Date: June 2012

Common Terminology Services 2

FTF Beta 2

OMG Document Number*: ptc/2012-06-20
Standard document URL: http://www.omg.org/spec/CTS2/1.0/

---

original document: ad/11-06-14, dtc/2011-09-01, dtc/2012-05-02

This OMG document replaces the submission document (ad/2011-05-12, Alpha). It is an OMG Adopted Beta Specification and is currently in the finalization phase. Comments on the content of this document are welcome, and should be directed to issues@omg.org by December 1, 2011.

You may view the pending issues for this specification from the OMG revision issues web page

http://www.omg.org/issues/.

The FTF Recommendation and Report for this specification will be published on June 29, 2012. If you are reading this after that date, please download the available specification from the OMG Specifications Catalog.

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE.
IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE,

INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 140 Kendrick Street, Needham, MA 02494, U.S.A.

TRADEMARKS

MDA®, Model Driven Architecture®, UML®, UML Cube logo®, OMG Logo®, CORBA® and XMI® are registered trademarks of the Object Management Group, Inc., and Object Management Group™, OMG™ , Unified Modeling Language™, Model Driven Architecture Logo™, Model Driven Architecture Diagram™, CORBA logos™, XMI Logo™, CWM™, CWM Logo™, IIOP™ , MOF™ , OMG Interface Definition Language (OMG IDL)™ , and OMG Systems Modeling Language (OMG SysML)™ are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

Table of Contents

Preface

About the Object Management Group

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at *http://www.omg.org/*.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. A catalog of all OMG Specifications is available from the OMG website at:

http://www.omg.org/technology/documents/spec_catalog.htm

Specifications within the Catalog are organized by the following categories:

OMG Modeling Specifications

- UML
- MOF
- XMI
- CWM
- Profile specifications

OMG Middleware Specifications

- CORBA/IIOP
- IDL/Language Mappings
- Specialized CORBA specifications
- CORBA Component Model (CCM)

Platform Specific Model and Interface Specifications

- CORBAservices

- CORBAfacilities
- OMG Domain specifications
- OMG Embedded Intelligence specifications
- OMG Security specifications

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. (as of January 16, 2006) at:

OMG Headquarters
140 Kendrick Street
Building A, Suite 300
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: *pubs@omg.org*

Certain OMG specifications are also available as ISO standards. Please consult *http://www.iso.org*

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text

Helvetica/Arial - 10 pt. Bold: OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier - 10 pt. Bold:` Programming language elements.

Helvetica/Arial - 10 pt: Exceptions

Note – Terms that appear in *italics* are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification to *http://www.omg.org/technology/agreement.htm*.

1 Scope

Structured terminologies provide a foundation for information interoperability by improving the effectiveness of information exchange. They provide a means for organizing information and serve to define the semantics of information using consistent and computable mechanisms.

Terminologies are constructed to meet scope specific domain requirements. The domain specific nature of structured vocabularies often leads to variation in design patterns across the available terminology space. The ability to provide consistent representation and access to a broad set of terminologies enables multiple disparate terminology sources to be available to a community, and helps to ensure consistency across the domain space of that community. Service interfaces to structured terminologies should be flexible enough to accurately represent a wide variety of vocabularies and other lexically-based resources.

The PIM specified in this document for CTS 2 is intended to mediate among disparate terminology sources by providing a standard service information and computational model. The *Information Model* specifies the structural definition, attributes and associations of *Resources* common to structured terminologies such as Code Systems, Binding Domains and Value Sets. The *Computational Model* specifies the service descriptions and interfaces needed to access and maintain structured terminologies.

2 Conformance

This specification defines a PIM that specifies an Information Model as well as a Computational Model. Conformant implementations of this PIM must provide an implementation that represents both the Information Model and Computational Model. This base level implementation provides the foundation for providing data type specific profiles (i.e., ISO 21090 data types, HL7 data types, etc.).

Conformant implementations must adhere to the profiles outlined in the Computation Model of this PIM, which are derived from the CTS 2 SFM.

2.1 Implementation Profiles

The CTS2 specification allows modular implementation. CTS2 service instances may chose to implement only the components and functionality that are relevant to their specific needs and use cases. The intent of this modularity is two-fold. The first goal is that the specification provides what the NCI has been calling the "linear value proposition", where relatively simple things are easy to implement and the cost of the implementation increases in proportion to the desired complexity. As an example, an organization that maintains a catalog of metadata about available ontologies could publish this catalog using the CTS2 standard by implementing the CODE SYSTEM structural profile along with the READ and QUERY functional profiles.

The second goal of the CTS2 modularity is to enable terminology resources to be distributed and federated. If, for instance, an organization needed to maintain a small, specialized ontology that builds on a number of other already existing ontologies, they might implement the ASSOCIATION and, possibly, the ENTITY structural profiles including the READ, QUERY, and AUTHORING profiles and could link to the first organization’s code system catalog for additional information about the ontologies that were imported.

This section starts by defining the common elements that are shared by the different profiles. It defines a core set of data types, several key structural components and a number of abstractions that are reused throughout the rest of the models.

It also defines the core functional components – characteristics that all CTS2 service implementations must possess as well as characteristics that are common to each specific functional profile.

It then provides a section for each of the eleven possible structural profiles. Each section describes what the specific profile is intended to represent, defines the structural components specific to the profile and then defines the functional characteristics each functional profile provides with respect to the target component.

The individual structural, functional, and implementation profile components are described briefly below. Readers are encouraged to refer to the corresponding chapters for additional detail.

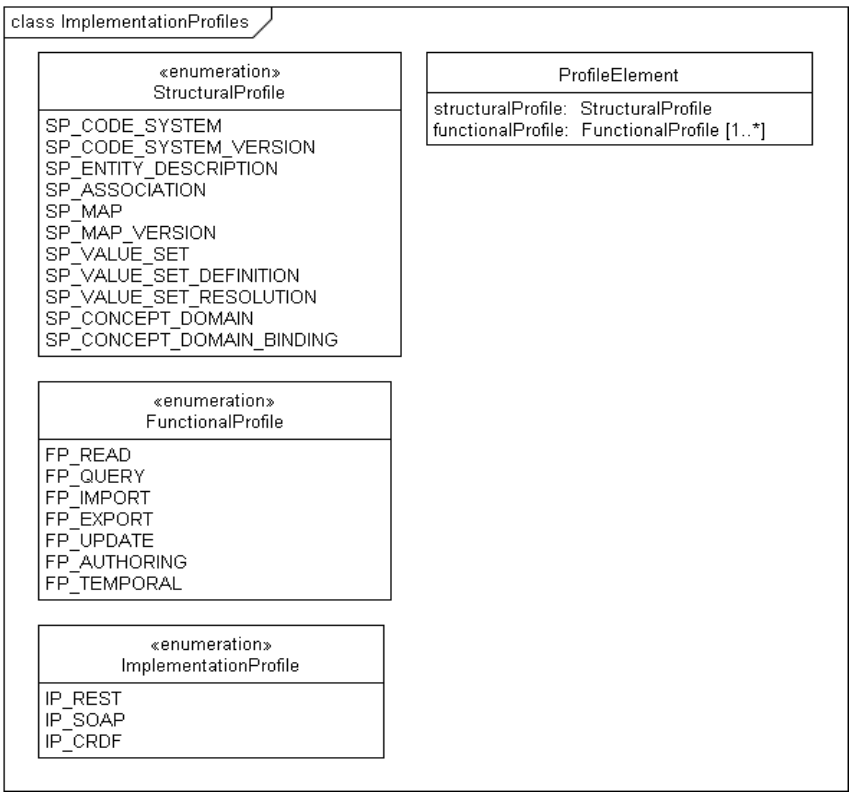

Figure 2.1 - Implementation Profiles

Class ProfileElement

`ProfileElement` appears in service implementations, once per structural profile that is supported by the implementation instance. Each occurrence records the set of functional profiles that are supported for the specific structural profile.

Attributes

- structuralProfile – a structural profile that is supported by the service.
- functionalProfile – a functional profile that is supported for the particular structural profile.

Enum FunctionalProfile

An enumeration of the possible functional profiles, some or all of which can be implemented by a conformant CTS2 service.

Attributes

- **FP_READ** – the implementation supports direct read access for artifacts of the associated structural profile.
- **FP_QUERY** – the implementation supports search and enumerate access for artifacts of the associated structural profile.
- **FP_IMPORT** – the implementation supports the ability to import resources from one or more external formats into elements of the associated structural profiles.
- **FP_EXPORT** – the implementation supports the ability to export elements of the associated structural profiles in one or more external formats.
- **FP_UPDATE** – the service supports the ability to apply incremental updates (`ChangeSets`) to the associated resources.
- **FP_MAINTENANCE** – the service supports the ability to create `ChangeSets`.
- **FP_TEMPORAL** – the system supports the ability to read and query (as supported by the service) the service in the context of a date and time different.

Enum ImplementationProfile

Indicates what PSM(s) are supported by the given service implementation.

Attributes

- **IP_REST** – the service supports the REST PSM.
- **IP_SOAP** – the system supports the SOAP implementation profile.
- **IP_CRDF** – the service supports the “Canonical RDF” PSM.

Enum StructuralProfile

The CTS2 specification defines eleven distinct structural profiles. CTS2 compliant implementations may elect to implement any combination of these profiles to meet their individual requirements and use cases.

Attributes

- **SP_CODE_SYSTEM** – The CODE_SYSTEM profile provides a catalog of classification systems, code systems, ontologies, thesauri, etc. known to the service and may also carry information about their publisher, release cycles, purpose, etc.
- **SP_CODE_SYSTEM_VERSION** – The CODE_SYSTEM_VERSION profile carries information about the various versions of a code system, including the release date, release format, contact information, etc.
- **SP_ENTITY_DESCRIPTION** – The ENTITY_DESCRIPTION profile provides a representation of a collection of descriptions about classes, roles, or individuals along with links to the code system version(s) in which these descriptions originate. An entity description provides the “lexical” or “non-semantic” components of a description, while the ASSOCIATION profile provides the logic-based “semantic” counterpart.
- The service supports the
- `EntityDescription` structural model, which means that it can represent sets of lexical assertions about classes, roles, and/or individuals as asserted by a specific code system version.

- **SP_ASSOCIATION** – The ASSOCIATION profile represents a collection of structured, “semantic” assertions about classes, roles, and/or individuals along with links to the code system versions(s) that were the source of these assertions. The service supports the Association structural model, which means that it can represent sets of semantic assertions about classes, roles, and/or individuals as asserted by a specific code system version.
- **SP_VALUE_SET** – The VALUE_SET profile provides a catalog of value sets that are known to the service.
- **SP_VALUE_SET_DEFINITION** – The VALUE_SET_DEFINITION structural profile provides definitions that, when interpreted using specified code system versions, result in a set of entity descriptions. Definitions are associated with value sets and can vary and evolve over time.
- **SP_VALUE_SET_RESOLUTION** – The VALUE_SET_RESOLUTION profile describes rules for ordering and associating additional properties with the result of interpreting a value set definition. VALUE SET RESOLUTION allows resolved definitions to be rendered in a given language and context, sorted and filtered.
- **SP_CONCEPT_DOMAIN** – A CONCEPT_DOMAIN profile describes the equivalent of a 11179 Data Element Concept. It identifies an abstract unit of information that can appear on a message, form or database along with metadata about its use, author, purpose, etc.
- **SP_CONCEPT_DOMAIN_BINDING** – A CONCEPT_DOMAIN_BINDING profile describes the equivalent of the 11179 Data Element, associating a CONCEPT-DOMAIN with a value set and describing the context and rules where the association applies.
- **SP_MAP** – The MAP profile defines collections of rules for transforming information represented using one code system into information represented in a section. MAP describes abstract collections such as “The SNOMED-CT to ICD-10 map,” along with the creators, intended use, code systems involved, etc.
- **SP_MAP_VERSION** – The MAP_VERSION profile represents the actual content of a MAP at a given point in time. It carries the from and to components as well as a representation of the rules and process for the conversion.

3 Normative References

The following normative documents contain provisions which, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply.

[ISO 11179]	ISO/IEC 11179, Information technology – Specification and standardization of data elements
[ISO 21090]	ISO 21090, Health informatics – Harmonized data types for information exchange
[HL7 Data Types]	HL7 V3 – Data Types – Abstract Specification

4 Terms and Definitions

For the purposes of this specification, the following terms and definitions apply.

Computation Independent Model (CIM)

A computation independent model is a view of a system from the computation independent viewpoint. A CIM does not show details of the structure of systems. A CIM is sometimes called a domain model, and a vocabulary that is familiar to the

practitioners of the domain in question is used in its specification. Some ontologies are essentially CIMs from a software engineering perspective.

HL7 Model Interchange Format (MIF)

A set of XML formats used to support the storage and exchange of HL7 version 3 artifacts as part of the HL7 Development Framework.

Platform Independent Model (PIM)

A model of a subsystem that contains no information specific to the platform, or the technology that is used to realize it.

Platform Specific Model (PSM)

A model of a subsystem that includes information about the specific technology that is used in the realization of it on a specific platform, and hence possibly contains elements that are specific to the platform.

Unified Modeling Language (UML)

An OMG standard language for specifying the structure and behavior of systems. The standard defines an abstract syntax and a graphical concrete syntax.

XML Metadata Interchange (XMI)

An OMG standard that facilitates interchange of models via XML documents.

5 Symbols

CIM	Computation Independent Model
ISO/IEC	International Organization for Standardization / International Electrotechnical Commission
OMG	Object Management Group
PIM	Platform Independent Model
PSM	Platform Specific Model
UML	Unified Modeling Language 2.0
URI	Uniform Resource Identifier
XMI	XML Metadata Interchange
XML	Extensible Markup Language

6 Additional Information

6.1 Acknowledgements

This specification in response to the Common Terminology Services Release 2 (CTS2) RFP is made by the Mayo Clinic /

Foundation. This specification is supported in part by the National Cancer Institute, caBIG® initiative.

The following companies submitted this specification:

- Mayo Clinic / Foundation

The authors would like to thank the National Cancer Institute (NCI), caBIG® community for supporting the development of CTS 2 throughout the OMG standards adoption process, as well as the active members of the OMG's Healthcare DTF, Ontology PSIG, and the HL7 Vocabulary community for their help and support towards the preparation of this document.

The following companies and organizations support this specification:

- 3M Health Information Systems, Inc.
- Apelon, Inc.
- Everware-CBDI
- Hewlett-Packard Company
- Intermountain Healthcare
- International Health Terminology Standards Development Organisation (IHTSDO)
- Model Driven Solutions
- National Cancer Institute (NCI) Enterprise Vocabulary Services
- NoMagic, Inc.
- Sandpiper Software, Inc.
- Sparx Systems
- Tethers End
- University of Oxford, UK, Department of Computer Science
- Visumpoint

6.2 Guide to Specification

This specification represents a PIM and PSM for the Common Terminology Services – Release 2 (CTS 2) terminology service. It specifies a platform independent service interface to a broad set of structured terminology resources. The requirements for the CTS 2 were initially developed as a Service Functional Model (SFM) within a project sponsored by the Vocabulary Work Group within the Health Level Seven (HL7) community. The HL7 SFM served as the basis for developing the OMG RFP for CTS 2, calling for responses to specify a PIM and PSM for CTS 2.

Introduction to the Specification

The CTS2 specification is designed to address a broad range of requirements within the ontology and terminology community. The use cases range for a need to be able to publish simple catalogs that identify what resources are available to the ability to serve the content of multiple formal ontologies, performing online reasoning and classification. The CTS2 specification also recognizes that terminological services will not necessarily be centralized – one organization may publish catalogs, a second content and yet another may serve value sets, maps and other resources based on these tools.

The goal of this specification includes the ability to provide *distributed*, *federated* terminology services, enabling replicated service instances that periodically synchronize their content as well as service instances that reference the content in other instances. Our goal in no small part is to provide the core infrastructure that allows terminology services to be coupled and interlinked in much the same way that pages are interlinked today in the World Wide Web. Many of the design decisions that went into this specification reflect this need.

Design Philosophy

The CTS2 specification is based on the RESTful Architectural Style as described by Ray Fielding. It identifies a number of relatively fine-grained resources that have persistent identity and then describes how these resources are accessed through generic *create* (PUT), *read* (GET), *update*(POST or PUT) and *delete*(REMOVE) operations. The specification adheres to the idempotency rules laid out in Fieldings document, while introducing an additional notion of a transactional (ChangeSet) layer that allows the synchronization and exchange of collections of changes between service instances.

Linkage between CTS2 resources are loosely coupled and are based on Universal Resource Identifiers (URI's). This makes it possible for one service implementation to implement a query service that references resources in many other services. Similarly, a service may implement a map from one code system to another that depends on a second service to represent the source codes another service the target codes and yet another to represent the metadata such as languages and mime types that are consumed by the implementation itself.

The goal of the CTS2 specification is to provide what has been called a "linear value proposition" – the idea that simple requirements are simple to implement and the complexity of the implementation increases in direct proportion with the complexity of the service requirements. We believe that we have achieved this goal, in that it has been demonstrated that it is possible to implement any of the basic read or simple query modules of the CTS2 specification with nothing more than an XML editor and a backing file system.

Identifiers and Linkage

The CTS2 specification requires that all of the resources represented in service implementations be represented by Universal Resource Identifiers (URI's) that uniquely the name and, ideally, are permanent and persist across service instances. The specification also recognizes that large collections of URIs are difficult to maintain and access without the addition of some sort of a more succinct, human readable form. Where appropriate, the specification requires that resources also be provided an additional local identifier that uniquely references the resource within the context of the service implementation. This identifier model is analogous to the XML Namespace model, where the header of an XML specification names the URIs that are used in the XML document and assigns surrogate identifiers that can be used in place of the URI in the context of the containing document. As with XML documents, local service identifiers are not transferrable across service instances. Just as one XML document may use "xs" to represent the URI of XML Schema and a second may use "xsd", one CTS2 service may use "SCT" to represent the SNOMED-CT ontology while another instance may use "SNOMED-CT" or some other identifier.

Entity (concept, class, predicate, individual) identifiers are a bit more complex. While not strictly required, code systems and ontologies have arrived at various mechanisms for combining a scoping namespace and a code to form the complete URI. As (a) this is not always the case – opaque GUIDs, Digital Object Identifies and other schemes are also used and (b) even when the namespace URI and code are known it there is no standardized way of combining them into a URI, the CTS2 specification remains silent on the relationship between namespaces, codes and URIs. It does, however, require that an entity be referenced by both a local identifier, in the form of a *namespace*/*name* tuple and by a URI. The namespace itself is a service specific local identifier that is associated with a URI. As with RDF and XML, namespace identifiers are service specific and may not be shared across service instances.

An additional nuance of entity identifiers is the *name* component. Debates continue to rage about the merits of semantically opaque identifiers vs. recognizable language specific labels that risk becoming incorrect or dated as the ontology evolves. CTS2 remains silent on this aspect of terminologies and simply states that every *namespace/name* tuple

must map to the official URI of the entity. Different service implementations may use different mechanisms to accomplish this goal including assigning meaningful labels in the place of meaningless codes or even allowing more than one namespace/name combination to reference the same resource.

Specification Structure

This specification is divided into a number of cleanly separable sections or "structural domains":

Code System Catalog – metadata about code systems (ontologies, code sets, thesauri, classification systems, etc.). A service would implement this section to publish information about what sort of terminological resources are available, who maintains them, what they are intended to be used for, how often they are released, what copyrights pertain to them, etc.

Code System Version Catalog – metadata about specific versions of code systems. A service would implement this section to provide information about specific versions of code systems – when they were released, what format they are in, when they were intended to become active, which version they replace, etc.

Entity Description – a set of entity (aka. "class", "category", "concept", "predicate", "property", "term", "individual") identifiers known to the service along with information about which code system versions make assertions about these identifiers and what they say. The entity description service focuses on the lexical aspect of entity identifiers – providing access to the designations, definitions, descriptions, examples, usage notes and other artifacts used to represent the meaning of the entity to human consumers. Services would implement this section to publish information about the codes described in various versions of code systems along with their intended meanings.

Association – sets of "semantic" assertions about entity identifiers, in which the entity identifier may play the role of subject, predicate (verb) or object in the assertions. This area represents the formal machine interpretable aspects of code systems, and a service would implement this section to support classification, reasoning or other computational logic systems built on terminological content.

Value Set Catalog – metadata about sets of entity identifiers (value sets) that have been grouped for some purpose. A service would implement this section if it wanted to publish information about these collections such as who created them, who maintains them, what is their purpose, what code systems do they depend on, how they are updated, etc.

Value Set Definition - information about how value sets are constructed. A value set definition can be a simple enumeration of its elements or can contain instructions about how the elements are assembled from aspects of entity descriptions and associations. Value set definitions may be coupled to specific versions of code system, or may be defined in a way that they could be applied to different versions, potentially with different results. Services would implement this section to publish the rules on the construction of value sets. The value set definition section includes an optional sub-section (**Resolved Value Set**) that enables the publication, loading and use of the results of applying value set definitions. Services would implement this section when they needed to consume and use value sets without having access to the underlying code systems.

Concept Domain Catalog – a catalog of abstract "concept domains" that represent a collection of possible meanings. Concept domains are intended to represent the intended meaning of a field on a form, a column in a database, a field in a message, etc. The CTS2 specification focuses on enumerated concept domains – fields that represent discrete collections of "meanings", each of which is represented by a permissible value. A service would implement this section to provide a list of generic fields that would be used in data interchange.

Concept Domain Binding – the coupling of a concept domain with a value set, where the value set provides a list of possible meanings that can be used in a concept domain in a particular use case or context.

Map Catalog – a catalog of "maps" - collection of rules that allow human or machine assisted transformation between the codes in one value set or code system and those in a second. A service would implement this section to publish information about which rule sets are available, which code systems or value sets they map between, their intended purpose, how often they are published, what formats they are in, etc.

Map Version – an instance of a map, including the specific value set definitions that and code systems that they are based on and the actual rules. A service would implement this section if it wished to publish the content of maps. This section includes an optional sub-section (**Map Resolution**) that provides access to the machine aided interpretation of map versions – a service that does the actual map transformation.

Statement - a bridge between the information contained in the various sections described above and the actual assertions made by the information providers. A service would implement this section when it needed to provide additional provenance about assertions made in catalogs or resource versions including what was actually said, how it mapped to the CTS2 service representation, the provenance of the assertion, etc. **Statement** is also intended to act as a bridge between the structured CTS2 model and simple subject/predicate/target systems as represented by OWL and RDF.

This specification is also intended to support a number of functional areas including:

Read – direct access to the contents of a resource via URI, local identifier or, where applicable, a combination of an abstract resource identifier and version tag (e.g. SNOMED-CT / Current version)

Query – the ability to access, combine and filter lists of resources based on the resource content and user context

Import and Export – the ability to import external content into the service and/or export the contents of the service in different formats

Update – the ability to validate load sets of changes into the service that updates its content

History – the ability to determine what changes have occurred over stated periods of time

Maintenance – the ability to create and commit sets of changes

Temporal – the ability to ask questions about the state of the service at a given point in the past (or future).

Specialized – service specific functions such as the association reasoning services, the map entry services and the resolved value set services.

The **Import, Export** and **Temporal** functions are generic – there are no resource specific aspects to these services and, as such, they are defined once in the **Core Service Elements** module. The remaining components have different signatures depending on the structural domain to which they are applied.

The CTS2 specification is subdivided into the following sections:

Core Model Elements – this section defines the data types, building blocks, and basic interfaces that are shared across more than one structural or functional area. All of the remaining sections have dependencies on the **Core** no dependencies between any of the remaining sections that follow. Each of the remaining sections, when combined with the core, can stand by itself.

Code System and Code System Version Catalog Services – this section describes two independent modules: *Code System Catalog Services* and *Code System Version Catalog Services*.

Entity Description Services – this section describes two independent but closely related modules: *Entity Description Services* and *Association Services*.

Value Set Services – this section describes the *Value Set Catalog* and the *Value Set Definition* services.

Concept Domain and Concept Domain Binding Services - this section describes the *Concept Domain* and *Concept Domain Binding* services

Map Services – this section describes the *Map Catalog* and *Map Version* services, which includes the *Map Resolution* service.

Statement Services – this section describes the statement services

With the exception of the *Core Model Elements*, each of the modules described above is specified using the following pattern:

1) Resource Information Model – the first section of the module describes the structure and content of the resource(s) used in this module. This description includes what constitutes the identity of the particular resource, which elements must be present, which are optional and which are computed by the service itself. Identifying and computed components are marked as *read only* to make it clear what aspects of the resource can be modified.

2) Resource Directory and List Model – the *query* function returns lists of resources. There are two purposes for these lists – the first is to summarize the set of resources that have passed the filter criteria and to provide links that can be used to access the details of the resource directly. This type if list is referred to as a *Directory*. The second purpose of these lists is to gather complete images of the actual resource for some secondary purpose. These sets of complete resource images are referred to as *Lists*. Note that the modules will define both types of list (Directory and List) even when the summary consists of the entire resource image.

3) Read Services – the methods that are available for direct access to the resources. These methods come in pairs – one for testing existence of a resource and a second for actually retrieving it. All resources provide URI access. Note that the URI is passed *as a parameter* to the function. The CTS2 specification makes a clear distinction between the HTTP URI that would be used to access the query service and the URI of the resource itself. In no case is the CTS2 service URI to be used as the identity of the resource itself.

4) Query Services – query services all start with a general pattern. The service provides a URI of type *DirectoryURI* that represents *all* of the instances of the particular resource that is known to the service. This URI represents both active and inactive resources and, if the Temporal compliance profile is supported, represents the all possible service states. The query service then provides a number of generic and structural specific methods, each of which takes a *DirectoryURI* as input and returns another *DirectoryURI* that, when resolved, returns the result of applying the filter to the input URI. Operations are also available for the union, intersection and difference of resource instances. The query services then provide two additional methods, *resolve* and *resolveAsList* which respectively return *Directories* and *Lists* (as described above).

5) History Services – history services consist of several common methods to access and query change sets along with three additional methods – one to return the earliest known state of a resource, a second to return the current state along and the third to return an ordered list of states. Resource states include historical information about what changed, who did it, when, etc.

6) Maintenance services – each module will have a method that allows the creation of the minimal valid resource (identity and required fields) and a second that allows modification of a resource, which allows the addition and update of non-identifying, non-computed content. The services also include generic methods for creating, querying, committing and rolling back change sets. The CTS2 specification requires the following sequence in order to make a change to the service state through a maintenance service:

a. Create a new change set

b. Make one or more changes to one or more resources, providing the URI of the created change set

c. Update any additional provenance information on the change set

d. Commit the change set

URI Persistence

As stated above, the CTS2 specification is based on the RESTful Architectural Style. All resources defined in the CTS2 specification have identity that is expressed as one or more Universal Resource Identifiers (URIs). The specification recognizes that while ideally, each resource would have exactly *one* identifying URI, this is not practical in a loosely coupled environment. Various communities have their own identification schemes and, even when this isn't the case, identifiers are frequently duplicated because one group doesn't know that an ID already exists. The CTS2 model requires that a service implementation assign a single "preferred" URI to each resource that is valid in the service context, but the service should resource retrieval via any valid identifying URI known to the service.

There are couples of issues, however, that need further clarification. The first is that of *DirectoryURIs* – URI's that, when resolved, represent the result of a partial or complete query. First, it should be noted that these URIs are service specific. There is no assumption that a *DirectoryURI* that was created in one service instance will be applicable in a second. The second issue involves URI persistence – (a) how long a URI is valid and (b) can URI's ever be reused. We address each of these issues below.

DirectoryURI Validity

There CTS2 PIM does not require (a) that a given directory have a unique URI and (b) that DirectoryURIs remain valid over an extended period of time. A query service may return a different DirectoryURI for the "allResources" directory every time it is accessed. This also means that a service can determine that a previously supplied DirectoryURI at any time subsequent to its issue. A service, however, cannot *reuse* a DirectoryURI unless the URI has an identical meaning. A service may, for instance, return URI "A" in response to the *allCodeSystems* query. At some time in the future, it may refuse to honor resolve requests on URI "A". It *may*, however, return "A" again in a subsequent response to the same query – even if, in the interim, additional code systems have been made known to it. It may *not*, however, return URI "A" in response to any other query – either about code systems or other resources. Similarly, lists and directories involve the notion of iteration – "pages" as it were. A service may return URI "B" representing page 2 of a specific query , may invalidate "B", may subsequently return "B" representing the same page of the same query, but it may never return "B" to represent a different resource, query or page within a query.

Note also that CTS2 PSMs and/or service implementations may choose to provide additional constraints with respect to URI persistence. PSMs may be created that require that Directory URI's persist indefinitely and service instances may wish to offer various guarantees about the minimum time of URI validity.

6.3 How to Read This Specification

The initial **six chapters** of this specification are *informative*, pertaining to the document itself, providing a general introduction and the purpose of common terminology services, identifying the submitter, outlining the business case for the specification, and providing discussion related to how the specification addresses the RFP (this chapter).

Annex A describes the existing work. (*informative)*

Annex B provides the submission inventory of files. (*informative)*

To adequately and effectively represent the specification, separate PIM component documents (*normative)* are provided:

Core Model Elements Document describes data types, building blocks, abstract resources and abstract service model

Code System and Code System Version Catalog Services Document describes services for representing and maintaining a catalog of code systems and/or code system versions.

Entity Description Services Document describes the model and set of services for representing collections of assertions about classes, predicates (properties) and individuals.

Map Services Document describes services for representing and maintaining a catalog of maps between sets of entities as well as the rules and content.

Value Set Services Document describes services for representing and maintaining a catalog of value sets, their corresponding definitions and their resolution.

Concept Domain Catalog and Concept Domain Binding Services Document describes services for representing and maintaining a catalog of concept domains (Data Element Concepts) and their associated bindings (Data Elements).

Statement Model and Services Document describes services for representing the minimal subject, predicate, and object model used in RDF.

Included with this specification are the following PSM artifacts:

CTS2 REST PSM includes one XML schema per component and WADL to support REST.

- Functional Profile conformance points: Read, Query, Import, Update, Maintenance, Temporal.
- Structural Profile conformance points: Code System, Code System Version, Entity Description, Association, Value Set, Value Set Definition, Value Set Resolution, Concept Domain, Concept Domain Binding, Map, Map Version, Statement.

CTS2 SOAP PSM includes same schema, but functionally is invoked via SOAP procedure calls.

- Functional Profile conformance points: Read, Query, Import, Update, Maintenance, Temporal.
- Structural Profile conformance points: Code System, Code System Version, Entity Description, Association, Value Set, Value Set Definition, Value Set Resolution, Concept Domain, Concept Domain Binding, Map, Map Version, Statement.

6.4 Existing Artifacts

The PIM described below is based on and generalized from existing work on terminology service interfaces that have been actualized as a core piece of enterprise level vocabulary service infrastructure within the NCI caBIG®. Reference to this existing work is included in Annex A. It is anticipated that this PIM and PSM will serve to evolve and provide a standards based implementation for the continuing work at NCI caBIG®.

6.5 Statement of Proof of Concept

Mayo Clinic has been developing tools to support and validate portions of the specification:

- LexEVS is a collection of terminology service interfaces that provide users the ability to store, manipulate, and query controlled terminologies and ontologies.
- Parts of the model presented in the specification were implemented in LexEVS.

A Description of Existing Work

LexEVS has become a mission critical infrastructure for the National Cancer Institute (NCI) since it provides caBIG® and CBIIT with runtime access of the base semantics that under lays all NCI data semantics. For the past five years the LexEVS team at Mayo Clinic has been evolving LexGrid/LexBIG/LexEVS to meet the needs of the semantic community.

The LexGrid Model is Mayo Clinic's proposal for standard storage of controlled vocabularies and ontologies. The LexGrid Model defines how vocabularies should be formatted and represented programmatically, and is intended to be flexible enough to accurately represent a wide variety of vocabularies and other lexically-based resources. The model also defines several different server storage mechanisms. This model provides the core representation for all data managed and retrieved through the service, and is now rich enough to represent vocabularies provided in numerous source formats including:

- Open Biomedical Ontologies (OBO)
- Web Ontology Language (OWL), e.g., NCI Thesaurus
- Unified Medical Language System (UMLS) Rich Release Format (RRF), e.g., NCI MetaThesaurus

Once disparate vocabulary information can be represented in a standardized model, it becomes possible to build common repositories to store vocabulary content and common programming interfaces and tools to access and manipulate that content. The HL7 Common Terminology Services (CTS) and LexBIG API as developed for the Cancer Biomedical Informatics Grid (caBIG®) initiative are two examples of APIs able to query information stored in the LexGrid Model.

LexEVS is the convergence of LexBIG and EVS services into a collection of programmable interfaces that provide users with the ability to access controlled terminologies supplied by the NCI Enterprise Vocabulary Services (EVS) Project.