examples are not.

2. `Definition` has been extended to include the role that the definition plays and the contexts in which it applies.

Class ChangeNote

A change note. See: skos:changeNote

[20] http://www.w3.org/TR/skos-reference/#notes

Superclasses:

- Every instance of `ChangeNote` is also an instance of `Comment`.

Class Comment

A `Note` about the history, scope or provenance of the containing element. `Comment` is differentiated from `Example` and `Definition` specifically because it is frequently searched and displayed under different circumstances and usage contexts. `Example` and `Definition` are frequently made available to the end users of a code system while `Comment` s are typically consumed by authors and editors.

Superclasses:

- Every instance of `Comment` is also an instance of `Note`.

Class Definition

Text or other representation that is intended to communicate the intended meaning of the associated entity to a human being. While this is intended to be very close in meaning to skos:definition [21], its intent is slightly different in that the CTS_2 specification does not treat `definition` as a subproperty of `note` - rather it views comments, examples and definitions as separate entities.

Superclasses:

- Every instance of `Definition` is also an instance of `Note`.

Attributes:

- **definitionRole** - the role that the definition plays with respect to the defined entity
- **usageContext** - the context(s) in which the definition is considered applicable

Class EditorialNote

An editorial note. See skos:editorialNote

Superclasses:

- Every instance of `EditorialNote` is also an instance of `Comment`.

Class Example

An example. See: skos:example [22]

Superclasses:

- Every instance of `Example` is also an instance of `Note`.

Class HistoryNote

A history note. See: skos:historyNote

Superclasses:

- Every instance of `HistoryNote` is also an instance of `Comment`.

[21] http://www.w3.org/TR/skos-reference/#definition
[22] http://www.w3.org/TR/skos-reference/#L1693

Class Note

`Note` corresponds to the target of skos:note [23]. It contains an attributed literal that may include a language, format and, when appropriate, schema.

Superclasses:

- Every instance of `Note` is also an instance of `EntryDescription`.

Attributes:

- **assertedInCodeSystemVersion** - the code system version that contains the assertion(s) represented in the `Note`
- **externalIdentifier** - an external identifier assigned to this note by an outside party
- **correspondingStatement** - the URI of the `Statement` from which this note was derived. This will only be present in services that support the STATEMENT profile.

Class ScopeNote

A scope note. See: skos:scopeNote

Superclasses:

- Every instance of `ScopeNote` is also an instance of `Comment`.

Enum DefinitionRole

The role that a given definition plays for a given entity.

Attributes:

- **NORMATIVE** - the definition is considered to be official or normative by the assigning body
- **INFORMATIVE** - the definition is considered to be of value, but not completely normative

[23] http://www.w3.org/TR/skos-reference/#notes

Statement Target Model

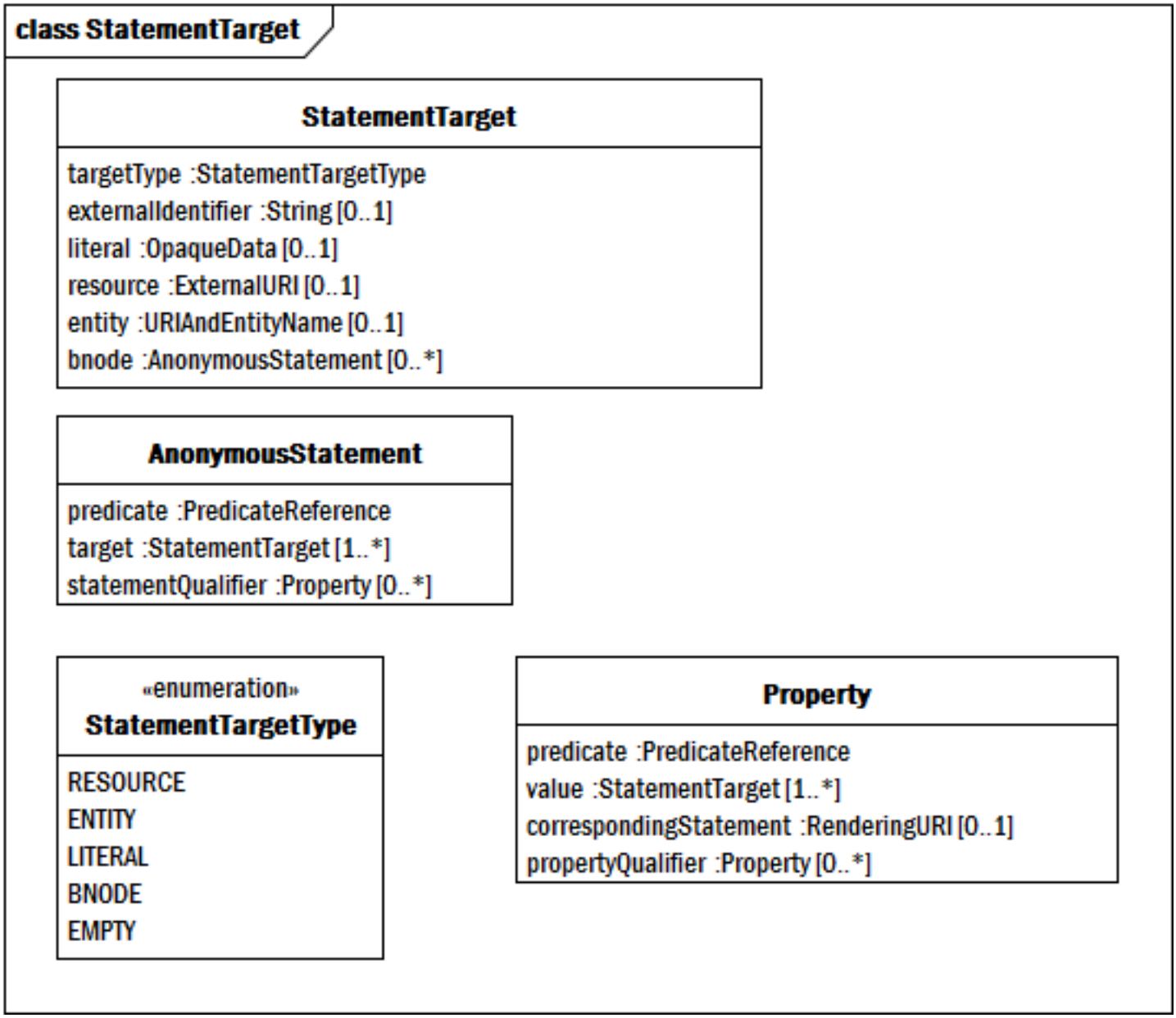


Figure 2.9: Statement Target

`StatementTarget` is an extension to the *object* model in RDF. It provides two minor enhancements:

1. `StatementTarget` distinguishes between simple URI references and references to entities that are described in code systems. In the latter case, the reference includes two additional bits if information - the namespace and local identifier for the entity and, where appropriate, references to the code system version(s) that make assertions about the target.
2. `StatementTarget` does not represent BNodes as first class statements - BNodes are *contained* in the statement itself. This eliminates the need to deal with pseudo-identifiers and other issues surrounding these items.

Class AnonymousStatement

A statement lacking a named subject.

Attributes:

- **predicate** - the predicate of the anonymous statement
- **target** - the target of the anonymous statement
- **statementQualifier** - assertions whose subject is the anonymous statement

Class Property

A tag/value pair that does not have a corresponding model attribute. `Property` represent any statement about a resource (e.g. CodeSystem, Entity, etc.) that does not have a corresponding attribute in the CTS_2 model. As an example, the NCI Thesaurus uses a tag named *BioCarta_ID* (C43677) to associate appropriate thesaurus entities with Bio Carta pathway references. This would be represented by a property, whose predicate is the URI for C436777 and the value is the actual id.

Attributes:

- **predicate** - the name and URI of the property predicate
- **value** - the target(s) of the property. Note that this can only represent the literal format of the property. The details about the original property will be found in the `correspondingStatement` if the CTS_2 implementation supports the STATEMENT profile.
- **correspondingStatement** - a link to the original statement from which this `Property` is derived. Will only be present in CTS_2 implementations that support the STATEMENT profile.
- **propertyQualifier** - an assertion whose subject is the assertion in the property instead of the property subject

Class StatementTarget

the target of a `Statement`. `StatementTarget` represents one of a literal value, a reference to a non-entity type `resource` , an `entity`, or an anonymous blank `BNODE` .

Attributes:

- **targetType** - the target discriminant
- **externalIdentifier** - an external identifier that has been assigned to the statement with this particular target by the authoring body. As an example, this would carry the SctId if the authoring body were SNOMED-CT
- **literal** - the literal target when the statement type is `LITERAL`.
- **resource** - the resource URI when the statement type is `RESOURCE`
- **entity** - the URI and optional namespace/name when the target type is `ENTITY`
- **bnode** - a collection of statements about an anonymous subject

Invariants:

1. If the statement target is `Resource` , `resource` must be present .
2. If the statement target is `Entity` , `entity` must be present .
3. If the statement target is `Literal` , `literal` must be present .
4. if the statement target is a `BNODE` , it cannot have a `literal` , `resource` or `entity` .
5. if the statement target is EMPTY , it cannot have any content .
6. there can be at most one `literal` , `resource` or `entity` .

Enum StatementTargetType

a discriminant that determines the type of `StatementTarget`

Attributes:

- **RESOURCE** - the statement target is a resource URI
- **ENTITY** - the statement target is an entity reference
- **LITERAL** - the statement target is a literal
- **BNODE** - the statement target is a blank node - a collection of statements without a named subject
- **EMPTY** - the `StatementTarget` represents the empty set or null

Directories and Lists

Directory and List Model

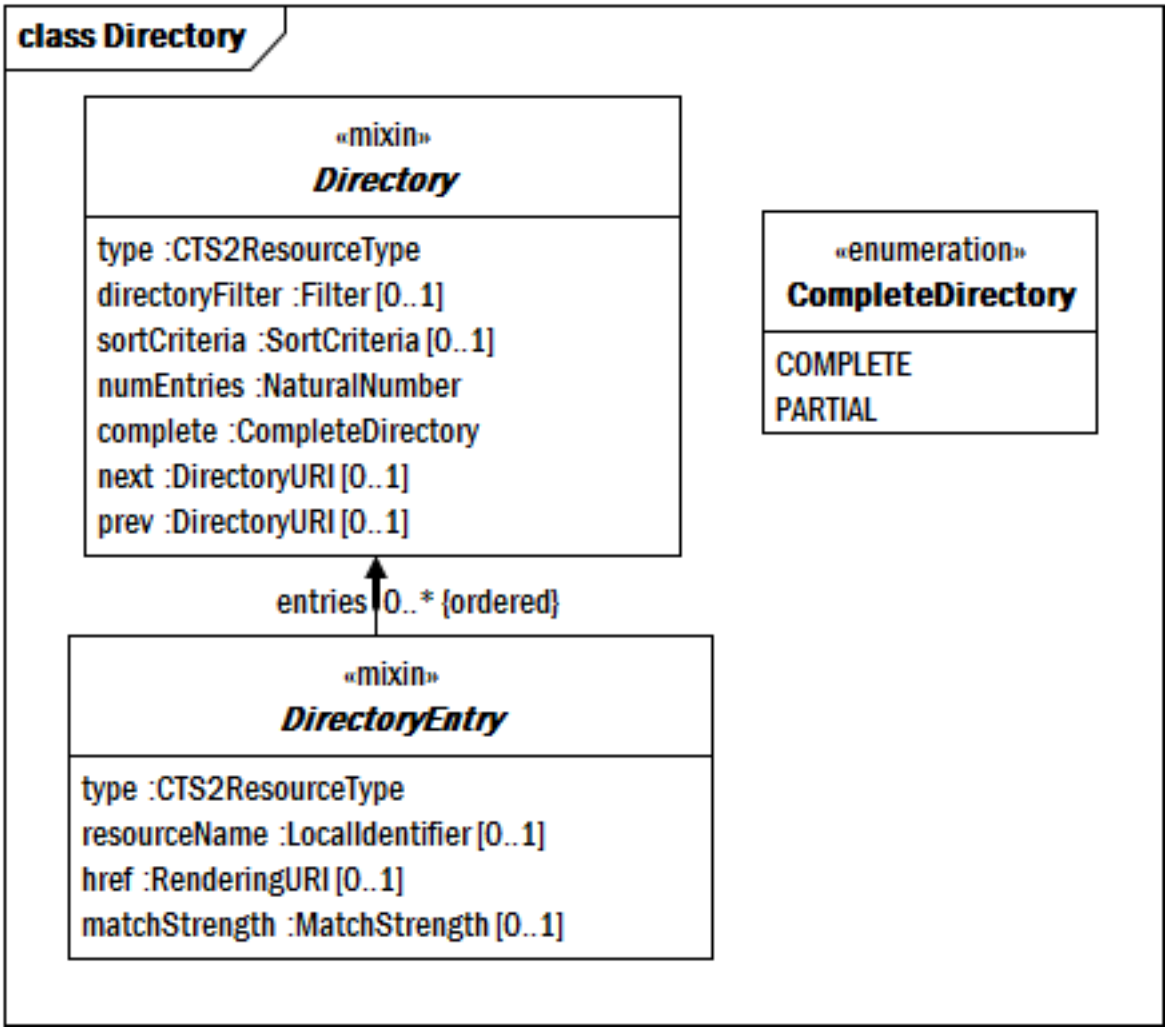


Figure 2.10: Directories and Lists

Directories contain the results of queries. A query is formed by starting with a base `DirectoryURI` , applying one or more optional filters and then invoking the appropriate `resolve` operation in the corresponding service. As an example, if one wanted to get a list of all entity descriptions that have the word "oncology" in one or more designations, one would start at the `EntityDescription` query service. One would first fetch the `EntityDirectoryURI` that represented all entities known to the service. One would then invoke the `restrict` operation with a filter referencing the `designation` attribute, a "contains" match algorithm and the text "oncology. This would return a second `EntityDirectoryURI` that represented that set. One would then invoke the `resolve` operation in the service to return a `Directory` summarizing the matching `EntityDescriptions` or, alternatively, the `resolveAsList` operation to return a directory of complete `EntityDescriptions`.

Class Directory

Provides a directory or list of resources that match a specific filter and are ordered in a specified order. A directory is read-only and is not necessarily immutable.

Note that the name of the link to `DirectoryEntry` is called "entries" rather than "entry". This allows subclasses to use "entry" without type collisions in the Z.

Attributes:

- **type** - the type of directory being represented. This value is asserted by the implementing subclass.
- **directoryFilter** - the filter(s) that were applied to generate this directory
- **sortCriteria** - the sort criteria used in the directory. When this is left as optional, there is the possibility that some directories may not be ordered. Some PSMs may chose to make `sortCriteria` mandatory, meaning that all directory listings must reflect some sort order.
- **numEntries** - the number of entries in this directory segment. Note that this is *not* the total number of entries in the complete directory listing - just the number of entries in this segment.
- **complete** - an indicator that states whether the complete directory listing is included in `entries` or whether additional retrievals are needed to get the full listing.
- **next** - a URI that, when de-referenced, produces the next set of entries in the directory.
- **prev** - a URI that, when de-referenced, produces the preceding set of entries in the directory.
- **entries** - represents an entry that matches the supplied directory filter criteria

Invariants:

1. Either a directory is `COMPLETE` or it will have a `next` and/or a `prev` entry. .
2. The number of entries in the directory header is exactly the number of entries in the entry list. .

Class DirectoryEntry

an entry in a directory. `DirectoryEntry` is a mixin that identifies the attributes that must or may be present in any element that can be represented as an entry in a `Directory` .

Attributes:

- **type** - the type of the entry. Note that this attribute may not appear in platform specific models (PSMs), as the type system of the implementation itself may serve to identify the particular resource type.
- **resourceName** - a local identifier that names a unique resource within the context of `type` and the service context. This attribute must be present if the corresponding element has a local name. Note, however, that some elements (e.g. `Association` , `ValueSetDefinition` , etc.) do not local names and are identified exclusively by their `href` .
- **href** - a `LocalURI` that resolves to the full resource described by the `DirectoryEntry` . This should be present if the service either (a) supports the READ functional profile for the specified `type` or (b) is aware of another service that does.
- **matchStrength** - a relative measure of the "goodness of fit" of the directory entry within the context of the `directoryFilter` in the containing `Directory`.

Enum CompleteDirectory

An indicator that determines whether a `Directory` contains all of the qualifying entries or only some.

Attributes:

- **COMPLETE** - the `Directory` contains all of the qualifying entries
- **PARTIAL** - the directory contains only a partial listing of the qualifying entries

Directory Types

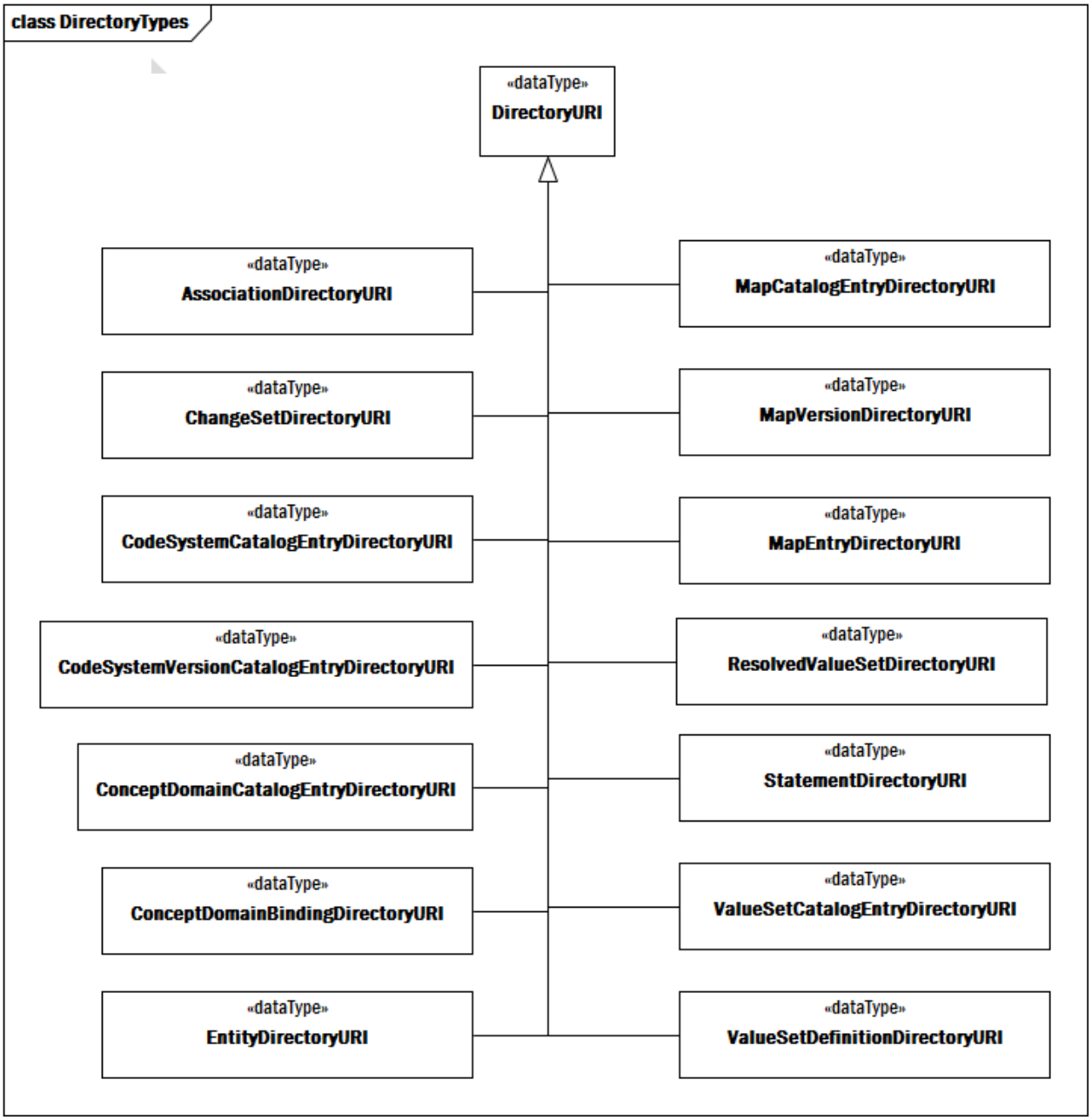


Figure 2.11: Directory Implementations

A directory of `CodeSystems`

Elements:

- **AssociationDirectoryURI** - a `DirectoryURI` that references a set of `Associations`.
- **ChangeSetDirectoryURI** - a `DirectoryURI` that references a set of `ChangeSets`.
- **CodeSystemCatalogEntryDirectoryURI** - a `DirectoryURI` that references a set of `CodeSystemCatalogEntries` .

- **CodeSystemVersionCatalogEntryDirectoryURI** - a `DirectoryURI` that references a set of `CodeSystemVersionCatalogEn` .
- **ConceptDomainBindingDirectoryURI** - a `DirectoryURI` that references a set of `ConceptDomainBindings`.
- **ConceptDomainCatalogEntryDirectoryURI** - a `DirectoryURI` that references a set of `ConceptDomainCatalogEntries`
- **DirectoryURI** - the unique name of a query that, when executed results in a list of resources that, in the context of a given service, satisfy the query.
- **EntityDirectoryURI** - a `DirectoryURI` that references a set of `EntityDescriptionDirectory`
- **MapCatalogEntryDirectoryURI** - a `DirectoryURI` that references a set of `MapCatalogEntries`
- **MapEntryDirectoryURI** - a `DirectoryURI` that references a set of `MapEntrie` s
- **MapVersionDirectoryURI** - a `DirectoryURI` that references a set of `MapVersions`
- **ResolvedValueSetDirectoryURI** - a `DirectoryURI` that references a set of `ValueSetCatalogEntries`
- **StatementDirectoryURI** - a `DirectoryURI` that references a set of `Statements`
- **ValueSetCatalogEntryDirectoryURI** - a `DirectoryURI` that references a set of `ValueSetCatalogEntries`
- **ValueSetDefinitionDirectoryURI** - a `DirectoryURI` that references a set of `ValueSetDefinitions`

Filters and Sorting

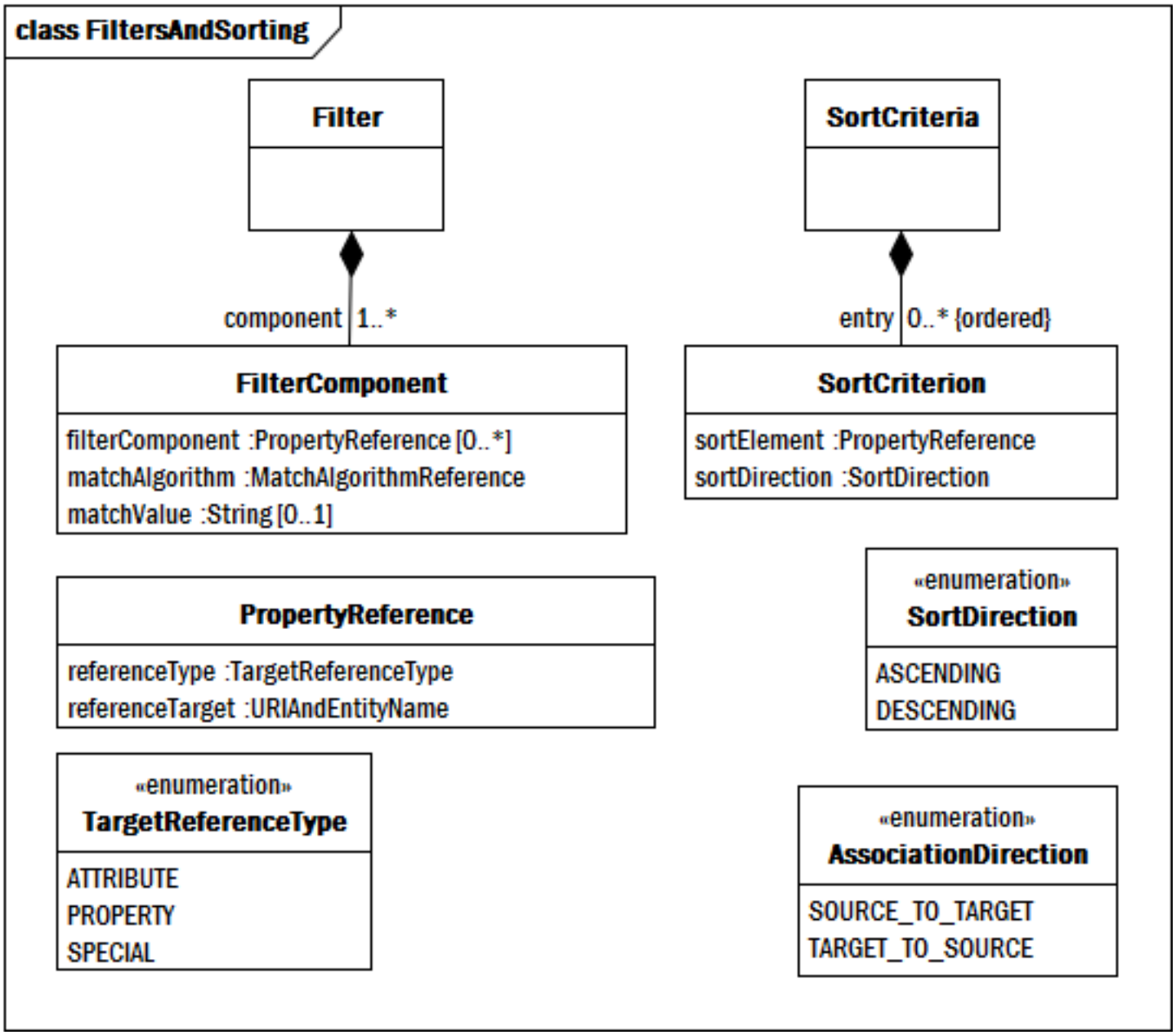


Figure 2.12: Filters and Sorting

The above diagram contains the elements that are used in the construction of query filters and return sort criteria. `AssociationDirection` is also included in this diagram because it crosses modules and we didn't want to create a separate diagram with just one element.

Class Filter

A collection of one or more filters. The result of applying a `Filter` component is the *intersection* of the sets of qualifying elements. As an example, a filter having two components - one which says that the `rights` attribute must exist and a second that says that the text "SNOMED" must appear in the synopsis would return all resources having BOTH a `rights` attribute and "SNOMED" in the description.

Attributes:

- **component** - an entry in a filter

Class FilterComponent

A restriction on an attribute, property or special field

Attributes:

- **filterComponent** - the name or URI of the property or model element to be filtered. Properties are referenced by their predicate and model elements all have URI's that are established by this specification.
- **matchAlgorithm** - the algorithm to be used for testing the referenced component. Examples might include "starts with", "regular expression match", "exists", "inRange", etc.

  NOTE: The CTS_2 specification needs to establish a core set of match algorithms that all compliant implementations must support.
- **matchValue** - the value to be used in comparison. The structure and format of `matchValue` depends on the specific `matchAlgorithm` . As an example, a "startsWith" algorithm would be plain text, a "regularExpression" algorithm would have a regular expression while an "exists" algorithm would have nothing in the `matchValue` attribute.

Class PropertyReference

A reference to a CTS_2 model element. `PropertyReference` may reference a model attribute, a `Property` or a special element such as match strength.

Attributes:

- **referenceType** - the type of thing being referenced
- **referenceTarget** - a reference to the model attribute or predicate that is used for sorting or filtering

Class SortCriteria

An ordered list of sort criterion. The first entry in the list identifies the primary sort order, the second entry the sub sort order, etc.

Attributes:

- **entry** - a rule for sorting

Class SortCriterion

The particular attribute, property or special element to be sorted along with the sort direction

Attributes:

- **sortElement** - the type and name of the attribute, property or special element to be sorted
- **sortDirection** - the sort order

Enum AssociationDirection

An indicator that determines whether an entity reference / predicate combination is to be evaluated with the entity reference in the source (left hand side) position of the association query or the target (right hand side or "object") position.

Note: `AssociationDirection` doesn't strictly belong in this diagram, but it is kind of an orphan class and we didn't want to create an entire new diagram just for one element.

Attributes:

- **SOURCE_TO_TARGET** - the statement is to be resolved with the entity reference in the role of `source` .
- **TARGET_TO_SOURCE** - the statement is to be resolved with the entity reference in the role of `target` .

Enum SortDirection

The collating order of a sort.

Attributes:

- **ASCENDING** - sort in ascending collation order
- **DESCENDING** - sort in descending collation order

Enum TargetReferenceType

the possible types of property references

Attributes:

- **ATTRIBUTE** - a reference to an attribute in a CTS_2 model such as `formalName` , `designation` , etc.
- **PROPERTY** - a reference to a model `Property` . The reference target carries the namespace/name or URI of the property predicate.
- **SPECIAL** - the target of the reference is a special element such as the match strength of a search

Change Model

Updates

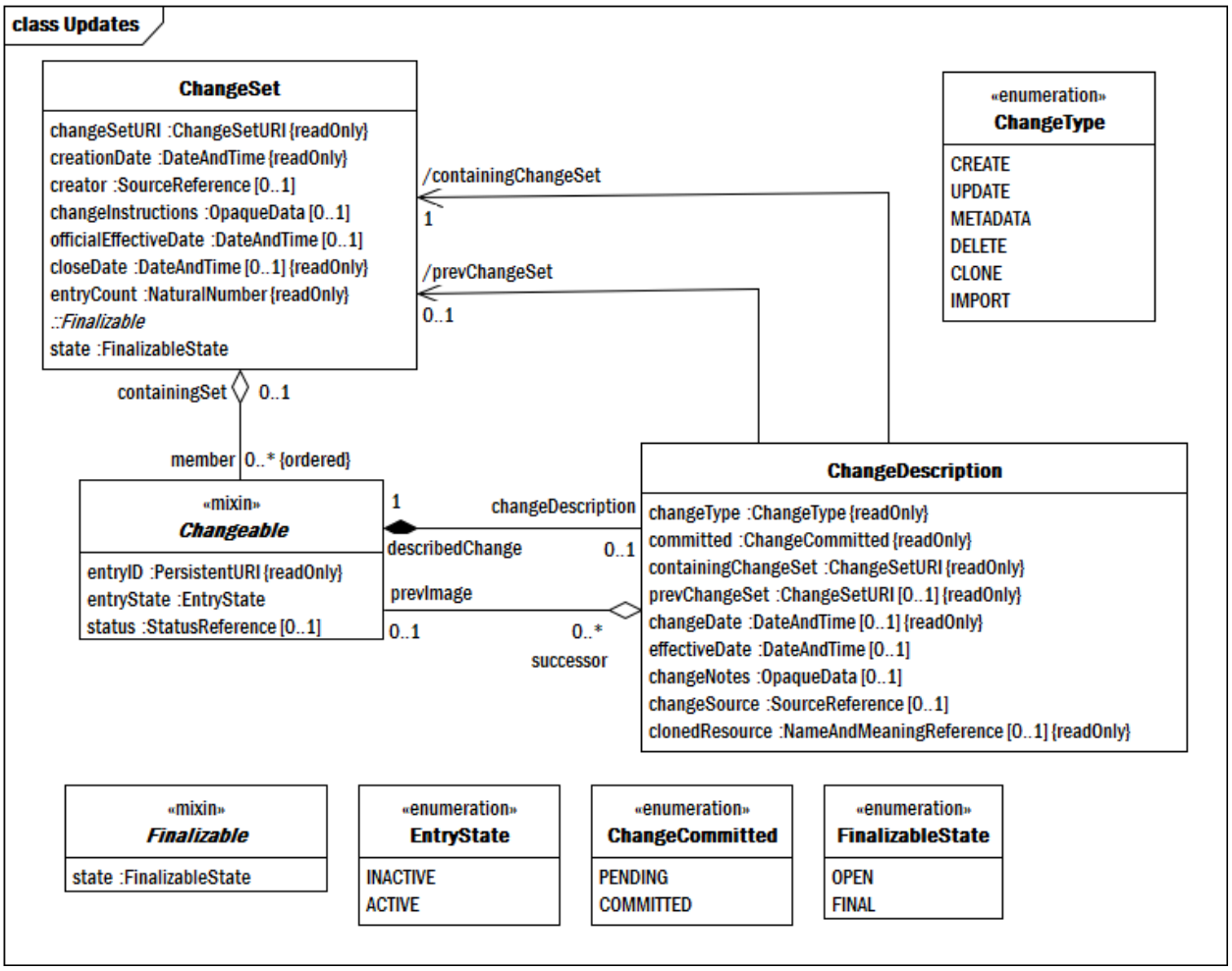


Figure 2.13: CTS_2 Change Model

The UPDATE model represents a `ChangeSet` - an ordered collection of updates that, when applied to a service, transform it from one consistent state to a second. `ChangeSets` can consist of as few as one `Changeable` element or can consist of a set of changes that could be used to construct the contents of an entire service from scratch. The key rule on `ChangeSets` is that, unless specifically overridden, they are considered “atomic" - either all of the (applicable) changes are applied or none of them are applied.

The `Changeable` mixin must be implemented in all compliant CTS_2 service implementations as it forms the core model of resource identity. `ChangeSet` and `ChangeDescription` need only be implemented in services that support the HISTORY , UPDATE or MAINTENANCE profiles.

Class ChangeDescription

The detailed description of what happened to a changeable resource. `ChangeDescription` will only be present in service profiles that support either the HISTORY or MAINTENANCE profiles.

Attributes:

- **changeType** - the type of change that occurred to cause the associated `Changeable` element to reach the state it is in

- **committed** - an indicator that states whether the change has been committed and is available to service consumers or is still pending while further authoring may occur. `committed` must always be `COMMITTED` in services that do not support the MAINTENANCE profile
- **containingChangeSet** - the URI of the `ChangeSet` that contains the described change
- **prevChangeSet** - the URI of the `ChangeSet` that contains the change that immediately preceded this change, if any
- **changeDate** - the date and time when this change was applied to the *local service instance*. `changeDate` is only present on committed changes.
- **effectiveDate** - the date and time that this change is (or was) scheduled to take effect in the *local service instance*. Typically this attribute is used to schedule a component to become active on a given date.

  A change having a future `effectiveDate` will not be visible to service calls whose reference time is earlier to this time. This attribute may only be present in committed changes. A compliant CTS_2 service implementation MUST not allow a change to be written with a `effectiveDate` that is earlier than `changeDate` . The purpose of this requirement is to prevent the “rewriting of history" - making a change effective in the past.
- **changeNotes** - a note, set of instructions and other information about the nature and purpose of this change
- **changeSource** - the person or organization responsible for this change
- **clonedResource** - the local identifier and URI of the resource that was cloned in this change if this is a `CLONE` operation.

Invariants:

1. There cannot be a previous change set on a CREATE operation. .
2. There cannot be a previous change set on a CREATE operation. .
3. Updates, deletes and metadata changes MUST have a previous image .
4. A `clonedResource` MUST be present if and only if `changeType` = `CLONE` .
5. Only versioned resources can be cloned .

Class ChangeSet

An ordered collection of changes that, when applied, will transform a service instance from one consistent state to another. A `ChangeSet` is viewed as an atomic unit of change - either all of the `Changeable` elements in a change set will be applied or none of them will be applied. It is anticipated that service implementations will provide a mechanism by which it can apply local business rules to the validation and application of change sets. These rules may include the option to selectively apply, reject, modify or ignore the elements of change sets as they arrive. In this case, it is up to the developers of the business rules to determine what constitutes a "complete" change set that can be applied.

Superclasses:

- Every instance of `ChangeSet` is also an instance of `Finalizable`.

Attributes:

- **changeSetURI** - a globally unique identifier that signifies this particular change set
- **creationDate** - the date and time that the change set was initially created
- **creator** - the person or organization who initially created the change set
- **changeInstructions** - documentation and instructions about the purpose and application of the change set

- **officialEffectiveDate** - the date and time that this set of changes became (or should become) effective *from the perspective of the authors*. This parameter enables history queries from both the perspective of the service ("What did the service return on July 1") and the perspective of the resource author ("What would the state of the terminology have been on July 1 had it been loaded prior to that date and not been locally modified?").
- **closeDate** - the date and time that this change set was finalized (`state` = `FINAL`). Once finalized, a change set cannot be further modified.
- **entryCount** - the number of `Changeable` members in the set
- **member** - A single atomic change. Members occur in the order in which they were / are to be applied

Invariants:

1. A `closeDate` is present if and only if the `state` is `FINAL` .
2. If any of the `Changeable` `members` have a `changeDescription` , all `members` MUST have a `changeDescription` .
3. Every `Changeable` `member` must be associated with a unique `changeDescription` .
4. If change descriptions are present, each change description must contain the URI if this change set in its `containingChangeSet` field. The change description state is `COMMITTED` exactly when the `changeSet` `state` is `FINAL` .
5. If a given `Changeable` element is changed more than once in the same `ChangeSet` , the second and following changes will reference the current change set URI in `prevChangeSet` .
6. `prevChangeSet` cannot be the same as `containingChangeSet` unless the an element is changed more than once .

Class Changeable

An element that can evolve over time. All `Changeable` elements must have identity.

Attributes:

- **entryID** - the globally unique identifier of the resource
- **entryState** - an indicator that states whether the `Changeable` element is `ACTIVE` , and subject to searching and browsing access or `INACTIVE` , meaning that it is only accessible if its identity is already known or if the service calls specifically state that they want to see inactive service elements
- **status** - the state of this model element in an externally defined workflow
- **changeDescription** - Detailed information about the last change that resulted in this changeable element being in the state that it is now. changeDescription is only present in services that support the HISTORY profile and then only when specifically requested.

Class Finalizable

The `Finalizable` mixin determines whether a resource version or a change set is still undergoing change (`OPEN`) or has reached its final state (`FINAL`).

Attributes:

- **state** - the state of the inheriting resource

Enum ChangeCommitted

the commitment state of an individual change

Attributes:

- **PENDING** - the specific change is part of an OPEN change set and has not yet been committed to the database. The change is only visible through queries that carry the `ChangeSetURI` of the open `ChangeSet`.
- **COMMITTED** - the specific change is a part of a `ChangeSet` that has reached a `FINAL` state, meaning that it is (or will be) considered an official part of the service content.

Enum ChangeType

An indicator that shows the type of change that occurred that transformed a `Changeable` element from its immediately previous state to the state that the `ChangeType` is associated with.

Attributes:

- **CREATE** - the element was newly created.
- **UPDATE** - one or more non-identifying attributes of the element changed. Note that `UPDATE` does not include changes to the `Changeable` attributes (`entryState` , `status` , and `owner`). These are considered to be `METADATA` changes.
- **METADATA** - a change occurred to the `entryState` or `status` attributes. `METADATA` changes are separated because they may often be service specific.
- **DELETE** - the changeable element was completely removed from the service and may no longer be retrieved by id *or* search. Service implementations may choose to disable and/or map `DELETE` operations to some other form, but this exists to allow the complete removal of errors and historically irrelevant information.
- **CLONE** - create the new versionable resource and create a (virtual) copy of all of the resource's dependents
- **IMPORT** - include the contents of an external resource version as a read-only part of the importing version

Enum EntryState

the current state of the `Changeable` element. Note that `entryState` represents the state of the element itself - *not* the state of a given change. The applicability of a given change is identified by its `effectiveDate`, not `entryState` .

Attributes:

- **INACTIVE** - the `ChangeAble` element is no longer considered to be an active component. The element may still be retrieved if its `entryID` is known, but it does not appear in search and browse operations unless specifically requested.
- **ACTIVE** - the `Changeable` element is considered to be an active member of the containing resource and may appear in any search, browse or query operations.

Enum FinalizableState

possible states of a `Finalizable` resource

Attributes:

- **OPEN** - the contents of a `ChangeSet` or `ResourceVersionDescription` may change, meaning that the contents cannot be determined by using the `ChangeSetURI` or `DocumentURI`.
- **FINAL** - the contents of a `ChangeSet` or `ResourceVersionDescription` cannot be changed, meaning that they can be unambiguously referenced via the corresponding URI.

Iteratable Change Sets

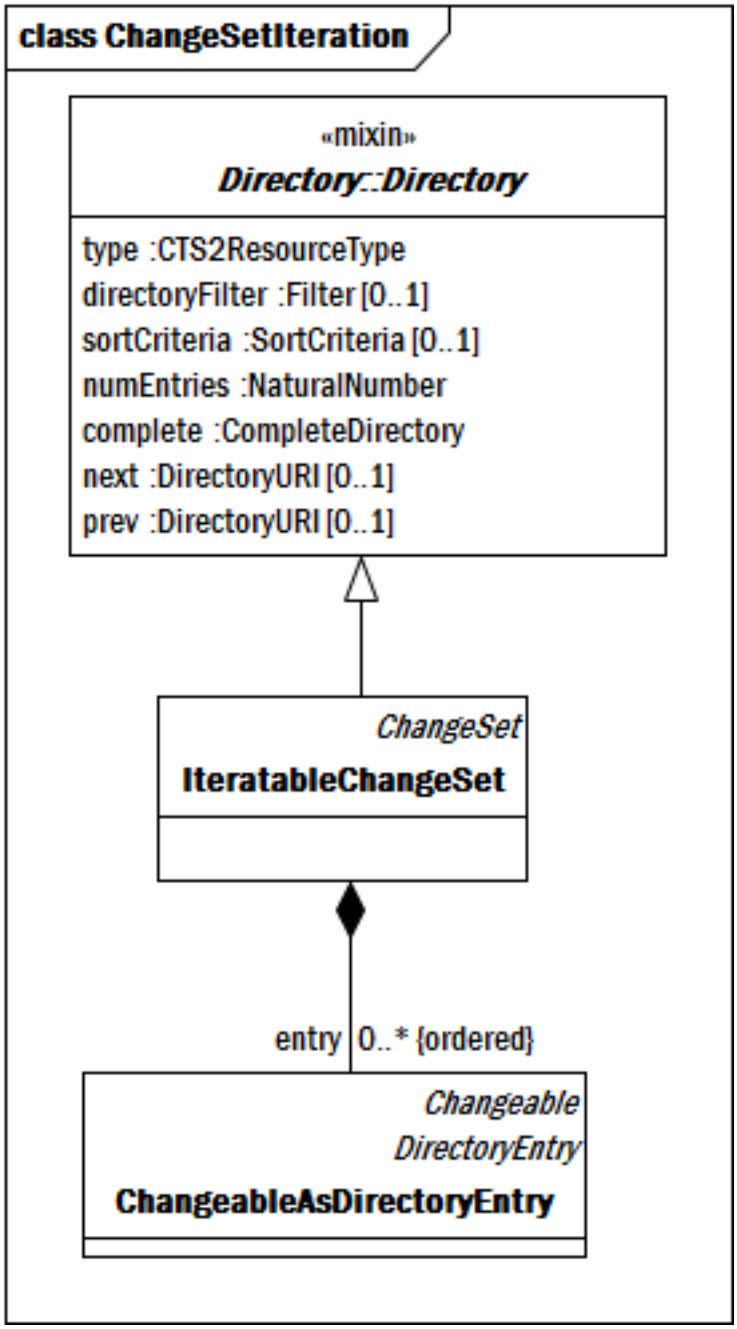


Figure 2.14: Iteratable Change Set

A `ChangeSet` can potentially grow to be quite large making it necessary to provide an version of a set that can be iterated for retrieval. The current version of the CTS_2 API does not specify any way to select contents from a change set resource - from the change set perspective, it is a complete unit and can only be viewed via retrieval and iteration.

Class ChangeableAsDirectoryEntry

A changeable element that occurs in an `IterableChangeSet`

Superclasses:

- Every instance of `ChangeableAsDirectoryEntry` is also an instance of `Changeable`.
- Every instance of `ChangeableAsDirectoryEntry` is also an instance of `DirectoryEntry`.

Class IteratableChangeSet

A change set whose contents is available as a set of directory entries that allows iteration.

Superclasses:

- Every instance of `IteratableChangeSet` is also an instance of `Directory`.
- Every instance of `IteratableChangeSet` is also an instance of `ChangeSet`.

Attributes:

- **entry** - An entry in an iteratable change set

Resource Description

Resource Description Model

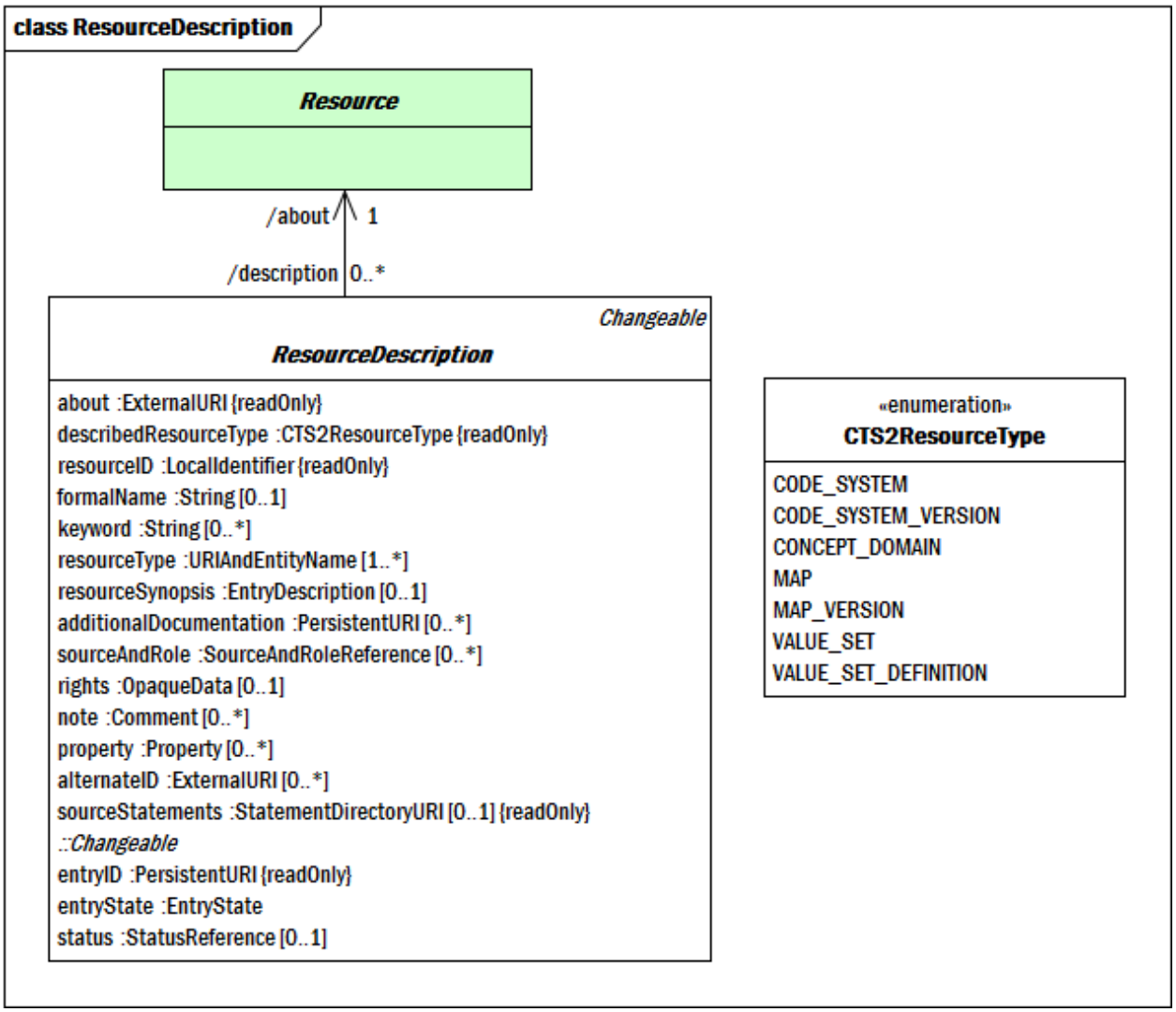


Figure 2.15: Resource Description

Class Resource

Anything that can have identity. Examples might include an electronic document, an image, a service (e.g. "today's weather report for Los Angeles"), and a collection of other resources. [24]

Class ResourceDescription

`ResourceDescription` represents the shared characteristics common to both abstract and resource version descriptions. `ResourceDescription` is an abstract type and, as such, cannot be directly created. Resource descriptions are `Changeable` , meaning that they have identity and can be created, updated and deleted.

Superclasses:

- Every instance of `ResourceDescription` is also an instance of `Changeable`.

[24] http://tools.ietf.org/html/rfc2396

Attributes:

- **about** - the (or a) definitive URI that represents the resource being described. Note that this is NOT the URI of the resource description in the CTS2 format, but of the resource itself. As an example, the `about` URI for the Wine ontology would be "http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#". The NCI Thesaurus [25] has, amongst others, the `about` URI of http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#. HL7 uses ISO Object Identifiers (OIDs) to label resources so, from the HL7 perspective, the `about` URI of the NCI Thesaurus would be "urn:oid:2.16.840.1.113883.3.26.1.1"
- **describedResourceType** - the type of resource being defined. From the perspective of the CTS_2 specification, these types are limited to those specified in `CTS2ResourceType` .
- **resourceID** - a local identifier that uniquely names the resource within the context of the `describedResourceType` and implementing service. As an example, this might be "SCT" for the SNOMED-CT code system, "SCT-2010AA" for a SNOMED-CT code system version.
- **formalName** - the formal or officially assigned name of this resource, if any
- **keyword** - additional identifiers that are used to index and locate the resource
- **resourceType** - the class(es) that this resource instantiates
- **resourceSynopsis** - a textual summary of the resource - what it is, what it is for, etc.
- **additionalDocumentation** - a reference to a document that provide additional information about the resource
- **sourceAndRole** - a reference to an individual, organization of bibliographic reference that participated in the creation, validation, review, dissemination of this resource and the role(s) they played
- **rights** - copyright and IP information. Note that `rights` applies to the source resource, not the CTS_2 rendering.
- **note** - an additional note or comment about the resource
- **property** - additional information about the resource that does not fit into any of the attributes described above
- **alternateID** - an alternative identifier that uniquely names this resource in other environments as contexts. As an example, if a resource had both an ISO Object Identifier and a DNS name, the DNS name might be assigned as the `entryID` of the resource by one service while the ISO OID would be recorded as an `alternateURI` using the "urn:oid" prefix. Note that `alternateId` s can be added or removed during resource updates.
- **sourceStatements** - a `DirectoryURI` that references the set of statements that were used to construct the containing resource. This attribute must (may?) be present if and only if the service supports the STATEMENT profile

Enum CTS2ResourceType

the resource types that can be described in a CTS_2 service

Attributes:

- **CODE_SYSTEM** - an ontology, classification scheme, thesaurus or code system
- **CODE_SYSTEM_VERSION** - a specific release or version of an ontology, classification scheme, thesaurus or code system.
- **CONCEPT_DOMAIN** - a "data element concept" - a specification of the conceptual domain of an element in a database, form, message, etc.
- **MAP** - a set of rules for transforming a set of entity references into a second set of entity references, drawn from a different code system or value set.
- **MAP_VERSION** - a specific version or release of a MAP

[25] http://ncit.nci.nih.gov/ncitbrowser/

- **VALUE_SET** - a set of entity references
- **VALUE_SET_DEFINITION** - a set of rules for establishing which entity references belong to a value set at a given point in time

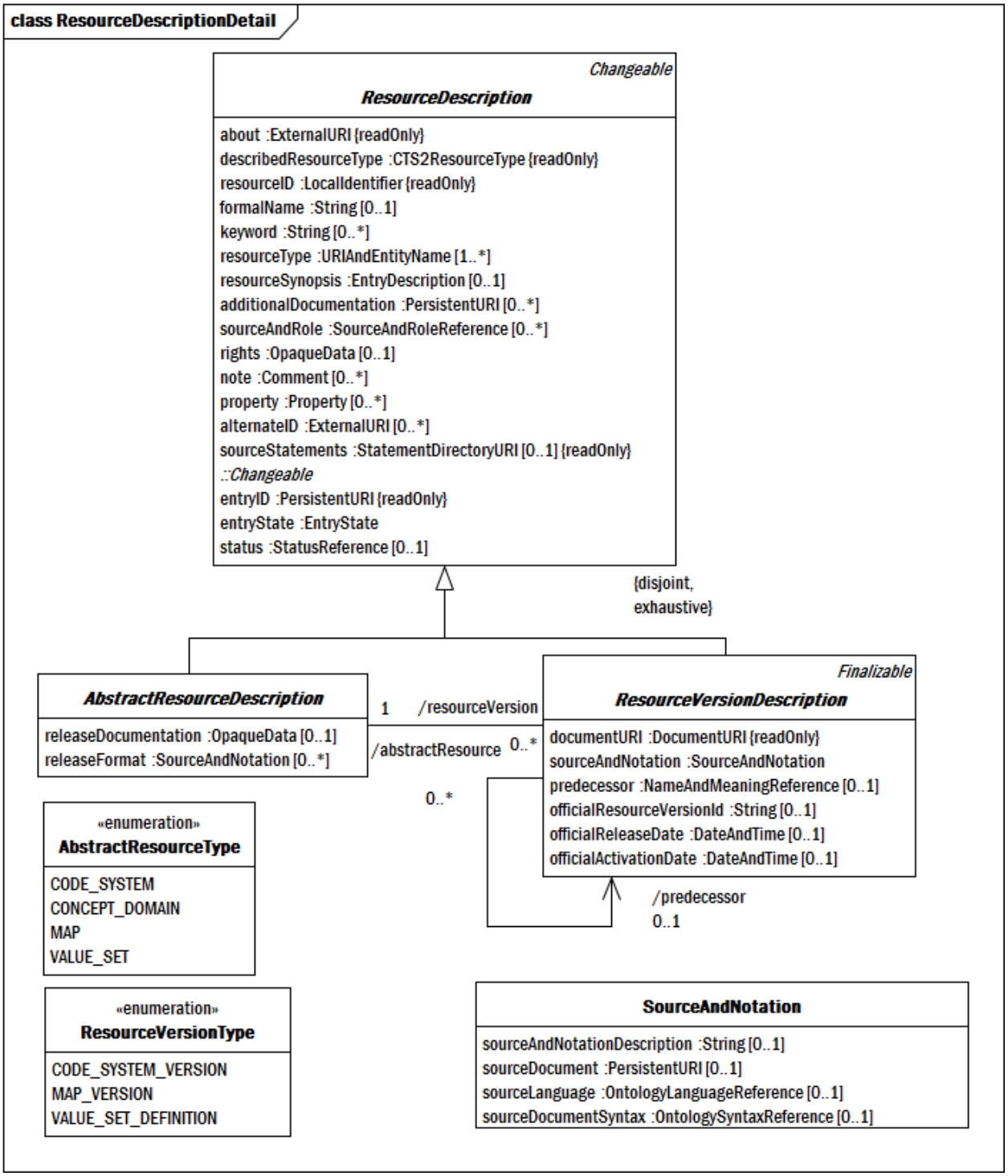


Figure 2.16: Detailed Resource Description

Class AbstractResourceDescription

the description of the characteristics of a resource that are independent of the resource content

Superclasses:

- Every instance of `AbstractResourceDescription` is also an instance of `ResourceDescription`.

Attributes:

- **releaseDocumentation** - documentation about the frequency and nature of releases (version) of this resource.[26]
- **releaseFormat** - a format and notation that the releases (versions) of this resource are published in

Invariants:

1. An abstract resource description must be one of `CodeSystem` , `ConceptDomain` , `ValueSet` , or `Map` .
2. The unique id of an abstract resource is the `about` URI .

Class ResourceVersionDescription

information about the source, format, release date, version identifier, etc. of a specific version of an abstract resource

Superclasses:

- Every instance of `ResourceVersionDescription` is also an instance of `ResourceDescription`.
- Every instance of `ResourceVersionDescription` is also an instance of `Finalizable`.

Attributes:

- **documentURI** - a URI that identifies the specific version, language and notation of the `about` resource. This URI needs to be constructed in such a way that, if necessary, it will be possible to differentiate resource versions that were loaded from different document syntaxes. As an example, if an image of a the wine ontology was loaded from a resource that was in Manchester Syntax, it should be given a different URI than the image loaded from the RDF/XML syntax. The reasoning behind this is, even in cases where different syntaxes are 100
- **sourceAndNotation** - a description of where the (or a) source of the version may be found, what format and language it is available in, etc.
- **predecessor** - a reference to the name and URI version of the resource from which this current version is derived - the version of the resource that immediately preceded it
- **officialResourceVersionId** - an official label or identifier that was assigned to this version by its publisher
- **officialReleaseDate** - the date that this version of the resource officially became available
- **officialActivationDate** - the date that this version of the resource is stated by its publishers to go into effect

Invariants:

1. A resource version description must be one of `CodeSystem` , `ConceptDomain` , `MapVersion` , or `ValueSetDefinition` .
2. The `documentURI` is the unique identifier for a `ResourceVersionDescription`. .

[26] OMV 2.4.1 pp 18

Class SourceAndNotation

A description of the source from which the `ResourceVersionDescription` was derived. When possible, `SourceAndNotation` should include a reference to the actual source document from which it was derived. As an example, if the resource was derived from the W3C Wine Ontology, the URI http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine would unambiguously name the document. In the cases, however, when a definitive source document is not available, a textual description should be provided, instead, in `sourceAndNotationDescription` . Where possible, the ontology language and ontology syntax should also be provided. In the case of the Wine Ontology above, the ontology language would be http://www.w3.org/2002/07/owl# (OWL) and the syntax would be application/rdf+xml.

Attributes:

- **sourceAndNotationDescription** - a textual description of where the specified resource version was derived from. This parameter must be supplied if a reasonable `PersistentURI` for the source document is not available.
- **sourceDocument** - a persistent URI that references the document from which the resource version was derived. This URI may be resolvable to a digital resource or may be the name of a book, publication or other external document.
- **sourceLanguage** - the formal language, if any, that the source for the resource version is expressed in. Examples include Common Logic, OWL, OWL-DL, CLAML [27], etc.
- **sourceDocumentSyntax** - the syntax of the source of the resource version, if known. Examples might include rdf/xml, Turtle, Manchester Syntax, CSV, etc.

Invariants:

1. `SourceAndNotation` must supply either a textual description of the source in `sourceAndNotationDescription` or the URI of the document from which the resource was derived in `sourceDocument` .

[27] http://esearch.cen.eu/Details.aspx?id=4244858

Directories of Resource Descriptions

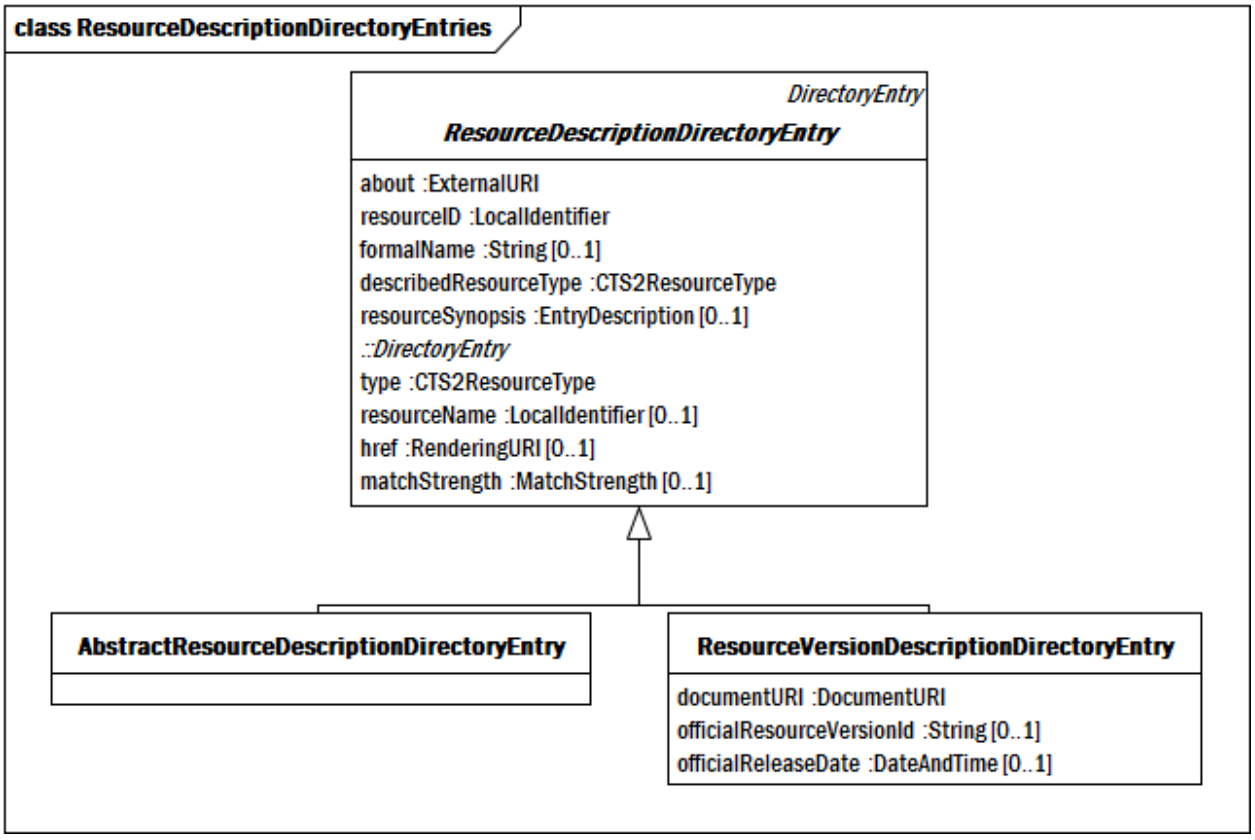


Figure 2.17: Directory Mixin for Resource Version Description

Class AbstractResourceDescriptionDirectoryEntry

a summary of an abstract resource description

Superclasses:

- Every instance of `AbstractResourceDescriptionDirectoryEntry` is also an instance of `ResourceDescriptionDirectoryEntry`.

Class ResourceDescriptionDirectoryEntry

A `DirectoryEntry` that identifies the elements of a resource description that appear in every directory of resources of a particular type. `ResourceDescriptionDirectoryEntry` is an abstract type and is realized in the implementing subtypes listed in `CTS2ResourceType`

Superclasses:

- Every instance of `ResourceDescriptionDirectoryEntry` is also an instance of `DirectoryEntry`.

Attributes:

- **about** - the (or a) definitive URI that represents the resource being described.
- **resourceID** - a local identifier that uniquely names the resource within the context of the `describedResourceType` and implementing service

- **formalName** - the formal or officially assigned name of this resource
- **describedResourceType** - type of resource represented in the entry
- **resourceSynopsis** - a textual summary of the resource - what it is, what it is for, etc.

Class ResourceVersionDescriptionDirectoryEntry

A summary of a resource version.

Superclasses:

- Every instance of `ResourceVersionDescriptionDirectoryEntry` is also an instance of `ResourceDescriptionDirectoryEntry`.

Attributes:

- **documentURI** - a URI that identifies the specific version, language and notation of the `about` resource
- **officialResourceVersionId** - a label or identifier that was assigned to this version by its publisher
- **officialReleaseDate** - information about the source, format, release date, version identifier, etc. of a specific version of an abstract resource

Chapter 3

Core Computational Model Elements

This section of the specification describes the interface characteristics common to one or more CTS_2 service specifications. It defines the attributes and methods common to all services and the attributes and methods common to each individual functional profile. It also defines the various parameters that are used in the interface specification and an exception model. The functional profiles that do not have any structural dependencies (**IMPORT**, **EXPORT**, and **TEMPORAL**) are also included in this section.

Interface Specific Structures

Miscellaneous Input Structures

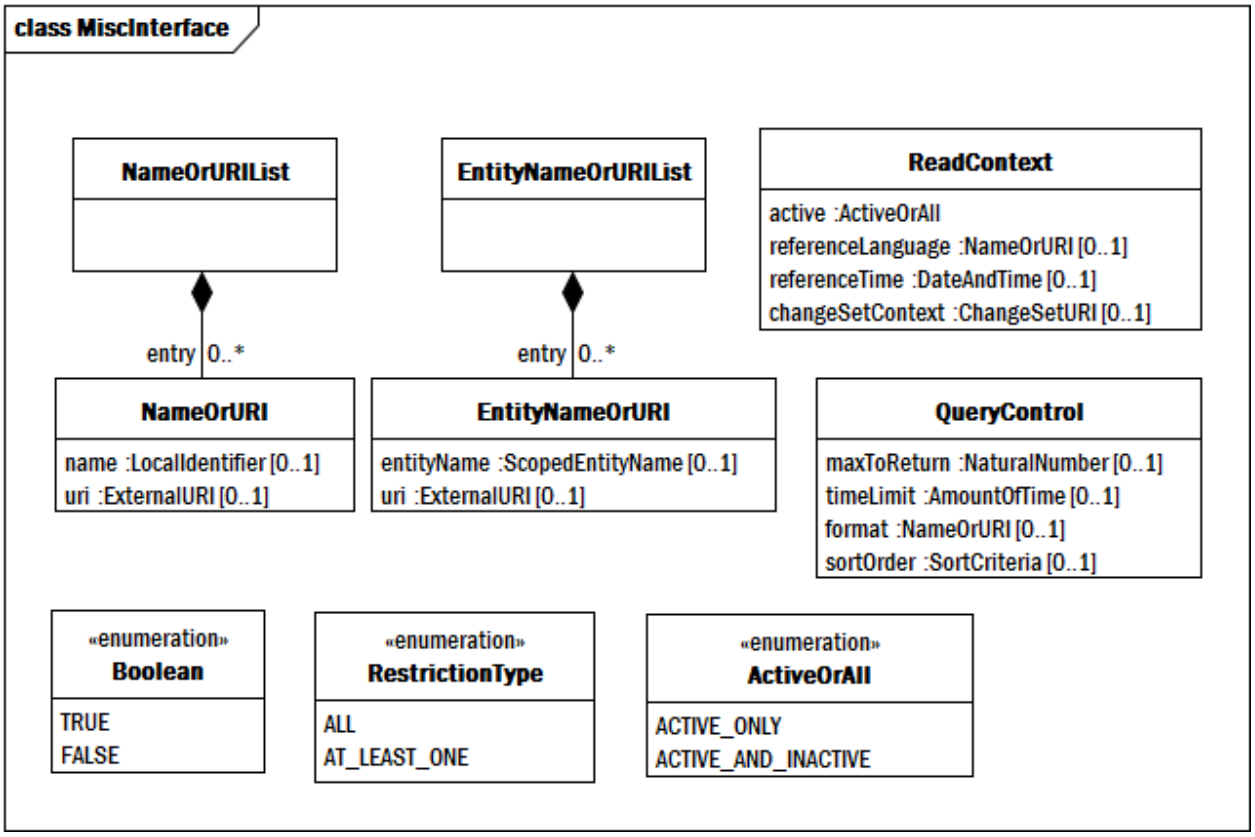


Figure 3.1: Interface Building Blocks

We begin this section by describing a set of interface elements that are shared among some or all of the CTS_2 service implementations.

Class EntityNameOrURI

A reference to a class, property or individual that is described in some Code System. `EntityNameOrURI` may either reference an entity that is known locally to the service or an entity that is described elsewhere. If the entity is known to the service, it is possible to use the `entityName` variant, but note that neither `scopingNamespace` nor the `entityName` are guaranteed to be the same in different CTS_2 implementations. The `entityName` variant is intended for use in human/browser interactions and should not be hard-coded into data tables or applications.

Attributes:

- **entityName** - a combination of a namespace identifier and a local name that, together, uniquely references an entity known to the service
- **uri** - an `ExternalURI` that references a class, property or individual

Invariants:

1. Either an entityName or a uri must be present but not both .

Class EntityNameOrURIList

A set of zero or more `EntityNmeOrURI` elements

Attributes:

- **entry** - an element in a list of entity identifiers in the form of either a name or a URI

Class NameOrURI

Carries either a local identifier (`name`) or a URI (`uri`) that references a resource in the service. `NameOrURI` is only used as an input parameter and its type is always defined by the usage context. Note that service calls that use the `name` option may not be portable across implementations, as there is no guarantee that any two CTS_2 service implementations will use the same local identifiers for the same resources.

Attributes:

- **name** - a `LocalIdentifier` that references a unique resource within the context of the service implementation and type of resource being accessed
- **uri** - an `ExternalURI` that references a unique resource within the context of the resource type

Invariants:

1. Either a name or a uri must be present but not both .

Class NameOrURIList

A set of zero or more `NameOrURI` elements

Attributes:

- **entry** - an entry in list of local resource names or global URIs

Class QueryControl

Parameters that control the return format, number of elements and amount of time that can be taken for a query to complete. If omitted, the service implementation may determine the default setting for all of the parameters.

Attributes:

- **maxToReturn** - the maximum number of `entries` that may be present in a return `Directory` . Note that a service may choose to return less than `maxToReturn` entries - this is simply an upper limit. If `maxToReturn` is not supplied, the service may use whatever return block size it determines to be most appropriate.
- **timeLimit** - the maximum amount of time that may be taken to process a query before a time out exception occurs. If not present, the service determines the maximum query time out.
- **format** - the local identifier or URI of the return format for query results. This parameter defaults to the `defaultFormat` in the `BaseService` interface if not supplied.
- **sortOrder** - the the sort order of the returned query if its output is a `Directory`

Class ReadContext

The context that controls the behavior of a read or query. The `ReadContext` parameter should be present in any *read* or *query* method call that returns data. `ReadContext` is always optional and, if not supplied the defaults are: `active` = `ACTIVE_ONLY` , `referenceLanguage` = (default language for the service or user), `referenceTime` = `now` , `changeSetContext` = (none).

Attributes:

- **active** - determines whether the query only applies to only active or all entries.
- **referenceLanguage** - the spoken or written language that should be used for the results of the inquiry, where appropriate. Should default to the default reference language of the service if omitted.
- **referenceTime** - the contextual date and time of the query. `referenceTime` is may only be present in services that support the TEMPORAL profile. If omitted, `referenceTime` defaults to the current system date and time.
- **changeSetContext** - the URI of an open change set whose contents should be included in the results of the access request. `changeSetContext` is only applicable in services that support the AUTHORING profile. If omitted, no change set context is supplied in the service call.

Enum ActiveOrAll

an indicator that determines whether the given service access request applies only to elements that are currently marked as `ACTIVE` in the context of the particular query or to both `ACTIVE` and `INACTIVE` entries.

Attributes:

- **ACTIVE_ONLY** - the inquiry only applies to `ACTIVE` entries
- **ACTIVE_AND_INACTIVE** - the inquiry applies to both `ACTIVE` and `INACTIVE` entries

Enum Boolean

a return type that indicates a positive or negative answer to the specific question that was posed.

Attributes:

- **TRUE** - represents an affirmative answer to an inquiry
- **FALSE** - represents a negative answer to an inquiry

Enum RestrictionType

a parameter used in queries where multiple elements are provided. It determines whether a candidate element must satisfy all restrictions or just one or more restriction in order to be considered as satisfying the restriction composite

Attributes:

- **ALL** - the candidate passes only if all of the elements are present
- **AT_LEAST_ONE** - the candidate passes if any of the supplied elements are present

Exception Model

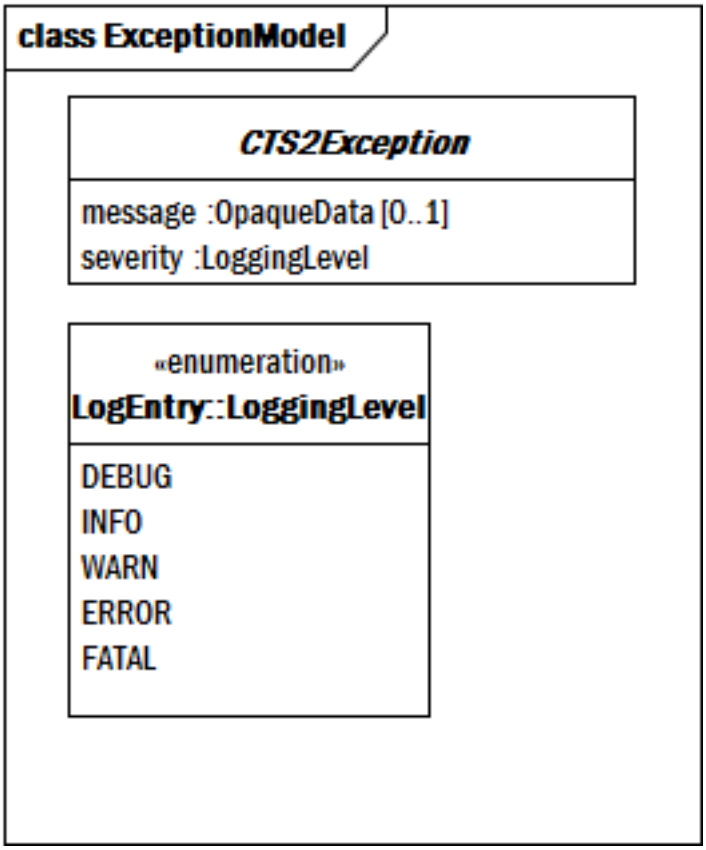


Figure 3.2: Exception Model

`CTS2Exception` represents the minimum exception element that MUST be returned by CTS_2 service implementations. It includes a general slot for a message as well as a `severity` indicator. The determination of the relative severity of individual exceptions has been left to the service implementer, which means that, at the moment, one service implementation may treat an error as FATAL and refuse to proceed while a second may choose to treat it simply as a warning. Obviously this is less than ideal and we anticipate that a future release of this specification will categorize errors more strictly.

Class CTS2Exception

An exception generated by the CTS 2 service.

Attributes:

- **message** - An structured or unstructured message detailing the cause and reason for the exception.
- **severity** - An indicator of the relative severity of the particular exception.

Enum LoggingLevel

The `LoggingLevel` entries are modeled after the corresponding levels in the Apache log4j package [1]. As with the log4j package, each level includes the entries in the lower level. In particular, levels are ordered. For the standard levels, we have DEBUG < INFO < WARN < ERROR < FATAL.

Attributes:

- **DEBUG** - detailed (verbose) information about the process or operation that allows a user to determine exactly what transpired
- **INFO** - informative messages about the progress of or results of a requested operation
- **WARN** - represents potential problems that do will not prevent completion of the requested operation, but may require attention or intervention.
- **ERROR** - represents an error that, while serious, may under circumstances be ignored and processing may continue
- **FATAL** - represents an error that prevents further processing. FATAL errors cannot be ignored or overridden.

Service Base

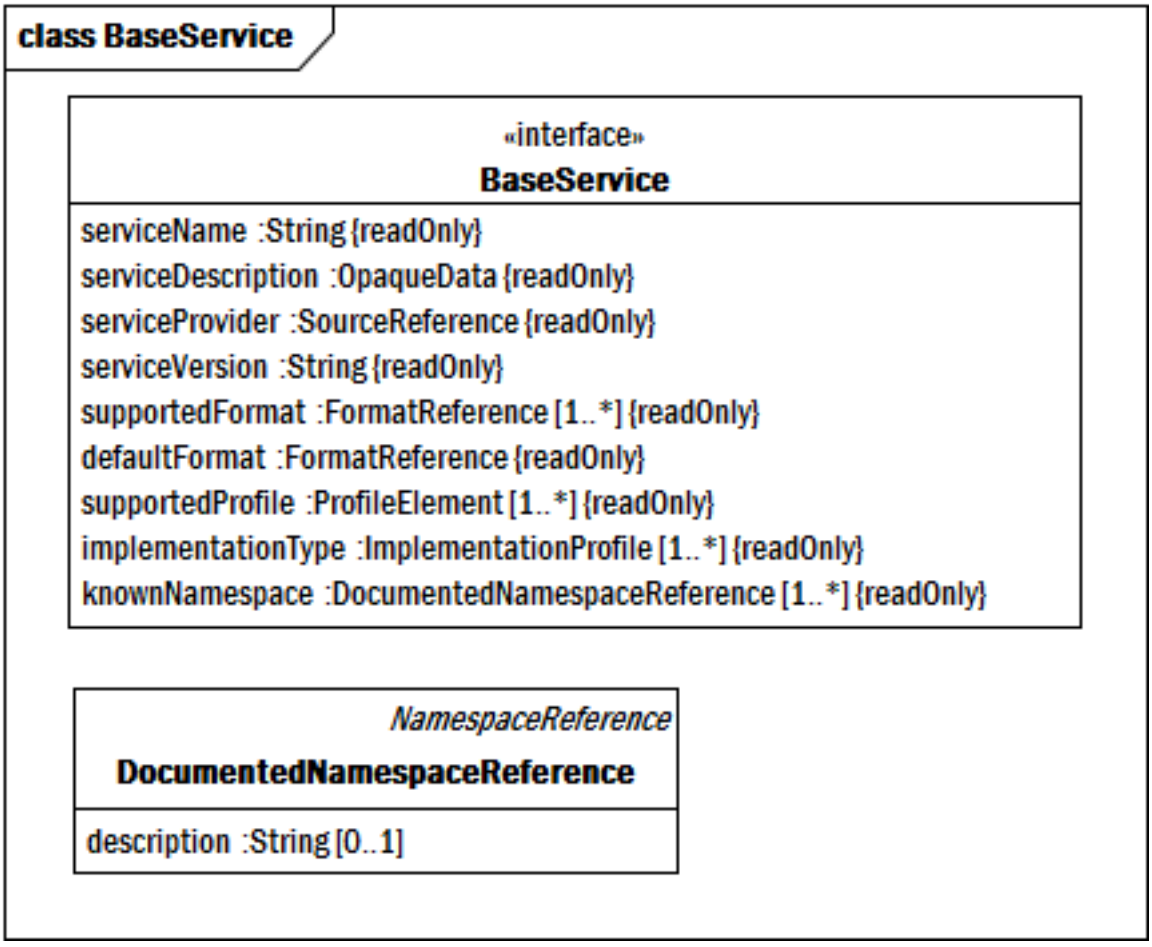


Figure 3.3: Service Base

[1] http://logging.apache.org/log4j/

This diagram shows the set of elements and attributes that must be present at the service root of any CTS_2 service implementation. It provides basic metadata about the service itself as well as information about what formats, profiles and platform specific implementations the service supports. It also lists all of the local namespace identifiers and namespace URIs known to the service implementation.

All of the elements in the `BaseService` resource are *readOnly*, meaning that they are established and updated by the service implementation and cannot be changed via a CTS_2 function call.

Note that CTS_2 namespace identifiers are service wide. As opposed to many ontology editors where a namespace such as XML Schema may be identified with the local name "xs:" in one schema and "xsd:" in a second, CTS_2 requires that namespaces be consistent across the service level.

Class BaseService

`BaseService` contains the components that are common to any CTS_2 service implementation. It includes information about the service itself, the namespaces and formats that are known to the service and the structural, functional and representation profiles that are supported by the service instance.

Attributes:

- **serviceName** - a short name that identifies the particular service and implementation
- **serviceDescription** - a description of the service, its use, etc.
- **serviceProvider** - a reference to the individual or organization that provides the service.
- **serviceVersion** - the version or release identifier of the service
- **supportedFormat** - alist of the representation formats supported by the service. Example: text/html, text/xml, application/json
- **defaultFormat** - the default format used by the service unless otherwise specified
- **supportedProfile** - the set of service profiles that are supported by this service implementation
- **implementationType** - the particular implementation type(s) supported by this profile
- **knownNamespace** - the set of namespaces recognized by this service. Because many namespace identifiers tend to be cryptic (i.e. HL7 OIDs, BioPortal URL's, etc.), `knownNamespace` includes the namespace name, an optional URI *and* a place to provide textual detail describing what the namespace references. Note that namespace names must be unique across an entire CTS_2 implementation - the same namespace name cannot represent one namespace in code system A and a second in code system B. Note also that namespace names are *local* to a service instance. Information that is communicated between service instances, recorded in data tables or client software *must* use full URIs.

Invariants:

1. Every `supportedProfile` entry describes a unique structural profile .
2. Every supported format name must be unique .
3. The default format is one of the supported formats .
4. Every known namespace must be unique .

Class DocumentedNamespaceReference

A namespace reference that may include additional documentation about the scope of the namespace.

Superclasses:

- Every instance of `DocumentedNamespaceReference` is also an instance of `NamespaceReference`.

Attributes:

- **description** - documentation about the scope and origin of the namespace

Functional Profiles

Read Service Base

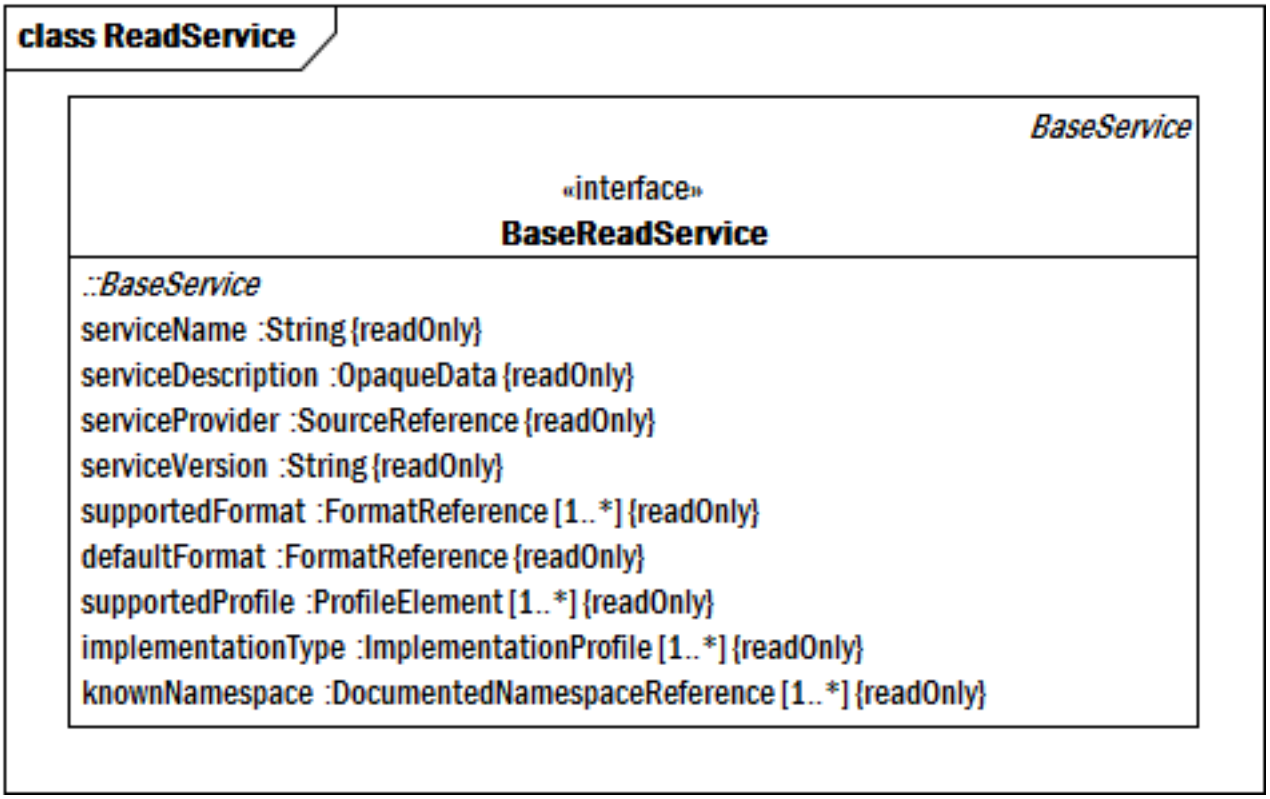


Figure 3.4: Read Service

Class BaseReadService

The common metadata about a CTS_2 service instance that is shared among all service instances. The `BaseReadService` interface does not add any additional information beyond that supplied by `BaseService`.

Superclasses:

- Every instance of `BaseReadService` is also an instance of `BaseService`.

Query Service Base

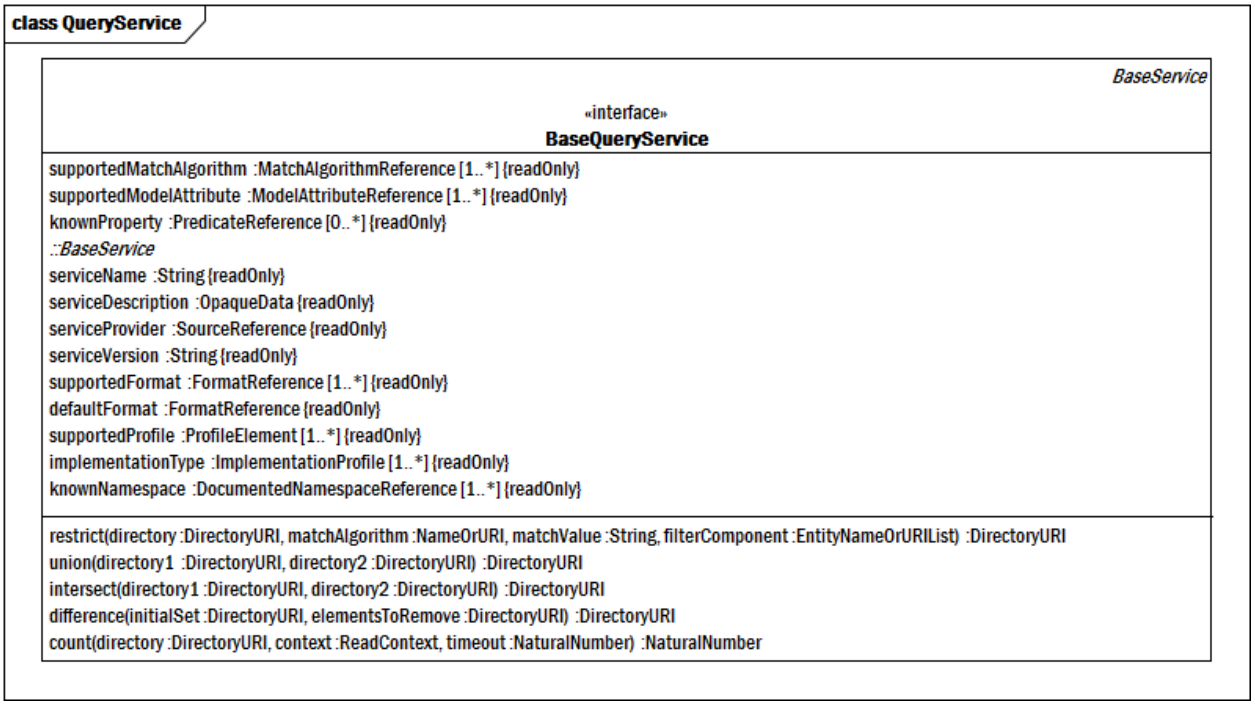


Figure 3.5: Common Query Service Components

The `QueryService` provides search and filtering access to the `Changeable` components of the corresponding structural profile. `QueryServices` constrain directories of the appropriate type. Each structural profile extends the query service to: (a) provide a starting directory handle (`DirectoryURI`) of the appropriate type and (b) provide resolution services to resolve directory URI's into Directories of the appropriate type and (c) provide additional filters where appropriate.

Class BaseQueryService

`BaseQueryService` represents the set of attributes and operations that are common across all query service implementations. It includes generic directory manipulation functions as well as enumerations of the permissible values for query parameters in the given service context.

Superclasses:

- Every instance of `BaseQueryService` is also an instance of `BaseService`.

Attributes:

- **supportedMatchAlgorithm** - the match algorithms that can be used in filters for this service instance
- **supportedModelAttribute** - the set of model attributes that can be referenced in filter instances for the given service implementation
- **knownProperty** - The set of properties that are used in one or more instances of the resource represented by this service. This list includes all properties that can be used in queries in this service, independent of the `entryState` or temporal state of the resource(s) being searched.

Operation: restrict

Return a `DirectoryURI` that references the set of all elements represented by `directory` that match the criteria specified in `filter`.

Input Parameters:

- **directory** - a URI that references a homogeneous set of resources (Type: `DirectoryURI`)
- **matchAlgorithm**$_{OPT}$ - The name or URI of the match algorithm to use when selecting values. The default value if the parameter isn't supplied is "CONTAINS" - the supplied match value appears anywhere in the target. (Type: `NameOrURI`)
- **matchValue**$_{OPT}$ - The value to be used in comparison. The structure and format of `matchValue` depends on the specific `matchAlgorithm` . As an example, a "startsWith" algorithm would be plain text, a "regularExpression" algorithm would ahve a regular expression, while an "exists" altorithm would have nothing in the `matchValue` argument. (Type: `String`)
- **filterComponent**$_{OPT}$ - The name or URI of a property or model element to be filtered. If omitted, all properties are searched. (Type: `EntityNameOrURIList`)

Return Type: `DirectoryURI`

Exceptions:

- **InvalidDirectoryURI** - The supplied `directory` URI is not valid
- **InvalidDirectoryType** - The type represented by the supplied `directory` URI is not the one required by the service invocation.
- **UnsupportedMatchAlgorithm** - The `matchAlgorithm` is not supported by the service
- **UnsupportedPredicate** - The predicate name or URI is not recognized by the service
- **UnsupportedNamespaceName** - The supplied namespace name is not one that is known to the service.
- **UnsupportedModelAttribute** - The name or URI of a CTS_2 model attribute is not recognized and/or supported by the service implementation

Operation: union

Return a directory that represents the combination of the contents of the supplied two supplied directories, with duplicate entries removed. Note that both of the directory URIs must resolve to entries of the same type.

Input Parameters:

- **directory1** - the first of the two directories to be combined (Type: `DirectoryURI`)
- **directory2** - the second of the two directories to be combined (Type: `DirectoryURI`)

Return Type: `DirectoryURI`

Exceptions:

- **InvalidDirectoryURI** - The supplied `directory` URI is not valid
- **InvalidDirectoryType** - The type represented by the supplied `directory` URI is not the one required by the service invocation.

Operation: intersect

Return a directory that represents the set of elements that are common to the two supplied directories. Note that both DirectoryURIs must represent resources of the same type.

Input Parameters:

- **directory1** - the first directory to be intersected (Type: `DirectoryURI`)
- **directory2** - the second directory to be intersected (Type: `DirectoryURI`)

Return Type: `DirectoryURI`

Exceptions:

- **InvalidDirectoryURI** - The supplied `directory` URI is not valid
- **InvalidDirectoryType** - The type represented by the supplied `directory` URI is not the one required by the service invocation.

Operation: difference

Remove any of the elements in `elementsToRemove` from `initialSet` if present. Elements in `elementsToRemove` that are not in `initialSet` are ignored.

Input Parameters:

- **initialSet** - the initial directory of elements (Type: `DirectoryURI`)
- **elementsToRemove** - thy set of elements to be removed from `initialSet` if present (Type: `DirectoryURI`)

Return Type: `DirectoryURI`

Exceptions:

- **InvalidDirectoryURI** - The supplied `directory` URI is not valid
- **InvalidDirectoryType** - The type represented by the supplied `directory` URI is not the one required by the service invocation.

Operation: count

Return the number of elements currently represented by the supplied directory URI.

Input Parameters:

- **directory** - a reference to the set of elements (Type: `DirectoryURI`)
- **context**$_{OPT}$ - the context of the count - whether active or all elements are to be counted and the temporal reference point for the count (Type: `ReadContext`)
- **timeout**$_{OPT}$ - the maximum amount of time allowed to do the count (Type: `NaturalNumber`)

Return Type: `NaturalNumber`

Exceptions:

- **QueryTimeout** - The `timeLimit` was exceeded by the service.
- **InvalidDirectoryURI** - The supplied `directory` URI is not valid
- **InvalidDirectoryType** - The type represented by the supplied `directory` URI is not the one required by the service invocation.

History Service Base

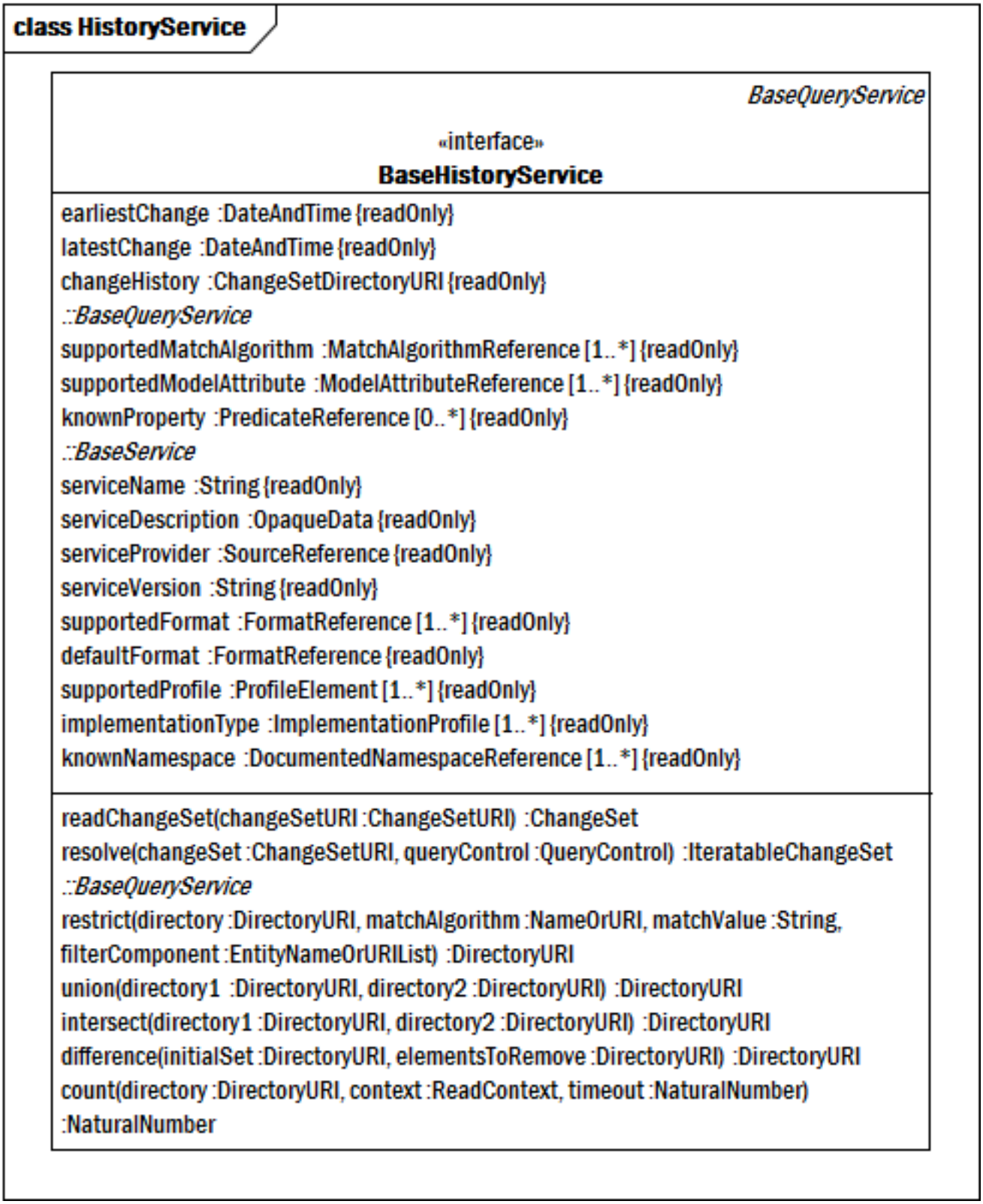


Figure 3.6: History Service

`HistoryService` documents the changes that have been applied to a service over time. It documents when the first change occurred, when the last change occurred and provides access to a set of changes in the order that they were applied.

Class BaseHistoryService

The `BaseHistoryService` provides access to ChangeSets and the history of what happened to individual elements.

Superclasses:

- Every instance of `BaseHistoryService` is also an instance of `BaseQueryService`.

Attributes:

- **earliestChange** - the date and time of the first change that occurred in the service. Note that the granularity of "change" is the application of an import operation or the submission of a change set, so the first change might encompass the loading of an entire ontology or even a complete ontology with history.
- **latestChange** - the date and time that the last change was applied to the service.
- **changeHistory** - a `DirectoryURI` that resolves to the set of all changes that have been applied to the service

Operation: readChangeSet

Return the change set referenced by the supplied URI.

Input Parameters:

- **changeSetURI** - the URI of the change set to be returned (Type: `ChangeSetURI`)

Return Type: `ChangeSet`

Exceptions:

- **UnknownChangeSet** - The change set specified could either not be read or located by the service.

Operation: resolve

Retrieve a change set as an iterable resource - a handy function for viewing large change sets

Input Parameters:

- **changeSet** - the URI of the change set to retrieve (Type: `ChangeSetURI`)
- **queryControl** - parameters the effect the iteration and ordering of the return values (Type: `QueryControl`)

Return Type: `IteratableChangeSet`

Exceptions:

- **UnknownChangeSet** - The change set specified could either not be read or located by the service.
- **UnsupportedMatchAlgorithm** - The `matchAlgorithm` is not supported by the service
- **UnsupportedPredicate** - The predicate name or URI is not recognized by the service
- **UnsupportedNamespaceName** - The supplied namespace name is not one that is known to the service.

- **UnsupportedModelAttribute** - The name or URI of a CTS_2 model attribute is not recognized and/or supported by the service implementation
- **UnsupportedFormat** - The `format` is not supported by the service implementation
- **QueryTimeout** - The `timeLimit` was exceeded by the service.

Maintenance Service Base

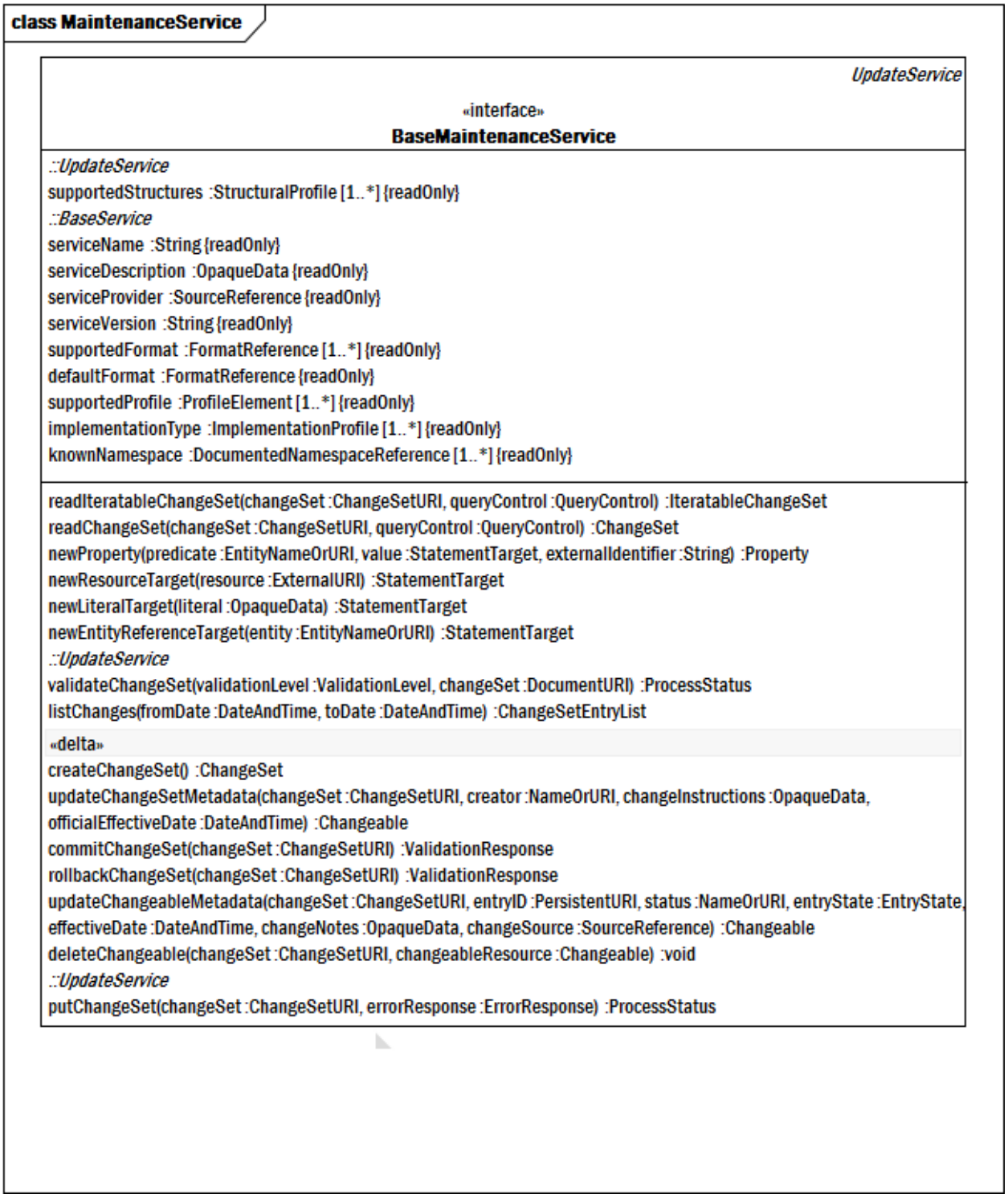


Figure 3.7: Maintenance Service

An maintenance service is an extension of an update service that allows change sets to be created through a sequence of create, update and/or delete method calls. An maintenance interaction always begins with a `createChangeSet` call, which returns the URI of a new, `OPEN ChangeSet` . This URI must be supplied in all subsequent create, update and/or delete method invocations. Each successful method call becomes a member of the open change set. These changes will not appear in the

state of the running service until the `commitChangeSet` method is successfully invoked. The impact of the changes can be examined, however, by including the open `ChangeSet` URI in the `ReadContext` of any method that supports the `readContext` parameter. When included, the service will respond as if the referenced `ChangeSet` had been committed.

An open `ChangeSet` can also be rolled back, which has the effect of removing all of the uncommitted changes as well of the change itself from the service. Once a `ChangeSet` is successfully committed, its `state` is changed to `FINAL` , and no further changes may be added to it. The current CTS_2 specification provides no mechanism to allow committed `ChangeSets` to be rolled back. Clever service extensions, however, should be able to devise an "inverse change set" that, when applied, undoes all of the changes in an existing change set.

Note that a change set contains both the "before" image (if any) of any change that is being applied as well as the URI of the last change set that contained a change that effected the named resource. This allows the service to detect incompatible changes - if the before image of any resource does not match the before image recorded in the change set the service should flag this as an error and not apply any of the changes in the set. The `putChangeSet` method carries an `errorResponse` parameter, however, that allows this behavior to be overridden.

Class BaseMaintenanceService

the set of attributes and functions common to all maintenance services

Superclasses:

- Every instance of `BaseMaintenanceService` is also an instance of `UpdateService.`

Operation: createChangeSet

Create a new change set and set it in an OPEN state

Input Parameters:

Return Type: `ChangeSet`

Postconditions:

1. A new change set is always created OPEN
2. changeSetURI is the new unique URI
3. The creation date is set to the current date and time
4. there is no creation source
5. There are no changeInstructions
6. There is no close date
7. The members list starts empty

Operation: readIteratableChangeSet

Return a change set with an iterator that allows its contents to be read in chunks.

Input Parameters:

- **changeSet** - the URI of the change set to be read (Type: `ChangeSetURI`)
- **queryControl**$_{OPT}$ - parameters that control the format, timeout and number of entries per block (Type: `QueryControl`)

Return Type: `IteratableChangeSet`

Exceptions:

- **UnknownChangeSet** - The change set specified could either not be read or located by the service.
- **UnsupportedMatchAlgorithm** - The `matchAlgorithm` is not supported by the service
- **UnsupportedPredicate** - The predicate name or URI is not recognized by the service
- **UnsupportedNamespaceName** - The supplied namespace name is not one that is known to the service.
- **UnsupportedModelAttribute** - The name or URI of a CTS_2 model attribute is not recognized and/or supported by the service implementation
- **UnsupportedFormat** - The `format` is not supported by the service implementation
- **QueryTimeout** - The `timeLimit` was exceeded by the service.

Operation: readChangeSet

Read a complete change set.

Input Parameters:

- **changeSet** - the change set to be retrieved (Type: `ChangeSetURI`)
- **queryControl**$_{OPT}$ - parameters that control the format and time limit (Type: `QueryControl`)

Return Type: `ChangeSet`

Exceptions:

- **UnknownChangeSet** - The change set specified could either not be read or located by the service.
- **UnsupportedMatchAlgorithm** - The `matchAlgorithm` is not supported by the service
- **UnsupportedPredicate** - The predicate name or URI is not recognized by the service
- **UnsupportedNamespaceName** - The supplied namespace name is not one that is known to the service.
- **UnsupportedModelAttribute** - The name or URI of a CTS_2 model attribute is not recognized and/or supported by the service implementation
- **UnsupportedFormat** - The `format` is not supported by the service implementation
- **QueryTimeout** - The `timeLimit` was exceeded by the service.

Operation: updateChangeSetMetadata

Update the descriptive information of a change set.

Input Parameters:

- **changeSet** - the URI of the open change set to update (Type: `ChangeSetURI`)
- **creator**$_{OPT}$ - the name or URI of the change set creator (type SOURCE) (Type: `NameOrURI`)
- **changeInstructions**$_{OPT}$ - instructions about when or how to apply the change set (Type: `OpaqueData`)
- **officialEffectiveDate**$_{OPT}$ - the date that the set of changes officially becomes effective (Type: `DateAndTime`)

Return Type: `Changeable`

Exceptions:

- **ChangeSetIsNotOpen** - The `changeSetContext` is recognized by the service, but its state is not `OPEN` .
- **UnknownChangeSet** - The change set specified could either not be read or located by the service.
- **UnsupportedSource** - The supplied source is not recognized by the service

Operation: commitChangeSet

Apply the proposed changes specified in the change set transaction to the changeable being updated.

Input Parameters:

- **changeSet** - the URI of an open changeset to finalize (Type: `ChangeSetURI`)

Return Type: `ValidationResponse`

Exceptions:

- **ChangeSetIsNotOpen** - The `changeSetContext` is recognized by the service, but its state is not `OPEN` .
- **UnknownChangeSet** - The change set specified could either not be read or located by the service.

Operation: rollbackChangeSet

Cancel the proposed changes specified in a change set transaction to the changeable being updated.

Input Parameters:

- **changeSet** - the URI of an OPEN change set to rollback and delete (Type: `ChangeSetURI`)

Return Type: `ValidationResponse`

Exceptions:

- **ChangeSetIsNotOpen** - The `changeSetContext` is recognized by the service, but its state is not `OPEN` .
- **UnknownChangeSet** - The change set specified could either not be read or located by the service.

Operation: updateChangeableMetadata

Update the metadata for a change set and return an image of the updated resource.

Input Parameters:

- **changeSet** - the URI of an OPEN change set (Type: `ChangeSetURI`)
- **entryID** - the entryID of the resource whose state is to be updated (Type: `PersistentURI`)
- **status**$_{OPT}$ - the name or URI of valid workflow status (Type: `NameOrURI`)
- **entryState**$_{OPT}$ - the new state of the resource (ACTIVE or INACTIVE) (Type: `EntryState`)
- **effectiveDate**$_{OPT}$ - the new effective date of the resource (Type: `DateAndTime`)
- **changeNotes**$_{OPT}$ - the new change notes of the resource (Type: `OpaqueData`)
- **changeSource**$_{OPT}$ - the new change source of the resource (Type: `SourceReference`)

Return Type: `Changeable`

Exceptions:

- **ChangeSetIsNotOpen** - The `changeSetContext` is recognized by the service, but its state is not `OPEN` .
- **UnknownChangeSet** - The change set specified could either not be read or located by the service.
- **UnknownEntity** - The `EntityNameOrURI` is not known to the service
- **UnsupportedStatus** - The name or URI of the `Changeable status` property is not recognized by the service.

Operation: deleteChangeable

Remove the `changeAble` resource from the service.

Input Parameters:

- **changeSet** - the identifier of an OPEN change set that will record the deletion (Type: `ChangeSetURI`)
- **changeableResource** - the *image* of the resource to remove (Type: `Changeable`)

Return Type: `void`

Exceptions:

- **UnknownChangeSet** - The change set specified could either not be read or located by the service.
- **ResourceStateMismatch** - The resource being deleted did not match the state of the resource in the service. A change has occurred since the resource image has been retrieved.
- **ChangeSetIsNotOpen** - The `changeSetContext` is recognized by the service, but its state is not `OPEN` .
- **ResourceIsNotOpen** - The target resource version description has been finalized and cannot be updated.

Operation: newProperty

Create a new `Property` for inclusion in a resource or statement construct

Input Parameters:

- **predicate** - the namespace/name or URI of the statement predicate (Type: `EntityNameOrURI`)
- **value** - the property target (Type: `StatementTarget`)
- **externalIdentifier**$_{OPT}$ - an external identifier that will eventually be assigned to the resource or entity or association triple (Type: `String`)

Return Type: `Property`

Exceptions:

- **UnsupportedNamespaceName** - The supplied namespace name is not one that is known to the service.
- **UnknownPredicateNamespaceName** - The namespace in the `predicate` identifier is not known to or supported by the service

Operation: newResourceTarget

Create a new `StatementTarget` of type `RESOURCE` . This is a helper function that is used in the creation and update of other resources and does *not* change the state of the service. Note that URI's that reference classes, predicates or individuals should be created using the `newEntityReferenceTarget` method

Input Parameters:

- **resource** - the resource URI (Type: `ExternalURI`)

Return Type: `StatementTarget`

Operation: newLiteralTarget

Create a new `StatementTarget` of type `LITERAL` . Note that this is a helper function and does not change the state of the service itself.

Input Parameters:

- **literal** - The target literal (Type: `OpaqueData`)

Return Type: `StatementTarget`

Operation: newEntityReferenceTarget

Create a `StatementTarget` of type `ENTITY` . Note that this is a helper function and does not effect the state of the service. Services may choose to validate the input parameters of this method when it is invoked or may postpone validation until the corresponding create or update service call is invoked.

Input Parameters:

- **entity** - the name or URI of the entity reference (Type: `EntityNameOrURI`)

Return Type: `StatementTarget`

Exceptions:

- **UnsupportedNamespaceName** - The supplied namespace name is not one that is known to the service.
- **UnknownEntity** - The `EntityNameOrURI` is not known to the service

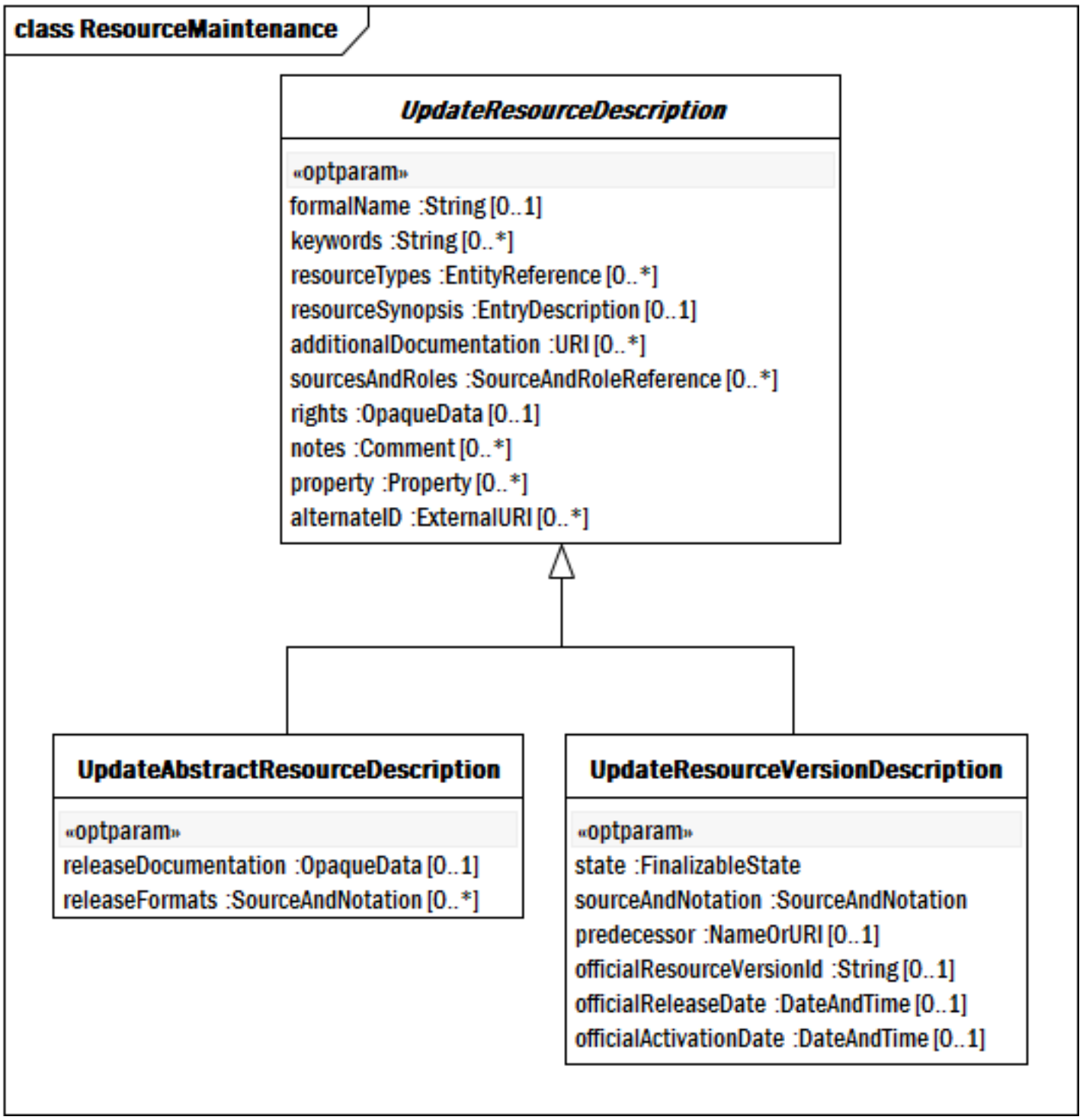


Figure 3.8: Resource Maintenance

The diagram above contains the common model elements that can be changed when updating an instance of an `AbstractResourceDe` and a `ResourceVersionDescription`.

Class UpdateAbstractResourceDescription

The list of attributes that can be updated in an abstract resource.

Superclasses:

- Every instance of `UpdateAbstractResourceDescription` is also an instance of `UpdateResourceDescription`.

Attributes:

- **releaseDocumentation** - documentation about the frequency and nature of releases (version) of this resource.[2]
- **releaseFormats** - a format and notation that the releases (versions) of this resource are published in

Class UpdateResourceDescription

The list of attributes that can be updated in any existing resource.

Attributes:

- **formalName** - the formal or official name of this resource
- **keywords** - additional identifiers used to index and locate this resource
- **resourceTypes** - the class(es) that this resource instantiates
- **resourceSynopsis** - a textual summary of the resource - what it is, what it is for, etc.
- **additionalDocumentation** - references to documents that provide additional information about the resource
- **sourcesAndRoles** - the sources that participated in the creation, validation, review, dissemination, etc. of this resource and the role(s) that they played
- **rights** - copyright and IP information about the referenced resource
- **notes** - additional notes and comments about the resource
- **property** - additional attributes that do not fit into any of the above areas
- **alternateID** - alternative URI's that uniquely name this resource in other environments and contexts

Class UpdateResourceVersionDescription

The set of attributes that can be updated in the description of a resource version

Superclasses:

- Every instance of `UpdateResourceVersionDescription` is also an instance of `UpdateResourceDescription`.

Attributes:

- **state** - the state of the resource version. Note that resource versions can change state from `OPEN` to `FINAL` , but not the other direction. Once a resource version description is finalized and committed, it becomes immutable.
- **sourceAndNotation** - a description of where the (or a) source of the version may be found, what format and language it is available in, etc.
- **predecessor** - the name or URI of the resource version that immediately preceded this version on an evolutionary path
- **officialResourceVersionId** - an official label or identifier that was assigned to this version by its publisher
- **officialReleaseDate** - information about the source, format, release date, version identifier, etc. of a specific version of an abstract resource
- **officialActivationDate** - the date that this version of the resource is stated by its publishers to go into effect

[2] OMV 2.4.1 pp 18

Non-Resource-Specific Services

Import And Export Services

Import and Export Resources

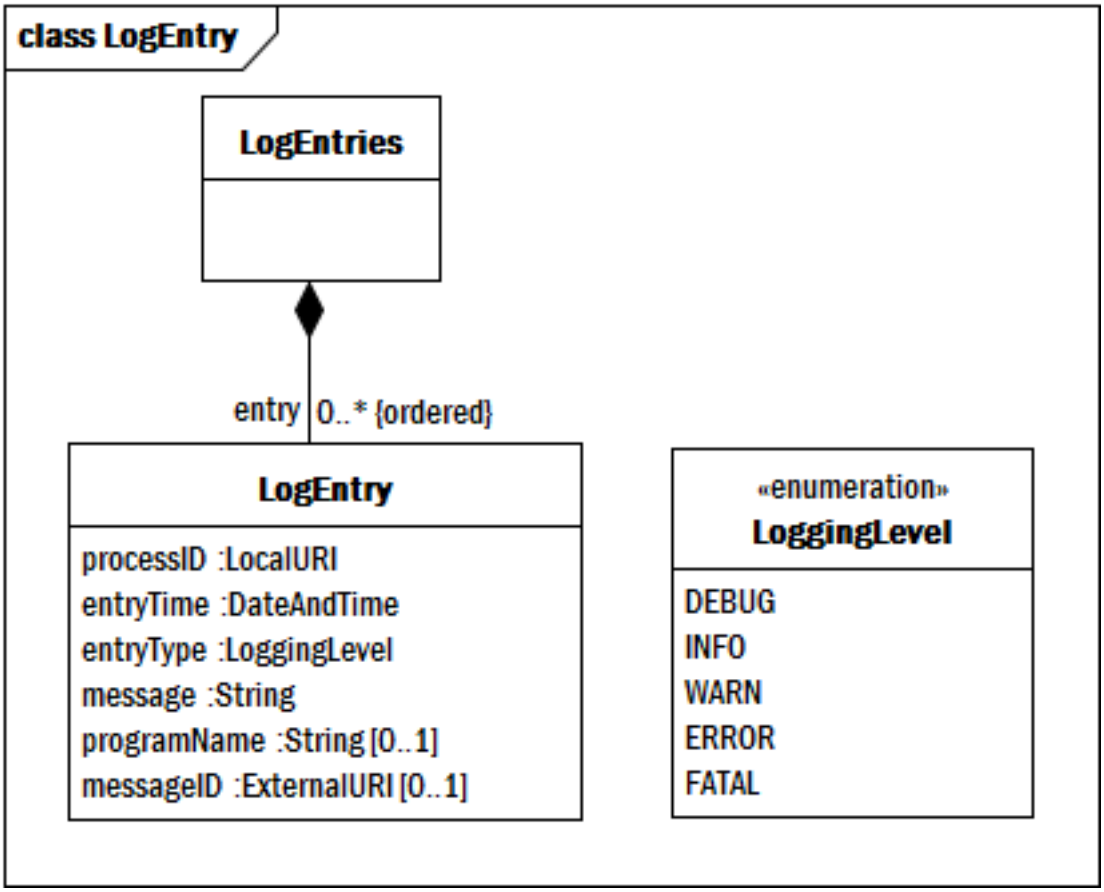


Figure 3.9: Import Export Log

Class LogEntries

A ordered sequence of log entries. The order of the entries is determined by the specific CTS_2 service implementation and should not be assumed by client applications.

Attributes:

- **entry** -

Invariants:

1. Log entries occur in reverse temporal order .

Class LogEntry

an entry in a sequence of messages related to a process or task

Attributes:

- **processID** - the identifier of the process that made the entry
- **entryTime** - the time the entry was made
- **entryType** - the significance of the particular message
- **message** - the text of the message
- **programName** - the name of the program or application that created the entry
- **messageID** - an external identifier that uniquely names the message. `ExternalURI` is present to enable automated processing of log entries where appropriate. The significance and use of `messageID` is not addressed within the context of the CTS_2 specification

Enum LoggingLevel

The `LoggingLevel` entries are modeled after the corresponding levels in the Apache log4j package [3]. As with the log4j package, each level includes the entries in the lower level. In particular, levels are ordered. For the standard levels, we have DEBUG < INFO < WARN < ERROR < FATAL.

Attributes:

- **DEBUG** - detailed (verbose) information about the process or operation that allows a user to determine exactly what transpired
- **INFO** - informative messages about the progress of or results of a requested operation
- **WARN** - represents potential problems that do will not prevent completion of the requested operation, but may require attention or intervention.
- **ERROR** - represents an error that, while serious, may under circumstances be ignored and processing may continue
- **FATAL** - represents an error that prevents further processing. FATAL errors cannot be ignored or overridden.

[3] http://logging.apache.org/log4j/

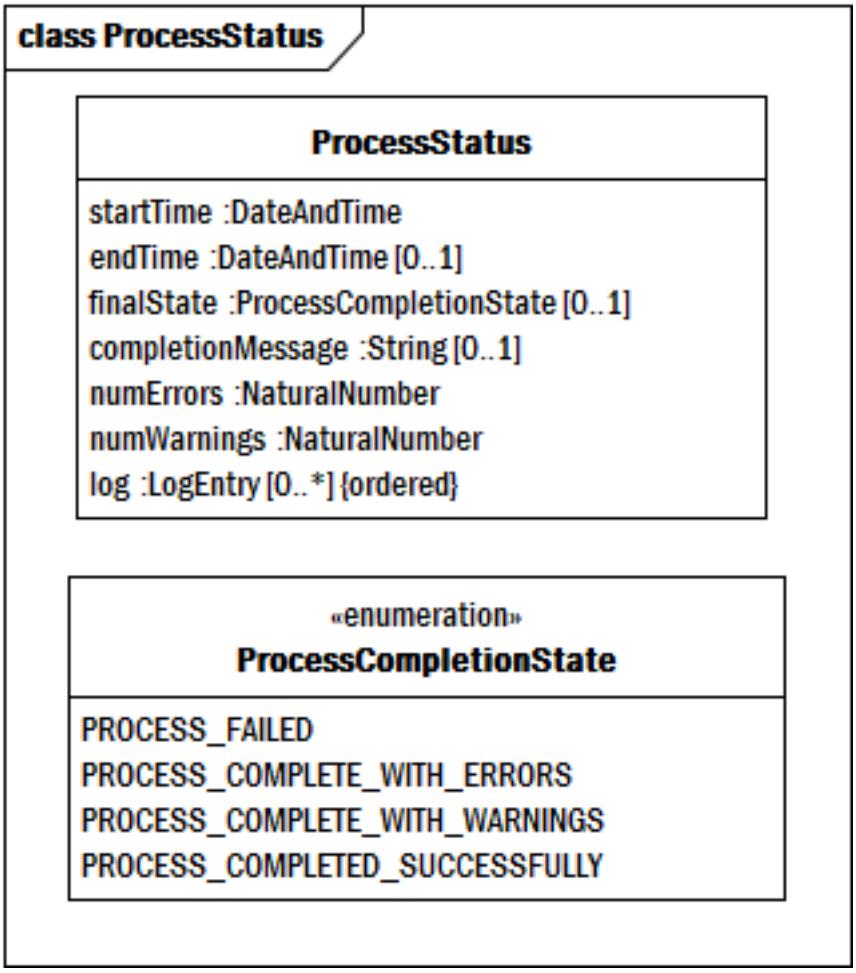


Figure 3.10: Import Export Process Status

Class ProcessStatus

The state of a running or completed load or export process.

Attributes:

- **startTime** - the date and time that the process began execution
- **endTime** - the date and time that the process finished execution if it isn't still running
- **finalState** - an indicator that determines whether the process completed successfully.
- **completionMessage** - a message summarizing the final results of the process
- **numErrors** - the number of errors (`LoggingLevel` = `ERROR` or `FATAL`) encountered by the process so far
- **numWarnings** - the number of warnings (`LoggingLevel = WARN`) encountered by the process so far
- **log** - the set of log records created by the process

Invariants:

1. If a process completes with errors, `numErrors` must be > 0 .
2. If a process completes with warnings, there can be no errors and the number of warnings must be > 0.

3. If a process completes successfully there can be no errors or warnings .
4. Either a process doesn't have an `endTime` , meaning that it is still running, or there is a `finalState` and `completionMessage` available .
5. If there are no errors reported, there will be no errors in the log. Otherwise there are at least as many log entries with an entryType of ERROR .
6. If there are no warnings reported, there will be no warnings in the log. Otherwise there are at least as many log entries with an entryType of WARN .

Enum ProcessCompletionState

An indicator of the completion state of a process

Attributes:

- **PROCESS_FAILED** - the process did not complete successfully
- **PROCESS_COMPLETE_WITH_ERRORS** - the process completed, but there were errors encountered
- **PROCESS_COMPLETE_WITH_WARNINGS** - the process completed and there were no errors but warnings were issued
- **PROCESS_COMPLETED_SUCCESSFULLY** - no errors, no warnings. Ship it!

Import and Export Services

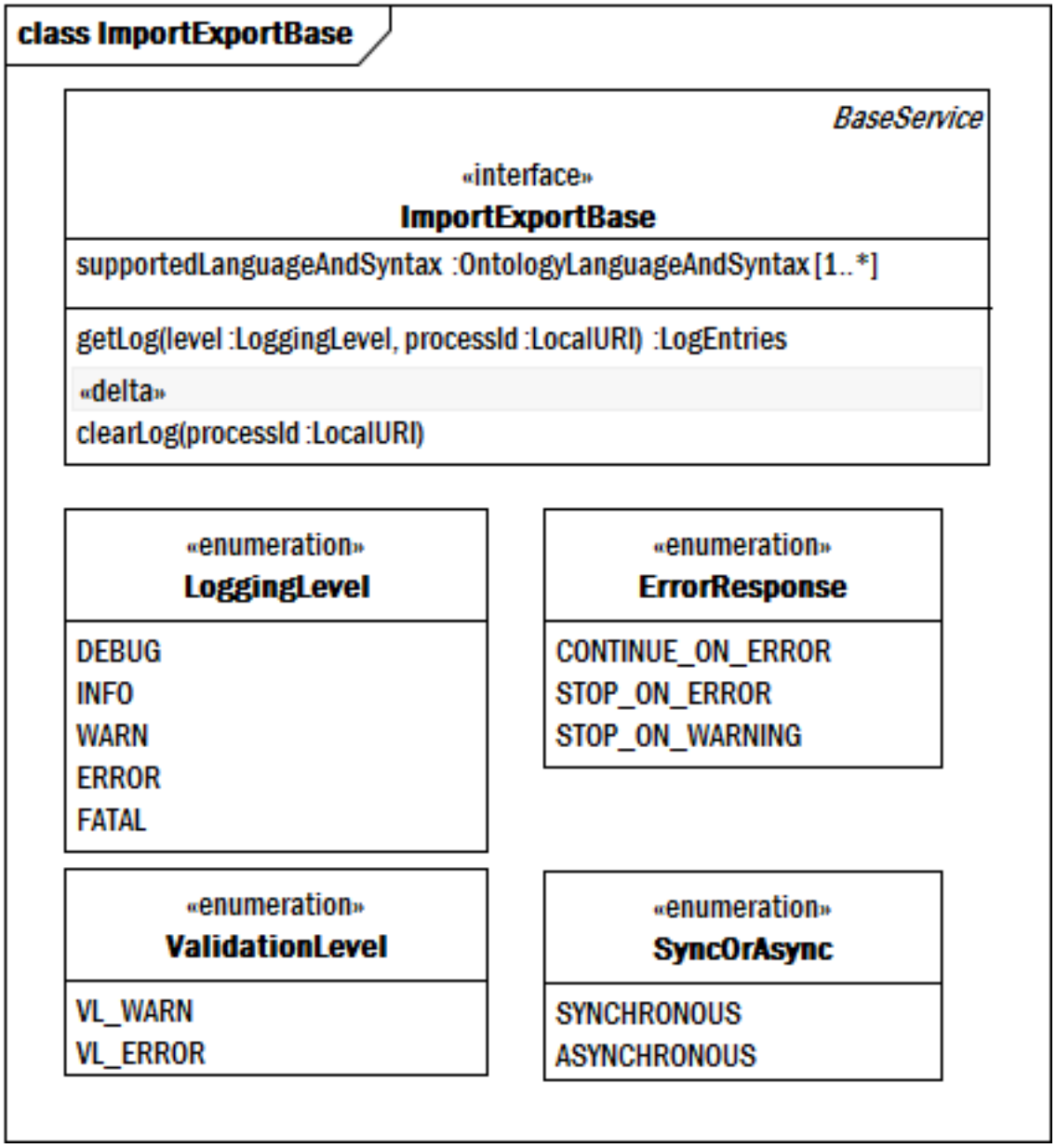


Figure 3.11: Shared Import and Export Service Components

Class ImportExportBase

Superclasses:

- Every instance of `ImportExportBase` is also an instance of `BaseService`.

Attributes:

- **supportedLanguageAndSyntax** - an ontology language and syntax supported by an import or export package

Operation: clearLog

Clear the log entries for the supplied process or, if none is supplied clear all log entries

Input Parameters:

- **processId**$_{OPT}$ - if present, all log entries are removed for this process. If absent, all log entries are removed for the package (Type: `LocalURI`)

Return Type:

Exceptions:

- **UnknownProcessId** - The `processId` is not known to the service

Operation: getLog

Retrieve the log entries for the supplied process and logging level. If a process id isn't supplied, return all log entries and if a log level isn't supplied, return all level of entries

Input Parameters:

- **level**$_{OPT}$ - If present, this parameter restricts the returned log entries to those equal to or more "severe" than the stated level. If absent, all log entries are returned. (Type: `LoggingLevel`)
- **processId**$_{OPT}$ - If present, only log entries for the supplied process are returned. If absent, log entries for all running or completed processes are returned. (Type: `LocalURI`)

Return Type: `LogEntries`

Exceptions:

- **UnknownProcessId** - The `processId` is not known to the service

Enum ErrorResponse

An indicator that determines the server's response to encountering warnings or errors.

Attributes:

- **CONTINUE_ON_ERROR** - continue processing if at all possible. Do not stop unless a `FATAL` error is encountered.
- **STOP_ON_ERROR** - continue processing unless a `FATAL` or `ERROR` event occurs
- **STOP_ON_WARNING** - stop processing if a `WARN` , `ERROR` or `FATAL` event occurs

Enum LoggingLevel

The `LoggingLevel` entries are modeled after the corresponding levels in the Apache log4j package [4]. As with the log4j package, each level includes the entries in the lower level. In particular, levels are ordered. For the standard levels, we have DEBUG < INFO < WARN < ERROR < FATAL.

Attributes:

- **DEBUG** - detailed (verbose) information about the process or operation that allows a user to determine exactly what transpired
- **INFO** - informative messages about the progress of or results of a requested operation
- **WARN** - represents potential problems that do will not prevent completion of the requested operation, but may require attention or intervention.
- **ERROR** - represents an error that, while serious, may under circumstances be ignored and processing may continue
- **FATAL** - represents an error that prevents further processing. FATAL errors cannot be ignored or overridden.

[4] http://logging.apache.org/log4j/

Enum SyncOrAsync

An indicator that determines whether an import, export or other potentially lengthy request occurs in the foreground or returns control to the user.

Attributes:

- **SYNCHRONOUS** - the method call does not return until the operation is complete
- **ASYNCHRONOUS** - the method returns control to the client as soon as the process begins execution

Enum ValidationLevel

An indicator that determines the detail level of a validation request.

Attributes:

- **VL_WARN** - report all `WARN` (warning) and `ERROR` (error) level validation errors
- **VL_ERROR** - only report `ERROR` (error) level validation errors

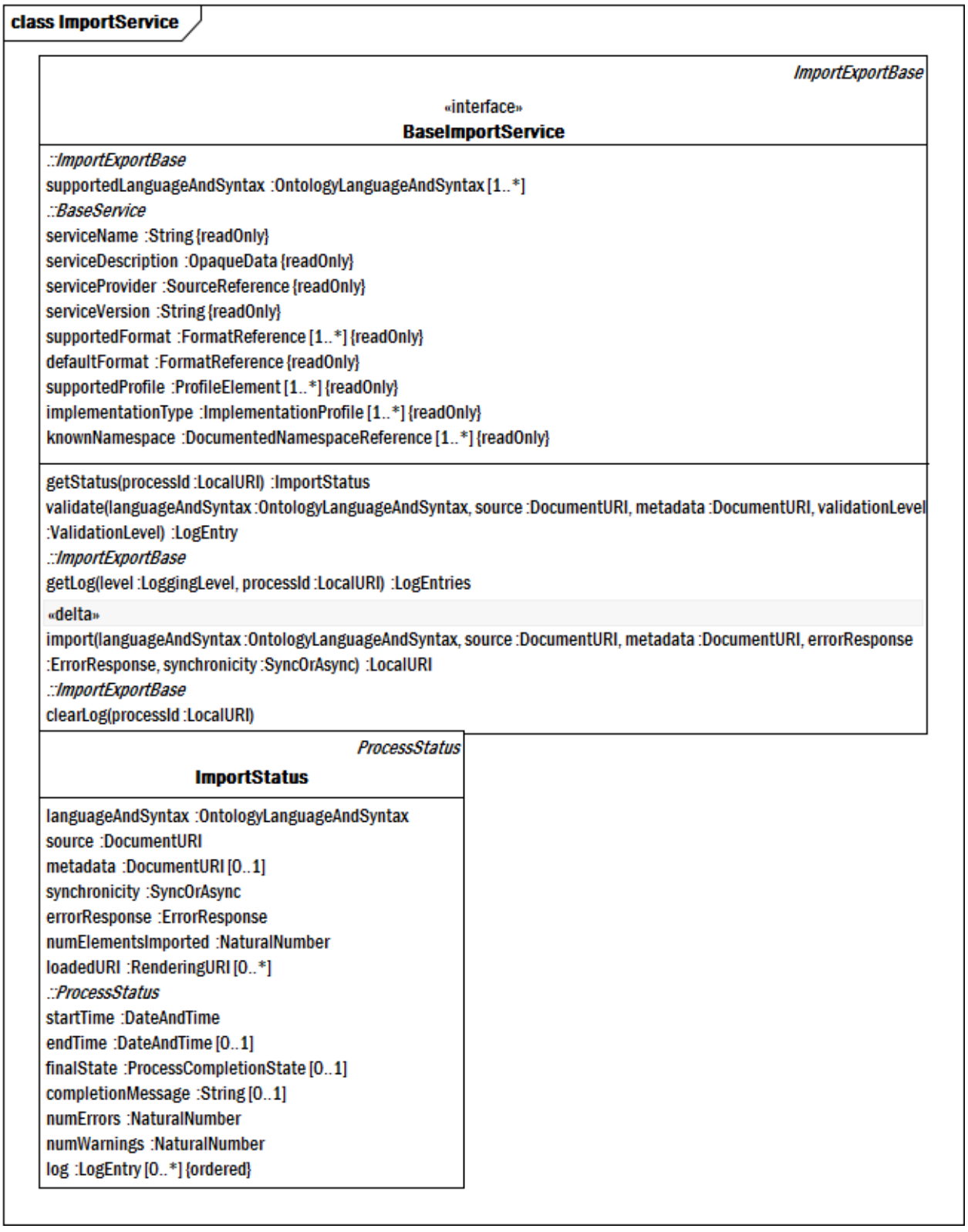


Figure 3.12: Shared Import Service Components

The `ImportService` provides basic access to the `Changeable` components of the corresponding structural profile. Each structural profile extends the base import service to provide access to that profile's components.

Class BaseImportService

Superclasses:

- Every instance of `BaseImportService` is also an instance of `ImportExportBase`.

Operation: getStatus

Return the import status for the supplied process identifier

Input Parameters:

- **processId** - The import process to return the status of (Type: `LocalURI`)

Return Type: `ImportStatus`

Exceptions:

- **UnknownProcessId** - The `processId` is not known to the service

Operation: import

Load structured content from a specified source and return a URI that references the "process" that performed or is performing the import

Input Parameters:

- **languageAndSyntax** - the formal language and syntax of the resource to be imported. languageAndSyntax must be in the service supportedLanguageAndSyntax (Type: `OntologyLanguageAndSyntax`)
- **source** - the source document or resource to import from (Type: `DocumentURI`)
- **metadata**$_{OPT}$ - the URI of additional metadata to control the import process (Type: `DocumentURI`)
- **errorResponse** - an indicator that determines how the service should behave when it encounters an error or warning (Type: `ErrorResponse`)
- **synchronicity** - an indicator that determines whether the service should return as soon as the process starts or should wait until it is completed (Type: `SyncOrAsync`)

Return Type: `LocalURI`

Exceptions:

- **UnableToOpenDocument** - the process is unable to open the primary document to be imported
- **UnsupportedOntologySyntax** - The supplied ontology syntax is not supported by the service
- **UnsupportedOntologyLanguage** - The supplied ontology language is not supported by the service
- **MetadataError** - An error was encountered in the metadata document. This error includes formatting errors, missing parameters and invalid content.
- **UnableToOpenMetadataDocument** - the service is unable to access the supplied metadata document
- **MetadataDocumentRequired** - Additional metadata must be supplied to perform this operation

Operation: validate

Determine whether the source document would import successfully if it were imported

Input Parameters:

- **languageAndSyntax** - the language and syntax of the import source (Type: `OntologyLanguageAndSyntax`)
- **source** - the URI of the resource to import (Type: `DocumentURI`)
- **metadata**$_{OPT}$ - metadata that controls the import process (Type: `DocumentURI`)
- **validationLevel** - level of validation to perform (Type: `ValidationLevel`)

Return Type: `LogEntry`

Exceptions:

- **UnableToOpenDocument** - the process is unable to open the primary document to be imported
- **MetadataError** - An error was encountered in the metadata document. This error includes formatting errors, missing parameters and invalid content.
- **UnableToOpenMetadataDocument** - the service is unable to access the supplied metadata document
- **MetadataDocumentRequired** - Additional metadata must be supplied to perform this operation
- **UnsupportedOntologySyntax** - The supplied ontology syntax is not supported by the service
- **UnsupportedOntologyLanguage** - The supplied ontology language is not supported by the service

Class ImportStatus

`ImportStatus` represents the state of an import process that is currently running or has completed.

Superclasses:

- Every instance of `ImportStatus` is also an instance of `ProcessStatus`.

Attributes:

- **languageAndSyntax** - the ontology language and syntax of the imported document
- **source** - the uri of the document or resource being imported
- **metadata** - the URI of the associated metadata document if any
- **synchronicity** - an indicator whether the import was done synchronously or asynchronously
- **errorResponse** - an indicator that determines how the process should (or did) behave when it encountered warnings or errors
- **numElementsImported** - the number of elements that have been successfully imported
- **loadedURI** - the CTS2 URI(s) of the loaded resources

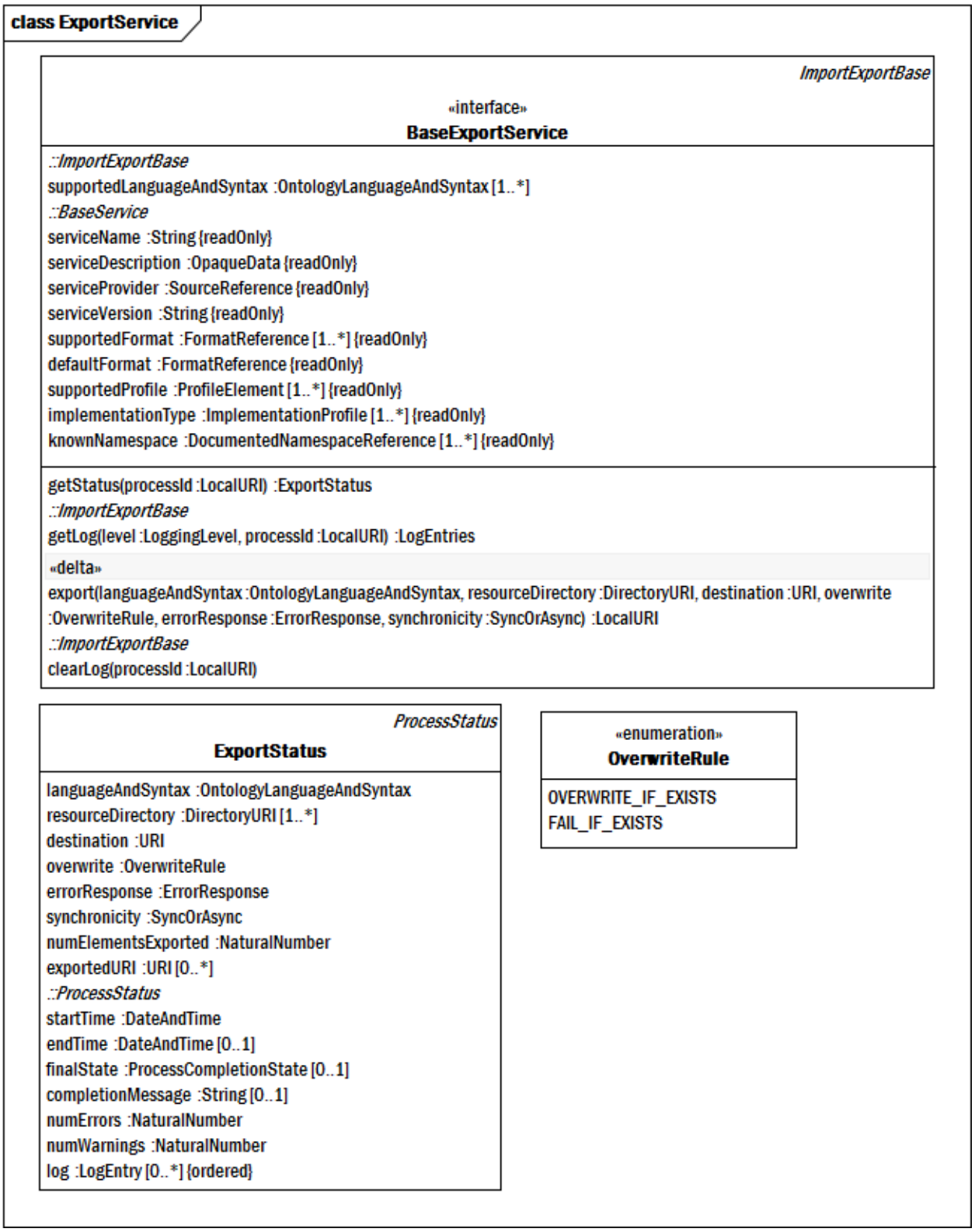


Figure 3.13: Shared Export Service Components

The `ExportService` provides basic access to the `Changeable` components of the corresponding structural profile. Each structural profile extends the base export service to provide access to that profile's components.

Class BaseExportService

Superclasses:

- Every instance of `BaseExportService` is also an instance of `ImportExportBase`.

Operation: getStatus

Obtain a current status of the export process.

Input Parameters:

- **processId** - the URI or handle of a running or completed export process (Type: `LocalURI`)

Return Type: `ExportStatus`

Exceptions:

- **UnsupportedStatus** - The name or URI of the `Changeable status` property is not recognized by the service.

Operation: export

Export structured content to a specified location with a specified format and return a URI that represents the (results of) the process that did the exporting

Input Parameters:

- **languageAndSyntax** - the language and syntax of the target resource (Type: `OntologyLanguageAndSyntax`)
- **resourceDirectory** - the URI of a directory that references the list of the CTS_2 resources / entities / associations to export (Type: `DirectoryURI`)
- **destination** - the URI of the export destination. May reference a file, directory or other resource depending upon the type of export implementation (Type: `URI`)
- **overwrite** - an indicator that determines whether existing resources can be overwritten or, if they already exist, the export operation fails (Type: `OverwriteRule`)
- **errorResponse** - what to do when an error is encountered (Type: `ErrorResponse`)
- **synchronicity** - an indicator that determines whether the operation is to block until completed or whether it is to run in the background (Type: `SyncOrAsync`)

Return Type: `LocalURI`

Exceptions:

- **UnsupportedResourceType** - the type of resource referenced by the directory is not supported by the service - either it can't be exported period or it can't be exported in the supplied language or format
- **ResourceAlreadyExists** - a referenced resource already exists and `overWrite` is set to `FAIL_IF_EXISTS`
- **UnableToOpenOrCreateTargetDirectory** - the export service is unable to open or create the supplied target directory
- **ResourceWriteError** - an error occurred while trying to write the exported image of a resource into the directory
- **UnsupportedOntologySyntax** - The supplied ontology syntax is not supported by the service
- **UnsupportedOntologyLanguage** - The supplied ontology language is not supported by the service

Preconditions:

1. The supplied language and syntax must be one of those supported by the service

Class ExportStatus

`ExportStatus` represents the state of an export process that is currently running or has completed.

Superclasses:

- Every instance of `ExportStatus` is also an instance of `ProcessStatus`.

Attributes:

- **languageAndSyntax** - the output language and syntax
- **resourceDirectory** - The directory of resources being exported
- **destination** - The export destination
- **overwrite** -
- **errorResponse** -
- **synchronicity** -
- **numElementsExported** - The number of elements that have been exported so far
- **exportedURI** - A list of URIs of the actual exported resources. This may match the destination URI or may represent separate resources (files) in the context of the destination URI

Enum OverwriteRule

Attributes:

- **OVERWRITE_IF_EXISTS** -
- **FAIL_IF_EXISTS** -

Update Service

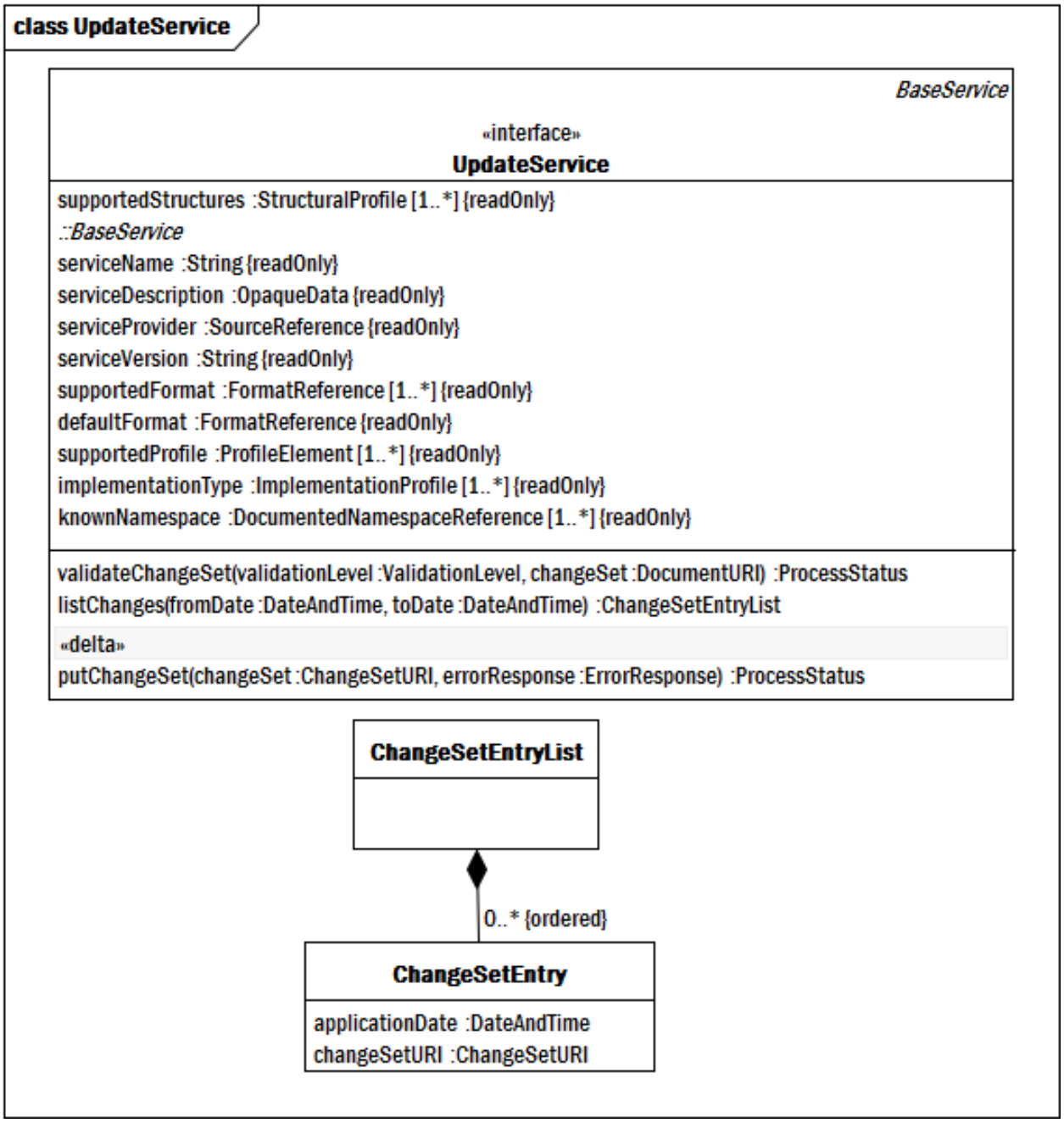


Figure 3.14: Update Service

Class ChangeSetEntry

A change set URI along with the date and time it was applied to a particular service implementation

Attributes:

- **applicationDate** - the date and time that a change set was applied to a given service instance
- **changeSetURI** - the he URI of the change set that was applied

Class ChangeSetEntryList

An ordered list of `ChangeSetEntries`. The order of the list reflects the order in which the change sets were applied to the service instance.

Attributes:

- - An entry in a ChangeSetEntryList

Class UpdateService

An update service provides the ability to apply incremental changes to one or more structural elements that are supported by the service

Superclasses:

- Every instance of `UpdateService` is also an instance of `BaseService`.

Attributes:

- **supportedStructures** - the set of structural profiles that the update service can apply changes to

Invariants:

1. Every structure listed in supported structures must also exist as a supportedProfile associated with a FP_UPDATE functional profile .

Operation: validateChangeSet

Validate the change set and report any errors or warnings that would occur were it to be applied.

Input Parameters:

- **validationLevel** - an indicator that determines whether just errors are reported or both errors and warnings (Type: `ValidationLevel`)
- **changeSet** - a uri the resolves to a valid change set. This could be the address of a change set in another service or an image of a change set on a network drive (Type: `DocumentURI`)

Return Type: `ProcessStatus`

Exceptions:

- **UnableToOpenDocument** - the process is unable to open the primary document to be imported

Operation: putChangeSet

Install the supplied change set into the service.

Input Parameters:

- **changeSet** - a uri the resolves to a valid change set. This could be the address of a change set in another service or an image of a change set on a network drive (Type: `ChangeSetURI`)
- **errorResponse**$_{OPT}$ - an indicator that determines how errors and warnings are processed. (Type: `ErrorResponse`)

Return Type: `ProcessStatus`

Operation: listChanges

List the URIs of all the changes that were applied between `fromDate` and `toDate` . If `fromDate` is absent or is less than `toDate` , entries will be listed in forward chronological order, otherwise they will be listed in reverse order.

Input Parameters:

- **fromDate**$_{OPT}$ - list change sets that were applied on or after this date. If not supplied, start with the first change set that was applied (Type: `DateAndTime`)
- **toDate**$_{OPT}$ - list all changes that were applied before this date. If not supplied, list all changes that were applied later than or equal to the toDate (Type: `DateAndTime`)

Return Type: `ChangeSetEntryList`