V3_TEMPLATESPEC_NORMATIVE_2018JAN

To be published as Normative Standard, planned for January 2018

HL7 Templates Standard: Specification and Use of Reusable Information Constraint Templates, Release 1

January 2018

HL7 Normative Standard

Sponsored by:
Templates Work Group

Acknowledgments

The authors of this document wish to recognize the following participants who contributed their time and expertise to the development of this guide.

Copyright

Primary Editor/ Co-Chair	Dr. Kai U. Heitmann Heitmann Consulting and Services, Gefyra GmbH, HL7 Germany hl7@kheitmann.de
Co-Chair	Mark Shafarman Shafarman Consulting mark.shafarman@earthlink.net
Co-Chair	John Roberts Tennessee Department of Health john.a.roberts@tn.gov
Co-Editor	Lisa R. Nelson Life Over Time Solutions, LLC Lisarnelson@cox.net
Contributor	Andy Stechishin CANA Software & Services Ltd. andy.stechishin@gmail.com
Contributor	Keith W. Boone GE Healthcare keith.boone@ge.com
Contributor	Rik Smithies NProgram Ltd rik@nprogram.co.uk
Contributor	Alexander Henket Nictiz henket@nictiz.nl
Contributor	Maarten Ligtvoet Nictiz ligtvoet@nictiz.nl
Contributor	Sean McIlvenna Lantana Consulting Group sean.mcilvenna@lantanagroup.com
Contributor	Abderrazek Boufahja IHE Europe Development abderrazek.boufahja@gmail.com

Table of Contents

1. Introduction

1.1. Purpose

HL7 V3 provides a global framework for exchange of healthcare information as documents or messages or services by providing a framework for constructing instances of data according to agreed definitions in a standard fashion. The globally agreed definitions are often fairly general in nature due to the intention of their scope or the requirements to be globally suitable, and a framework for making additional rules for more specific knowledge models is required. Templates are used to provide this framework within the context of HL7 V3 and especially the HL7 Clinical Document Architecture.

Besides HL7 V3 artifacts there are other HL7 standards that can profit from using the same template framework, mainly v2.xml (the XML representation of v2 messages [v2xml]), or FHIR [fhir]. In fact the template framework described here can be used for any XML based data representation.

This document arises from joint work of the HL7 Templates Work Group, a Special Interest Group (SIG) at the time, and the Template Specifications project of the Modeling and Methodology Work Group, a Technical Committee (TC) at the time. Templates have had a long genesis from the first identification of their need, and a series of draft documents have already been published, as a more complete understanding has evolved.

This document builds on the existing template specifications or drafts, e.g.

- *HL7 V3 Templates: HL7 Version 3 Standard: Specification and Use of Reusable Constraint Templates, Release 2* published in February 2008 (which is replaced by this specification),
- the recently balloted *HL7 Templates Registry Business Process Requirements Analysis Informative Ballot, Version 2*, published 2013,
- various implementation guides published by HL7's Structured Document Working Group,
- various implementation guides published by IHE and by other governance groups outside HL7 International and
- other new work in the general interoperability methodology space to provide a complete specification for templates.

1.2. Scope

This implementation guide describes how templates are specified, registered, used and exchanged. It describes how to exchange template definitions using an Implementable Technology Specification (ITS). It covers template characteristics, versioning of templates and definitions of relationships that exist for templates. It also explains how to use template references and validate them in XML instances.

It focuses primarily on HL7 V3 and especially the Clinical Document Architecture, but this specification has already been successfully used for the XML representation of v2 messages [v2xml], FHIR [fhir] and other XML-based artefacts.

1.3. Ballot Status of the Document

This document has some need of refinement and improvement based on the experience of implementers during the DSTU period. The authors intend to use the "dot" release process to capture the major changes and will address all comments received during the DSTU period in the Ballot for Normative Standard, in 2016. Comments can be made at http://www.hl7.org/dstucomments.

Specific opportunities for application of Template include FHIR Resource Profiles, v2.xml Profiles, and vMR-CDS Profiles. As work in these areas progresses during the DSTU period the interactions with this standard can be defined and added as DSTU comments.

1.4. Audience

The audiences for this implementation guide are architects and developers of healthcare information technology (HIT) standards and architects and developers of applications/systems anywhere in the world that exchange clinical data. Business analysts and policy managers can also benefit from a basic understanding of the use of templates.

1.4.1. Prerequisite Knowledge

- HL7 V3 Interoperability Standards (Messaging)
- Clinical Document Architecture CDA Release 2
- SNOMED (www. http://www.ihtsdo.org/snomed-ct)
- LOINC (http://loinc.org)
- UCUM (http://unitsofmeasure.org)
- OIDs (http://www.hl7.org/oid)

1.5. Organization of the Guide

This implementation guide is organized into the following chapters

- Chapter 1 is this **introduction**, defining scope, audience and prerequisites.
- Chapter 2 defines what **templates** are, how they function, their characteristics, and how they are defined.
- Chapter 3 covers **template versioning** to describe the life cycle of templates and the corresponding metadata used to identify and characterize versions of a template.
- Chapter 4 describes possible **relationships** between various templates and other artifacts like models.
- Chapter 5 provides **use cases** which demonstrate the creation, use, maintenance and governance of templates.
- Chapter 6 covers questions and answers around **management and governance** of templates.
- Chapter 7 defines the **Template Definition Exchange ITS**, an implementable technology specification for the exchange (and storage) format of template artifacts. It also covers the **use of templates** to generate instances which conform to the template definition and tools and techniques for **validating template conformance** in generated instances are discussed.

- Appendix A is a **glossary** defining the most important terms when discussing templates.
- Appendix B includes a lookup reference for **acronyms and abbreviations**.
- Appendix C summarizes **conventions and best practice for template documentation** with examples from various governance groups.
- Appendix D shows the constraint syntax used in various implementation guides and maps **conformance statements** to the template definitions in this guide.
- Appendix E records **additional references** which augment the topics covered in this document.

1.6. Conventions

This guide adheres to the following conventions:

- Examples are all non-normative
- If there are any differences between examples and the normative text, the examples are in error

Reference Implementation

The Template Definition and Exchange ITS Format described in this guide is not only a specification but has already been implemented successfully and is used in real projects dealing with template creation, maintenance and use. Most of the described features and functionalities have undergone a proof-of-concept phase and were determined by typical practical use cases and refined repeatedly by experiences from practice.

1.7. Contributions

In addition to the editors and authors mentioned above the following groups gave important input to this document.

- HL7 Working Groups
  - Templates (discussions running from October 2012 to October 2013)
  - Modeling and Methodology
  - Tooling
  - Structured Documents
- IHE (Integrating the Healthcare Enterprise)
- National Institute for Health IT (Nictiz), The Netherlands
- The ART-DECOR® Expert Group (ADEG), ART-DECOR® Open Tools UG (ADOT)
- HL7 Austria
- Austrian ELGA infrastructure project
- HL7 Germany (Interoperability Forum)
- IHE Germany
- The European epSOS Project (European Patients - Smart open Services)
- The European Semantic Health Net Initiative

2. Templates and Template Versions

2.1. What is a template?

A template represents a formal definition of a set of constraints on a model, e.g. an HL7 R-MIM. The constraints are expressed as a formal definition, for example as a restriction on the attribute value domains, cardinality, or optionality of the information model when it is applied to a particular use case or context. The definition may be expressed in one or more human readable languages, or as a formal definition such as a model. For example, an HL7 V3 template is an expression of a set of constraints on a specific message R-MIM. Another example is a CDA header level template, which is an expression of a set of constraints on the CDA R-MIM document header.

A template is logically split into

- A metadata part such as an identifier, versioning information and a definition of the purpose etc. and
- A body part that contains the actual set of constraints (design).

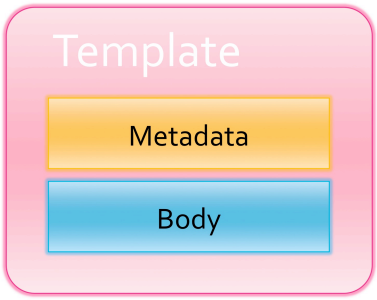


Figure 1: Template parts

Templates are used to further define and refine these existing models to specify a narrower and more focused scope. They function to apply additional constraints to an instance of data, or a portion of an instance of data. The definition represented in the template includes one or more implementation-specific representations, which can be used to validate instances in a particular context. As an example, the blood pressure template used by an intensive care-specific template would be more specific than the blood pressure template used in other situations.

Templates often provide additional definitional materials that describe how the information models are applied to very specific use cases or contexts. This material needs to be consistent with the underlying model fragments to which the template applies. It is typical for templates to be defined as reusable modules, e.g. a template that contains other frequently used templates. As an example, a vital signs template would typically include a blood pressure template.

2.2. What is a template version?

Templates, like other specification artifacts, have a life-cycle. They come into existence with an initial design, and then over time, their design is modified and updated to match the continually evolving environment in which they are used.

Each version of a template represents a distinct design for the associated definition. Metadata associated with the version describes the precise purpose and use of the particular definition for the template.

A governance group creates an initial design for a template, and after using the template, it reviews the design to determine if it is necessary to develop and release a new, enhanced or corrected version of the initial template. The revision history of a template

is managed through the creation of “versions”. Each version of a template represents a different design in a template’s evolution.

Furthermore, it must be possible to mark a version of a template as being in a certain state of maturity, so that its use in other designs or its use for content creators and content consumers can be controlled. Template versioning supports the management of specification artifacts at the design level. It is important that each version of a template contains metadata that associates it with the intended purpose of the template that is bound to the template identifier and metadata that uniquely identifies the version itself.

While multiple versions of a template can concurrently be useful, because they support instances of the template or they are used by systems that are not yet aware of an improved version of a template, the most recent version is thought to be the “current” version and should be used in creating new instances. When a different use case or entirely new purpose needs to be addressed, a new template is created. Versions permit refinement of a template’s design, not for making major changes to the purpose.

Each template comprises a set of versions, but each version is handled as a distinct and complete artifact, with metadata and a specific design body. Each template version belongs to a set of versions for one template. Within the set, a version’s relationship to other versions is expressed by an implicit or explicit *template version relationship*. This is because also parts of the metadata may change over time.

Please note that a template registry/repository provides a list of templates and presents the set of versions appropriately, e.g. summarizing all versions under the name of the template (see figure). The concept of a template (in contrast to a template version) is a *virtual* artifact, i.e. a template only exists as embodied in a version of itself. Every template is a template version, even if there is only one version in existence.

While computable relationships between template version and recorded in the metadata of a template version in a repository may assist the group responsible for the registry (i.e. governance group) in determining their organization and indexing of template versions, the cataloging of template versions is primarily a human endeavor.

In the figure below, determining that all three depicted templates support "intention A" likely was a human decision, at least for the statement “Y is a version of X with the same intention A”.

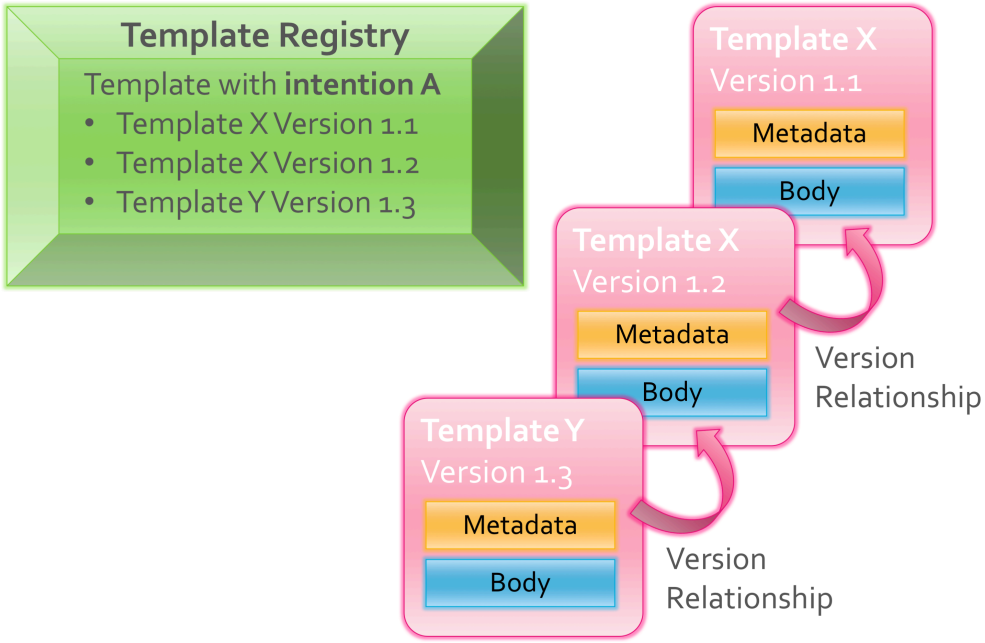


Figure 2: Template versions are handled as complete artifacts including metadata and body. They have an implicit or explicit version relationship. Template Registries/Repositories may list all versions of each registered template, for example, showing the versions as a hierarchical list. Please note that the templates may have different identification, e.g. X or Y (different ids or names), notwithstanding that they belong to the same set of versions (i.e., have the same intention) due to the explicit version relationship.

2.3. Why are templates needed?

In order for data to be interoperable, it must conform to expressions of the base object model which is commonly understood. In HL7 V3, instances of data will firstly conform to the HL7 Reference Information Model (RIM). The data may also conform to other models (R-MIMs) which are derived from the RIM and meet specific use case requirements. Templates are a documentation of the conformance rules associated with various models. The rules are applied when generating, validating, or processing messages, documents and services.

Other models may exist at many levels of abstraction. This includes static models, which describe a set of constraints on the RIM, to clarify how some real world domain is properly described within the framework of the RIM. However these static models are generally rather broad and generic. Other more detailed models may conform to these static models, refining the generic concepts to more specific ones such as particular laboratory tests.

Templates are used to define the structures of these more constrained concepts within the domain. A template's design can be reused in multiple different contexts, wherever the real-world concept they describe occurs. A template used to describe a finding in a laboratory report may also be used to describe supporting evidence for a diagnosis in a problem list. Templates are used to express conformance rules needed to further constrain the broader static model and provide the level of detail necessary to ensure interoperable data is created and shared.

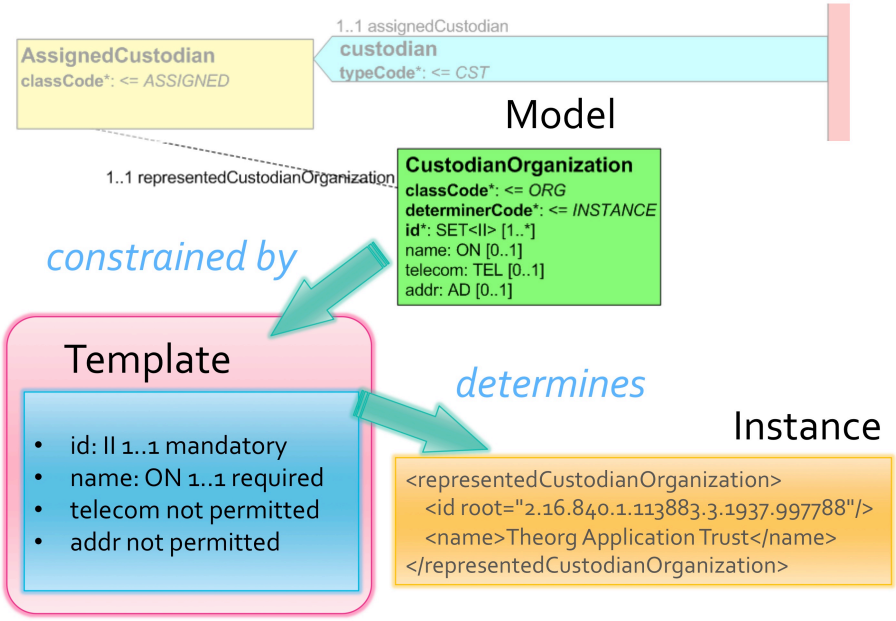


Figure 3: Templates typically constrain underlying models, e.g. a model fragment defining a generic Custodian Organization is further restricted by a template that restricts the identifier to be exactly 1 mandatory id, exactly one required name and does not allow telecom or addr in the instance. The template definition determines how a conformant instance is structured and what content can be used within to populate the instance. An instance may be validated against the template definition (and the underlying model).

2.4. Why is template versioning needed?

Template versioning is needed to enable template designs to evolve over time.

Template versioning enables template designers to control and shape the conformances that make up a template's design over time tailoring the design to fit the template's intended purpose.

2.5. How does template versioning work?

Each template version is associated with a particular template. The template – as a whole – has a mandatory globally unique, non-semantic, identifier. The identifier serves as the identifier of the original intent of the template and as the identifier of the set of versions that represent the template over time.

Template versions have a mandatory timestamp (date and optional time), called the "effective date". The date can be seen as the point in time when the template version "came into being", i.e. was recognized as existent by the governance group. Use of the template prior to this date would be considered an invalid use of the template.[1]

It can be expected that a template designer always knows when a version of a template is intended to come into being, regardless of the status of the template, i.e. whether it is a draft or an active (officially published) template. The effective date functions in

[1] Please note that there are also other dates like the official release date and the expiration date. Refer to section 7.3 for details.

conjunction with the other version identification properties to uniquely identify a template at the version-level.

In addition to the template identifier and the effective date, templates have a status code. The status code reflects the status of the design. It tells if a template version is still in draft, is considered active, is under review, or is retired and no longer recommended for use in new designs.

A template version may also have a version label. The version label is linked to the effective date and is used to add a name label that helps humans distinguish one version of a template from another.

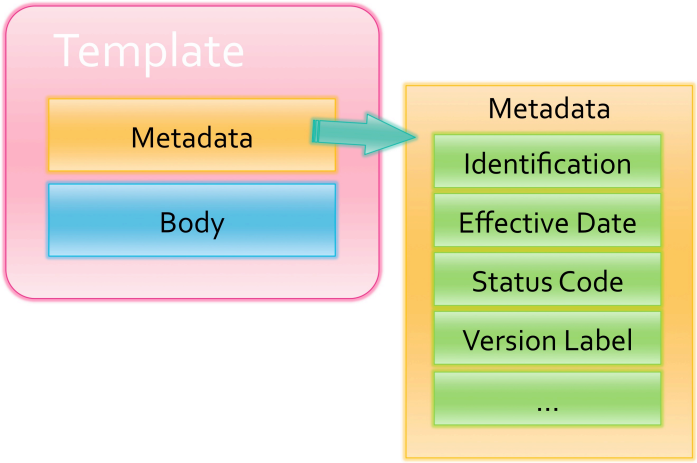


Figure 4: Template metadata (partially)

When "naming" a template version, some governance groups use the combination of the template identity and "effective date". Other groups prefer to use the template identity and the version label.

2.6. Why are templates helpful?

To facilitate interoperability, instances of data, either documents, messages or service payload fragments, need to conform to some reference model, like the HL7 V3 RIM, or some set of reference models. Standards such as HL7 CDA, define a document specific constraint on the RIM, or a message specification defined for a particular interaction. Standards such as HL7 Consolidated CDA, further constrain CDA, adding refinements to ensure that portions of the data conform to additional, even more specific models. Templates provide the mechanism to refine broader models at lower and more precise levels.

Templates can be seen as an "instruction" or "set of instructions" for the proper creation of an instance of data for a specific use. When an instance of data conforms to a template, the instance of data is said to "conform to the asserted template". Template ids in documents or messages aid data validation and can aid processing. For example, they can enable more efficient parsing by avoiding deep look-aheads, and they can also allow for quick routing. However, an instance of data must always carry its full semantic meaning independent of the template used to guide its creation and validation. The full meaning of the data must be understandable in the instance without making drawing any inference from the presence of the template assertions.

2.7. How does a template design get defined?

A template design can be defined in a number of ways. It can be created as human-written specifications or it can be developed using modeling tools which automate the process of creating designs with greater precision and consistency. Templates can be published in human-readable or in machine-readable forms. They can be defined in documents and managed in documents where they are defined, or developed using modeling tools developed specifically for the creation of templates and managed in repositories deployed through digital technology.

Recall, each template represents a set of constraints on a model and constrains a base model, such as HL7 CDA R2 which is general and designed to serve a broad range of uses. In the process of designing templates, it may be determined that templates designed for one purpose are usable for another purpose. For example, a section level template defined for the HL7 Continuity of Care Document may also be usable for an HL7 Discharge Document or an IHE Patient Care Coordination Patient Summary document. A template can be used or reused for any purpose which it is fit to serve.

To explain how templates get defined, it is helpful to understand the roles played by various "actors" involved in the processes associated with defining templates. These actors may represent an individual, an organization, or a system contributing a particular function.

2.7.1. Template Designer

A template designer is a person who develops the set of conformance statements which make up a template. While templates can be designed from scratch, a template designer considers the requirements for the template based on its intended purpose. A template designer may make use of previously defined templates in the construction of a new template. A template designer also may create a new template by further constraining a previously defined template. The refinements included in the design of a new template can at times be subtle, such as limits on the cardinality of an attribute or stricter expectations on the level of conformance (should versus may).

Choices about the reuse of existing templates or creation of new templates are what a template designer considers when developing template designs. The underlying design of the templates plays an important role in creating interoperable and meaningful data.

The development of high quality template designs requires a great deal of specialized knowledge about the subject matter where the template will be employed (domain expertise), the applicable interoperability standard, and the universe of existing template designs.

2.7.2. Governance Group

Template designers often create their designs within the context of a community. Often these communities are involved with standards creation, like HL7 and IHE, but they don't have to be. A template governance group establishes the rules and best practices associated with their creation and management of templates. The group establishes practices that are used to manage the intellectual property rights associated with the templates they create and use, and provide for governance functions. Some governance groups develop templates with the intent to register and share template designs. Others intent to maintain their template definitions as private works.

2.7.3. Template Repository

A template repository houses the design versions of multiple templates.

2.7.4. Template Registry

A template registry tracks the existence of templates and their versions from one or more template repositories. It typically represents the point of view of a single governance group and manages the status of the templates and designs based on the practices and policies of that governance group.

2.8. Characteristics of a template and its versions

Each template has a mandatory globally unique, non-semantic, identifier. It serves as the identifier of the guiding intent of the template.

Templates typically have a life-cycle like other specification artifacts. A governance group starts with the original design of a template, and after use or review it may be necessary to develop and release a new, enhanced or corrected version of the initial template. The revision history of a template is managed through the creation of "versions".

Each version of a template represents a different design in its evolution. Metadata exists for each version of a template. The template version metadata permits templates to be managed and tracked within template registries and managed within template repositories. For more information about the functions performed by a template registry and a template repository, see section 7.6.

A variety of relationships exist between templates and other templates, and between a template and its versions. A template may be defined to inherit all the conformances of a previously defined template. A template may incorporate another template into its design by including that prior design as a component. A template could have an identical structure, but be used for a different purpose. These all represent different ways one template may be related to another. Backward compatibility is a relationship between different versions of a particular template. It implies that the design of a subsequent version is compatible with the design of a prior version. It is a complex notion which considers aspects of the design as well as issues of template governance. For template relationships see section 3.

2.9. Template Key Metadata and Additional Metadata

The following metadata exists for each design or version, as noted, of a template. The template identifier, business name, status information (times, status code) and versioning metadata is considered to be "key" information of a template, while other items (description, relationship, context, examples, etc.) provide additional information.

2.9.1. Identifier

Each template design is identified using a unique identifier. The identifier is a mandatory globally unique, non-semantic identifier and is associated with the unique intent or purpose and acts as the primary identifier of the template definition. It serves as the identifier of the original intent of the template and as the identifier of the "family" of design versions which may exist to express the template over time.

2.9.2. Name

A template has a required (business) name for the template as a secondary identifier.

Please note that there is no guarantee that the name is globally unique but it shall be unique within a governance group. The name identifies the template. For discussions within a governance group a template can be referenced by name, in all other cases, the id of the template shall be used.

2.9.3. Effective Date

The template has a mandatory timestamp (date and optional time) after which the template existed regardless of its state (e.g. draft, pending, active, etc.). The date can be seen as the point in time "when the template came into being" or when it became "in effect" for the governance group. Use of the template prior to this date would be considered an invalid use of the template because the template had no definition prior to the effective date.

The rationale behind a mandatory effective date as a version identification property is that it can be expected that a template designer always knows when "his" template first came into being based on the processes followed by the governance group. It is important to distinguish between the effective date (in effect for the governance group) and the "official release date" (see section 2.9.8).

2.9.4. Version Label

A version of a template may contain an optional human readable version label. The version label provides a more readable name that helps humans identify the version of a template. For this reason, the version effective date and the version label of a template have a one-to-one relationship.

Example: A template X of version 2014-01-02 may carry a version label "v1.0" and there shall be no other template X with any other version date carrying the same version label.

Some governance groups may make this label to be mandatory; IHE for example recommends the use of a version label as a "version identifier" which is used in conjunction with the template identifier to uniquely identify a version of a template.

It is recommended to always use alphanumeric string, such as "v1.01" or "V1" for version label definitions to avoid any confusion with OID-fragments.

2.9.5. Version Governance

Changes in design requirements may lead to the need to revise the currently published version of an active template. As long as the requirements do not change the intent/purpose of the template, the new design can be created as a new version of the original template.

Further, if applications using the template, both as content creators and content consumers, do not need to make any substantial changes to their code to process data conforming to the new design, then the new version is said to constitute a **minor revision**.

If applications using the template, both as content creators and content consumer, will need to make substantial changes to their code for processing data conformant to the new design, then the new version is said to constitute a **major revision**. Some

governance groups may choose to indicate major or minor revisions in the naming scheme used for the version label.

Table 1:Recommended properties of a "version" of a template

Version of a template
Keeps the same identifier as the former template
Shall have a effective date to mark its creation
Shall have a status code
May have a new version label which must have one-to-one alignment with the effective date

When incorporating revisions, the governance group may decide whether a new version is required, as either a major or minor revision, to replace the former template design, or if the former template design can be modified to correct the errata. In some cases, error correction may not lead to the need to create a new version of a template design. If the intent of the template requires revision, it may or may not necessitate creation of an entirely new template. These types of decisions are determined by the group governing the template.

2.9.6. Status

Every version of a template has a status. The status indicates the level of maturity of the design and may be used to manage the use of the design. While different template governance organizations may establish additional status states to support local template management practices, or chose to reduce the number of statuses it uses, the follow set of statuses have been defined:

Table 2: Status states for a template version

Status code	Description
draft	Design is under development (nascent).
pending	Design is completed and is being reviewed.
active	Design has been deemed fit for the intended purpose and is published by the governance group.
review	Design is active, but is under review. The review may result in a change to the design. The change may necessitate a new version to be created. This in turn may result in the prior version of the template to be retired. Alternatively, the review may result in a change to the design that does not require a new version to be created, or it may result in no change to the design at all.
cancelled	A drafted design is determined to be erroneous or not fit for intended purpose and is discontinued before ever being published in an active state.
rejected	A previously drafted design is determined to be erroneous or not fit for intended purpose and is discontinued before ever being published for consideration in a pending state.

Status code	Description
retired/ deprecated	A previously active design is discontinued from use. It should (deprecated) or shall (retired) no longer be used for future designs, but for historical purposes may be used to process data previously recorded using this design. A newer design may or may not exist. The design is published in the retired/deprecated state. A deprecated template design may be included in current designs to enable smooth transitions and to indicated that it may go away in the future. Some governance groups distinguish between retired and deprecated: deprecated is handled as "this template *should* no longer be in use (contemplating)", while retired means it "*shall* no longer be in use". A governance group need to define the policy regarding retired and deprecated.
terminated	A design is determined to be erroneous or not fit for the intended purpose and should no longer be used, even for historical purposes. No new designs can be developed for this template. The associated template no longer needs to be published, but if published, is shown in the terminated state.

It must be noted that from a governance perspective it may be appropriate to use the full range of status codes. From an implementer's perspective only the status codes "draft", "active" and "retired" may be of interest. A governance group typically makes use of the whole template status machine but may decide to only publish templates with certain status codes, e.g. "active" and "retired" only.

The following figure shows the state transition diagram for template versions.

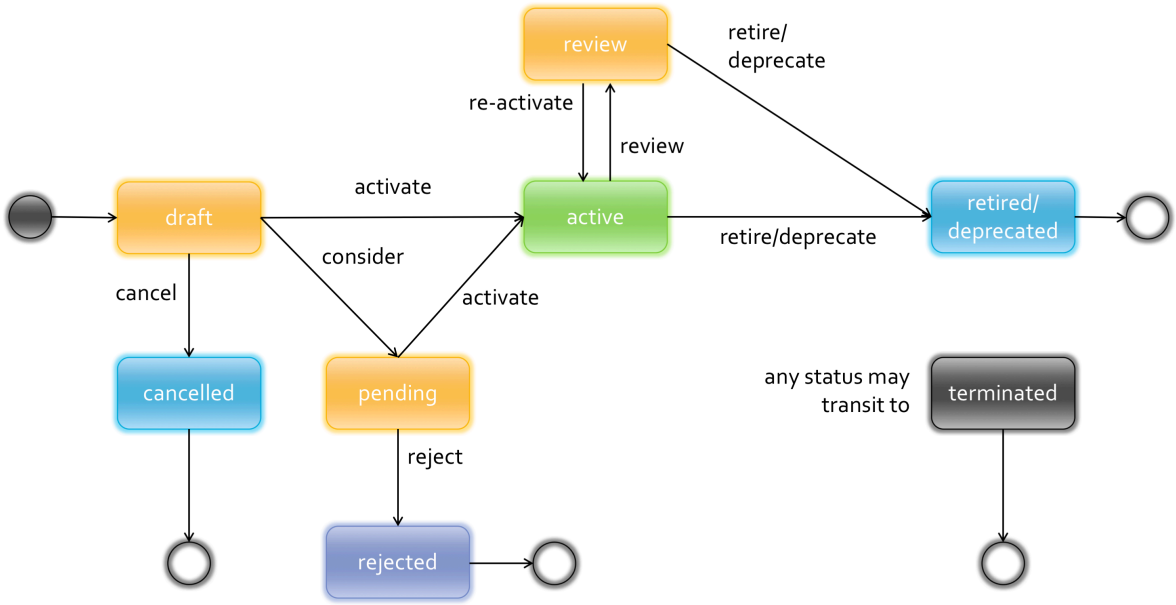


Figure 5: Template version state transition diagram (see text for status "terminated")

It is recommended that for historical template definitions (that do not have a status yet) a status code "active" is assumed/assigned unless stated otherwise.

2.9.6.1. Draft

Each template design begins as a draft design. It is assigned its own metadata which makes it a version of the associated template. The effective date records the creation date for this version. It is assigned to the design when it is first drafted.

When a design is in the draft state its metadata and conformance constraints may be incomplete. The status of each design allows users to be aware of the ongoing template management process. A governance group may decide to draft more than one design for a template, thus there could be multiple versions of a template in the "draft" status, i.e. draft designs of the same template which are distinguished by different effective dates and managed as different versions of the template.

As long as a template version has the status of "draft" the design may change, even substantially. Once the status advanced to "active", no further changes can be made to this version of the template's design without undergoing a formal review.

2.9.6.2. Pending / Review

The "pending" status indicates that a newly drafted design is undergoing consideration before being activated. The "review" status indicates that a previously active design is undergoing review.

The "pending" status indicates a design is in transition from the "draft" status toward the "active" status. For standards development, this typically represents a (public) comment phase. Review is set for active templates that need to undergo a review. The "review" status indicates a design is being examined to process comments requesting amendments to the design or purpose of the template.

After the review phase the template may become re-endorsed (=active), retired or terminated.

2.9.6.3. Active

The "active" status indicates a design that it is approved for use. The governance group has decided that this version of the template is fit for the intended purpose. When a design moves into the "active" status the effective date of the version does not change.

2.9.6.4. Cancelled

If the designers of a version of a template determine the design is a bad idea or is not suited for the purpose of the template or it is determined that the design is a duplicate effort, and the template was never published for review, the design can be moved into the status "cancelled". Other such situations include if one governance group decides to abandon their template (and its id) and, for example, join another governance group or if one governance group decides to use a template from another governance group. This is the end of the life cycle of this version of the template. However, new designs for this template may be created.

2.9.6.5. Rejected

During initial consideration of a design, before it ever becomes "active", the governance group may find a template design inadequate or erroneous, and therefore it becomes rejected. In this case, the expiration date (end date) is set (to the current date) indicating

that the use of this template is no longer recommended. This is the end of the life cycle of this version of the template. However, new designs for this template may be created.

2.9.6.6. Retired / Deprecated

If a governance group determines that a previously "active" template design should (deprecated) or shall (retired) not be used any longer, the expiration date is set and the status of the design is set to "retired", or "deprecated" respectively. This version of the template should/shall no longer be used for new instances, nor should/shall it be used to define new designs (template designs with containment or inclusion). It may continue to be useful to enable access to a retired design so that references to that version of the template can be resolved. A retired design should only be used for a historical validation (i.e. when the creation data of the instance falls between the *effectiveDate* and *expirationDate* of the retired template version). This is the end of the life cycle of this version of the template. However, new designs for this template may be created.

2.9.6.7. Terminated

If an already published template design was found a bad idea to create, duplicate effort, etc., the status can be moved to "terminated".

This is the end of the life cycle of this version of the template. It also is the end of the life cycle for the template as a whole. No new design versions will be created for this template. A "terminated" template should not be used for a historical validation (i.e. when an instance references any version of the template). If a document contains a terminated template design, the instance is invalid from a machine processing perspective (although it still may contain some reasonable or useful clinical information).

2.9.7. Expiration Date

A template may have an optional date at which the design represented by this template becomes stale, and should be reviewed for (clinical) relevance and accuracy. The expiration date is set to indicate that it should/shall no longer be used in the design of other templates (by anyone), i.e. another template design that uses this template by inclusion or containment after this date would be considered bad practice.

2.9.8. Official Release Date

A template may be released (published) by the governance group in any status it might have. However, it may be useful to set an "official" date since when the template is ready for use (trial implementation, production etc.). This is done by populating the optional "official release date".

2.9.9. Description

A description defines the purpose/intent and the scope of the template.

2.9.10. Classification

An optional classification of the template informing about

- The type of the template, i.e. the (root) class constrained by the template, e.g. CDA Section Level Template, Document Level Template etc. (see also section 2.12).

- Possible search tags to easily find templates in large registries, e.g. tagging the template with “blood pressure” or “diabetes regime” would enable a search engine to find the template by this term.
- The format of the template; as of now this is “HL7 Version 3”, but future developments may make other options possible.

2.9.11. Relationships

This covers an optional list of relationships to other templates or model artifacts (see chapter 3).

2.9.12. Context

The optional context of a template describes where in an XML instance the template rules are considered to apply to. From a practical viewpoint templates may have no explicit context. In this case, the template is not “exposed” for external use, but rather used for internal inclusion in other templates (see also section 2.10.2) and thus gets it’s context from the location where it is included / contained.

If the template is intended to be published for external purposes (for containment in other template designs) and for example be triggered by a template id in an instance, it shall have a context.

Please note that template definitions may not only contain a single root class but can also contain a set of constraints only.

Examples:

(1) a full “observation” class set of constraints

(2) only a set of constraints about the element *effectiveTime* to be “used” (included) in templates dealing with timing aspects such as a substance administration.

Template 1 shall have a context, e.g. by specifying a parent node template id.

The context of template 2 is defined by the location of the inclusion of the template, e.g. the substance administration act definition.

Typically, there are three types of context specifications.

Parent node context

The template rules apply to the parent node and all descendent nodes in the instance. This means that the context for validation is: all sibling nodes of any *templateId* element in an instance where the *@root* is populated with the same id as the template id including the parent node tag name.

A typical example is a CDA entry level template *observation*. The definition in the template design includes the name of a root element “observation” and child nodes like @classCode, id, code etc. The root element is parent to *templateId*, all other nodes are siblings to *templateId*.

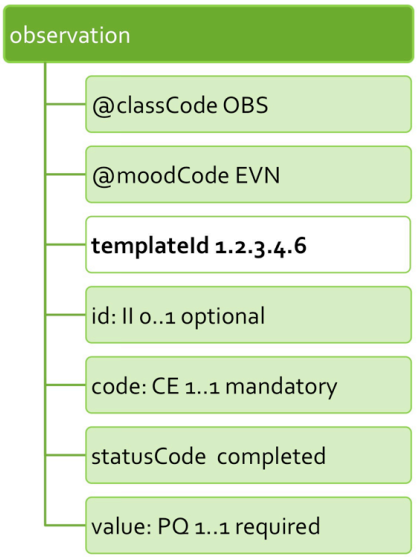


Figure 6: A template definition with parent level context, the scope of the template rules include the root element

The "scope" of the template, i.e. where the constraint rules are applied to, covers the root element and all siblings to the *templateId* in the instance.

```
<observation classCode="OBS" moodCode="EVN">
    <templateId root="1.2.3.4.6"/>
    <id root="2.16.840.1.113883.3.1937.67543"/>
    <code code="1234" codeSystem="1.2.3.4.5.6"/>
    <statusCode code="completed"/>
    <value xsi:type="PQ" value="1.01" unit="m"/>
</observation>
```

Figure 7: An instance fragment and the "area" covered by a template definition with parent level context

Sibling node context

The template rules apply to all sibling elements and descendent nodes in the instance. This means that the context for validation is: all sibling nodes of any *templateId* element in an instance where the *@root* is populated with the same id as the template id. This excludes the parent node tag name.

As an example you may think of a template that lists all attributes and elements of an organization, i.e. *@classCode, @moodCode, id, name, telecom* and *addr*.

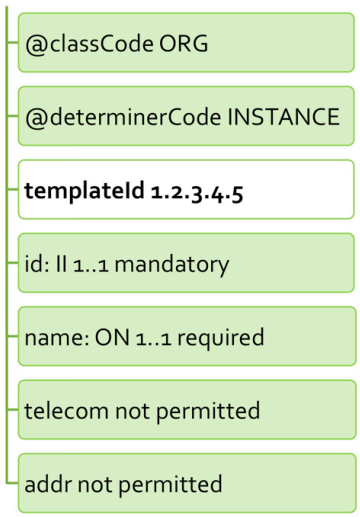


Figure 8: A template definition with sibling level context

By not specifying a root element this list of constraints can be used in the CDA author template as content of the *representingOrganization*, but also in the CDA custodian as content for *representedCustodianOrganization* (re-usability aspect). The template with the list above is defined in sibling context, i.e. it does not have a root element.

```
<representedCustodianOrganization classCode="ORG" determinerCode="INSTANCE">
    <templateId root="1.2.3.4.5"/>
    <id root="2.16.840.1.113883.3.1937.997788"/>
    <name>Theorg Application Trust</name>
</representedCustodianOrganization>
```

Figure 9: An instance fragment and the "area" covered by a template definition with sibling level context

Alternatively, you would have to define one template with *representingOrganization* and one for *representedCustodianOrganization* while the content is likely to be exactly the same.

Pathname specified context

A pathname (making use of XPath expressions) is specified, which allows to activate templates in an instance without the need to have template id elements in an instance.

A pathname context is often defined for CDA document level templates or non HL7 Version 3 XML-based specifications (SOAP, ebXML etc.).

```
<signedData xmlns="http://www.release.nl/805/"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
  wssecurity-utility-1.0.xsd"
  wsu:Id="token_2.16.528.1.1007.3.3.1234567.1_0123456789">
    <authenticationData>
        <messageId>
            <root>2.16.528.1.1007.3.3.1234567.1</root>
            <extension>0123456789</extension>
        </messageId>
        <notBefore>20070128173600</notBefore>
        <notAfter>20070128174059</notAfter>
        <addressedParty>
            <root>2.16.840.1.113883.2.4.6.6</root>
            <extension>1</extension>
        </addressedParty>
    </authenticationData>
    <coSignedData>
        <triggerEventId>QURX_TE990011NL</triggerEventId>
        <patientId>
            <root>2.16.840.1.113883.2.4.6.3</root>
            <extension>012345672</extension>
        </patientId>
    </coSignedData>
</signedData>
```

Figure 10: A non-HL7 V3 instance fragment (a digital signature) and the "area" covered by a template definition with path level context path = coSignedData

2.9.13. Open vs. closed

An indication whether a template is considered to be closed or open. For further discussion on open vs. closed templates see section 2.11.

2.9.14. Examples

It is best practice that a template contains one or more examples or example fragments to illustrate valid (or expose invalid) examples.

2.9.15. Publishing Authority

The authoritative body who has reviewed the template for (clinical) accuracy and relevance, and authorized it for publication. Template registries may require templates to populate this element. It typically contains identifications, names and contact information.

2.9.16. Endorsing Authority

A list of bodies who have reviewed the template for (clinical) accuracy and relevance, and endorsed it for use. It typically contains identifications, names and contact information.

2.9.17. Copyright Authority

A list of bodies who hold the copyright(s) of the template. It typically contains identifications, names and contact information.

2.9.18. Purpose

A text containing the purpose of the template.

2.9.19. Revision History

The free text description describing the changes in this version of the template as compared to the previous version; since template versions are built off of previous versions, the net effect of this field is to function as a comprehensive historical reference of the template, so this field should be populated if the template is not the first version.

2.10. Template Body

The actual template design, also seen as the "body" of the template, is a collection of constraints that ideally describes the structure and semantics of all instance elements. It is typically presented as a template item table, see section 2.14 for an example. The table presents the constraints on the XML elements and attributes. In addition, containment, choice, conditional requirements and vocabulary binding requirements may be defined.

2.10.1. Types of constraints

Constraints express a template's design in several different ways, described in the following section.

In general, the constraints in the design of the template define the properties of a conformant XML element or an XML attribute. In addition, co-occurrences of items or conditional requirements may also be defined. This specification does not describe a generic model constraint language.

2.10.1.1. Data Types constraints

The data type of an element is simply determined by specifying a data type name or a data type flavor name.

Table 11: Examples of Version 3 data type assignments / constraints

Item	Description	Data Type (Flavor)
id	An instance identifier	II
code	A coded element	CE
effectiveTime	A time stamp	TS
validityPeriod	A time interval	IVL_TS
value	A measurement of a physical quantity	PQ
repeatNumber	Repetitions, non-negative numbers	INT.NONNEG
date	A time stamp precise to the day	TS.DATE

Note that the constraint shall be equal to or a valid demotion of the corresponding data type of the underlying model (if present).

2.10.1.2. Cardinalities

The cardinality indicator (0..1, 1..1, 1..*, etc.) specifies the allowable occurrences within a document instance. The cardinality indicators are interpreted with the following format "m…n" where m represents the least and n the most:

- 0..1 zero or one
- 1..1 exactly one
- 1..* at least one
- 0..* zero or more
- 1..n at least one and not more than n
- n..m at least n times and not more than m times.

Note that templates can only constrain cardinality, not extend it (i.e. if a base model such as CDA says something is 1..1, a template cannot change it to 0..*).

This guide uses cardinalities for element and attribute definitions where attribute definitions can be either 0..0 (in XML attribute terms this is called prohibited or not allowed), 0..1 (in XML attribute terms: optional), and 1..1 (in XML attribute terms: required).

2.10.1.3. Mandatory/Conformance

Elements that are defined to be mandatory must be populated with valid data. If an element is not flagged as mandatory, data may be absent, e.g. by using a nullFlavor or the element may be simply omitted if optional.

The conformance expresses whether a system must support an element or not.

The following terms are used in this guide to express conformance criteria.

Mandatory (M)

The attribute is mandatory, i.e. a valid value shall be provided and no null value is allowed. The minimum cardinality is at least 1. This also implies that if the sender has no valid value for such an attribute, the message cannot be sent.

It is indicated as “M” in the conformance column of the template item table, a shorthand for “mandatory” with required conformance.

A sender must support elements with “mandatory” conformance, and a receiver must understand these elements.

Please use mandatory elements with caution as overly-strict requirements may result in not implementable templates or in situations that are not aligned with practice.

Required (R)

The item is required, i.e. a valid value should be provided or if missing a null value is allowed if its minimum cardinality is 1 (“empty element”), or may be omitted if its minimum cardinality is zero (“missing element”).

In messages, the element must be communicated if its minimum cardinality is one. In the case where the element is not mandatory, it may be communicated with a null value. Note that any element declared to be "Mandatory" must also be "Required" and have a minimum cardinality of one. If the minimum cardinality is zero, and the element is "Required", conforming applications need not send the element if data does not exist. In some governance groups this is referred to as “Required but may be empty”.

For required elements, conforming applications must demonstrate their ability to provide and communicate not null values. Receiving applications must demonstrate

their ability to receive and process (e.g. store, display to users) not null values for required elements.

It is indicated as "R" in the conformance column of the template item table.

A sender must support elements with "required" conformance, and a receiver must understand these elements.

Optional (O)

The item is truly optional, i.e. a valid value may be provided or if missing may be omitted.

It is indicated as "O" in the template item table, a may be considered as a shorthand for an unspecified conformance with a minimum cardinality of zero.

Not permitted (NP)

The item is not allowed, i.e. the number of allowable occurrences is 0. It is indicated as "NP" in the template item table.

Conditional (C)

This item has an associated condition predicate and may depend on the co-occurrence of other elements or properties of the instance or situations. The condition must be testable in the context of an instance (see also section 2.10.4).

The conditional conformance criteria are mentioned in a corresponding table like this:

Table 3: Example of a Condition Predicate Table

Card	Conf	Predicate
1..1	**M**	If number of gravidities greater than 0
0..1	**O**	otherwise

In this example, the following situation is covered: if the predicate is satisfied then

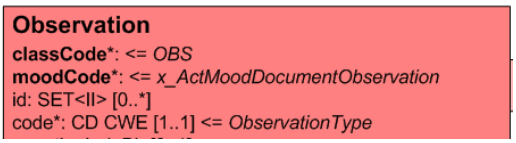
- a conformant sending application must always send the element
- a conformant receiving application must process or ignore data in the element; it may raise an error if the element is not present.

If the predicate is not satisfied then

- a conformant sending application should not (if conformance is optional) or must not (if conformance is NP = not permitted) send the element.
- a conformant receiving application must not raise an error if the condition predicate is false and the element is not present, though it may raise an error if the element is present.

A conditional attribute is indicated as "C" in the template item table, followed immediately by the condition predicate table.

Please note all definitions above may not violate defined cardinalities and conformance of a possible underlying model. As an example the “observation” class in the CDA bod allows for exactly one 1..1 code element and this may not be “overridden” by a template definition based on that model to e.g. 1..* code.



The constraints must be a proper specialization of the underlying model. A summary of specialization rules can be found in the HL7 Core Principles.

Fixed (F)

This is used for attributes only and indicates that an attribute has a fixed value. Its fixed value shall appear in an XML instance.

It is indicated as “F” in the template item table, a shorthand for a mandatory element with required conformance with a fixed value. The cardinality should be 1..1.

2.10.1.4. Conformance Verbs

The keywords SHALL, SHOULD, MAY, NEED NOT, SHOULD NOT, and SHALL NOT in this document and in template specifications are to be interpreted as described in the HL7 Version 3 Publishing Facilitator's Guide (http://www.hl7.org/v3ballot/html/help/pfg/pfg.htm):

- SHALL: an absolute requirement
- SHALL NOT: an absolute prohibition against inclusion
- SHOULD/SHOULD NOT: best practice or recommendation. There may be valid reasons to ignore an item, but the full implications must be understood and carefully weighed before choosing a different course
- MAY/NEED NOT: truly optional; can be included or omitted as the author decides with no implications

The keyword "SHALL" allows the use of nullFlavor unless the requirement is on an attribute for which the use of nullFlavor is explicitly precluded.

The objective of the Templates Exchange Format is to allow machine processable template representation. The representation with conformance verbs and natural language like used in CDA implementation guides is not the subject of this specification. It has been observed that in implementation guides there is a variation on natural language representation of conformance statements. Please refer to Appendix C to see the recommendations and best practice for template documentation and Appendix D on how existing constraints map to a template representation described in this document.

There is a relationship between conformance verbs and the cardinality/conformance indicators used in this guide. A summarization with these relationships can be found in the next table.

Table 4: Relationship between Conformance verbs and Cardinality/Conformance porperties used in this guide

Shown as	Manda-tory?	Confor-mance	Possible Cardinality	nullFlavor allowed?	Description
M	Yes	R	1..1 1..*	No	Element SHALL be present with a proper value
R	No	R	1..1 1..*	Yes	Element SHALL be present in instance and SHOULD be populated with a proper value; if no proper value is present, a nullFlavor attribute SHALL be present
R	No	R	0..1 0..*	No	Element SHOULD be present in instance and SHALL be populated with a proper value; if no proper value is present, the element SHALL NOT be present; this is some-times referred to as "RE" (HL7 Version 2) or "R2" (IHE)
O	No	O	0..1 0..*	Yes	Element MAY be present (is truly optional)
C		C			Element is conditional and may depend on the co-occurrence of other elements or properties of the instance or situations. The conditional conformance criteria are mentioned in a corresponding table.
NP	No	NP	0..0	No	Element SHALL NOT be present

2.10.1.5. Fixed Values

This allows expressing the intended fixed (prescribed) value for conforming instances.

The most frequent use of fixed values is with a structural attribute. From a model perspective, they are fixed and default to a single value (code). The cardinality of such

structural attributes is handled to be either optional (as is mostly the case in CDA) or required (as in V3 messages).

Table 5: Example of fixed values for structural attributes

Examples
observation classCode="OBS" moodCode="EVN"

Fixed values for attributes are denoted as “F” in the template item table.

2.10.1.6. Quantity Ranges

For quantities like real and integer it may be useful to specify allowed ranges in a template. This is typically done by defining a lower (minInclude) and a higher (maxInclude) boundary.

2.10.1.7. Units

For physical quantities (HL7 data type PQ) it is useful to specify allowed units. For HL7 V3 PQ items this has to be a case-sensitive UCUM expression.

Please note that quantity ranges are related to their units and therefore need to be grouped appropriately if there is more than one allowed unit.

As an alternative one may treat a unit as a simple code and bind a vocabulary (value set) to it.

2.10.1.8. Fractional Digits

In some cases, it makes sense to constrain the number of fraction digits for physical quantities / real numbers. This may include a required number of fraction digits or an indication of the maximum number of fraction digits.

2.10.1.9. Vocabulary Binding

For coded elements, a template design may restrict the use of terms from a specific code system. These vocabularies are defined in various supporting specifications and may be maintained by other bodies, as is the case for the LOINC® and SNOMED CT® vocabularies.

There are three types of vocabulary bindings.

- A coded element can be bound to a **specific code** from a specific code system and the population of the element can be considered as a “constant”; an element can also be bound to a set of specific codes.
- A coded element can be bound to a specific **value set** by value set id or name.
- A coded element can be bound to a certain **concept domain** by domain id or name.

Note that value set identifiers, e.g. value set with OID 2.16.840.1.113883.1.11.78 / name *HL7ObservationInterpretation* do not appear in messages/documents. They tie the appropriate value set to the coded item, for example to perform proper validation.

Concept domain indications are normally not testable in an instance as they indicate the binding to a possible abstract conceptual domain. In a template, they are typically

replaced by value set references by specializing the original template for certain use cases.

Value set bindings adhere to HL7 Vocabulary Working Group best practices, see section 2.10.5 about DYNAMIC vs. STATIC artifact binding.

2.10.2. Inclusion and Containment

One of the design principles of templates is the re-usability: a template, once defined, may be used again in any context wherever appropriate. From practice, two kinds of "re-use" mechanism are known.

An **inclusion** within a template design makes use of another template by "virtually" copying the included template definitions, also known as transclusion. In essence this means that template definitions are included by reference and shown as-is on demand, i.e. at time of displaying the template or using it for the creation of validation scripts. Inclusion is automatic and transparent to the user.

Example: a CDA document level template includes the definitions of a CDA *recordTarget*, *author*, *documentationOf* or *setId+versionNumber*.

A **containment** within a template design is a reference to another template without actually showing the contained definitions. A typical situation is a CDA section that may contain entries. The reference only is part of the section definition. The definition of the entry itself remains part of the entry level template.

The containment relationship allows composing a specific structure (context) and sub-structures (child elements) in an XML instance.

Example: a CDA section level template has an *entry* element defined where the content of that element is defined in an entry level template elsewhere. This is expressed by defining the section entry element *containing* that entry level template.

As a general rule of thumb, any represented model class (e.g. in CDA: recordTarget, section, entry, service event etc.) or even re-usable fragments (e.g. set of element complex effectiveTime dose structures to be used in more than one template or the CDA setId+versionNumber,) should be represented as a template in order to be able to refer to it through containment or inclusion somewhere else in a template design.

Table 6: Pattern of inclusion and containment in a typical CDA document template, using the Template ITS described in chapter 7.

Template definition pattern principle	
ClinicalDocument **...** **id** **code** **...** ***include*** **SetId+VersionNumber Template Fragment** ***include*** **Record Target Template Definition** ***include*** **Author Template Definition**	Document Level Template

... **component** **structuredBody** **component element *contains* Section Template**	
section **...** **code** **title** **text** **entry element *contains* Entry Template**	Section Level Template
observation **...** **code** **value** **entryRelationship element *contains* Entry Template**	Entry Level Template
substanceAdministration **...** ***include* effectiveTime Dosage Template Fragment** **...** ***include* doseQuantity Template Fragment**	Entry Level Template

2.10.3. Choice

In some cases a designer wants to offer a choice of template elements and restrict the choice to be one out of n (cardinality constraint). A typical example is the CDA header author definition where in the underlying model a choice is defined between an *assignedPerson* and an *assignedAuthoringDevice* playing the role of the author and where 0..1 playing entities may be chosen.

2.10.4. Co-occurrence

Co-occurrence means the presence of some data depending on the presence or value of some other data, also referred to as conditional data.

A condition has a predicate that falls “true” or “false” and an associated cardinality and conformance statement. A typical example is the co-occurrence definition of an observation “pregnancy”: if the “gender of the patient is female” (predicate) then the pregnancy observation has the cardinality / conformance 1..1 R, otherwise (default predicate) it is not present (NP). The following table illustrates this example.

Table 7: Example of the Condition Predicate Table for Observation "pregnancy"

Card	Conf	Predicate
1..1	**R**	If patient is female
	NP	otherwise

Co-occurrences are often expressed in natural language, in other cases a formal expression is possible that can be tested against the instance by means of validation mechanisms, such a schematron.

Typically, there are co-occurrences within a template design when the design depends on a condition within an instance. As an example, a documentation about a female patient shall contain information whether she is pregnant or not.

A co-occurrence may relate to external data, i.e. where the presence of data in an instance is influenced by external factors (outside the very same instance). As an example, a laboratory result that is expected to be present based on the history of the patient specified in an electronic health record.

In this specification, only co-occurrence constraints within an instance are handled. It is known that there are projects with business rules like "a specific laboratory value must be present at least once in three months" that need to be addressed as well but that is out-of-scope for this specification.

2.10.5. STATIC vs DYNAMIC artifact binding

For the binding of artifacts (i.e. templates and value sets) within a template definition, especially

- Vocabularies (value sets)
- Templates as Inclusions or Containments

the same mechanics are applied: an artifact is bound to an element either as

- STATIC, meaning that they are bound to a specified version (date) of the artifact, e.g. a template design contains template X as of version 2017-01-02
- or DYNAMIC, meaning that they are bound to the most current version of the artifact, e.g. a template design contains template X in the most recent version.

Value set bindings adhere to HL7 Vocabulary Working Group best practices, in principle they follow the same rules as the template bindings.

A STATIC binding is a fixed binding at design time whereas a DYNAMIC binding implies a need for a look-up of the (most recent) artifact at runtime.

It can also be found in practice to "freeze" any binding defined as "dynamic" to the most recent artifact at the time of the official publication, making "dynamic" bindings actually "static" for the most recent version. This makes the publication stable with regards to the binding of artifacts.

2.10.6. Conformance Constraint Statement Identification

Constraint statements may be uniquely identified by an identifier at or near the end of the constraint (e.g., CONF:7345). These Conformance Constraint Statement Identifiers are called item/conformance labels and are persistent but not sequential. An

implementation guide should have a policy on how to create, maintain and possibly pertain those item/conformance labels.

This Template ITS specification allows to assign an item/conformance label on the template metadata level that is used throughout the template. This means that every element in the template design body carries the template metadata item/conformance label and thus associates/labels the element belonging to the template. In addition, each element in the template design body may carry its own item/conformance label, identifying the particular element and overriding a possible "overall" item/conformance label given in the template metadata.

It is best practice that for validation messages the emitted error or warning is preceded or prefixed by the associated item/conformance label, either from the template metadata or from the particular element.

If you clone a template and create an adaptation of it, it is best practice to delete the original item / conformance labels, since conformance labels are not part of the normative specification.

2.11. Open versus Closed Templates

Open templates permit anything to be done in the underlying standard that is not explicitly prohibited. This allows templates to be built up over time that extend and go beyond the original use cases for which they were originally designed.

Closed templates only permit what has been defined in the template, and do not permit anything beyond that. There are good reasons to use closed templates, sometimes having to do with local policy. For example, in communicating information from a healthcare provider to an insurance company, some information may need to be omitted to ensure patient privacy laws are followed.

Most templates developed for CDA are of the open sort.

Another typical situation is that templates in a repository for re-use are defined as open as when they are used within a document definition (document level template) a governance group may decide to use all templates as closed, i.e. no other content then specified is allowed. The same may temporarily apply during conformance testing, for example a connect-a-thon where it may be required to detect undefined content.

Table 8: Example of a template definition specified as "open" or "closed" and its effect on an example instance

Template definition
observation **code** **@code = "XYZ"** **@codeSystem = "1.2.3.4"** **value** **@value REAL** **@unit = "kg"**

Instance	as open	as closed
<observation> <code code="XYZ" codeSystem="1.2.3.4"/> <value value="82" unit="kg"/> </observation>	Valid	Valid
<observation> <code code="XYZ" codeSystem="1.2.3.4"/> <effectiveTime value="20140201"/> <value value="82" unit="kg"/> </observation>	Valid	Invalid

It should be mentioned that it is possible to also have only parts of a template being defined as closed. An example is a template element e.g. *effectiveTime* with the datatype interval of time stamp where you allow only specific and explicitly defined elements like low and high, while all other possible sub elements are forbidden. This can be achieved by defining an *effectiveTime* element as closed and with sub-elements low and high to allow only these two.

2.12. Types/classes of Templates

According to their "position" or function within a message or a document or the root class they are constraining in the underlying model the following types/classes of templates are distinguished.

2.12.1. CDA Document Level

These templates constrain fields in the Clinical Document Architecture (CDA) header, and define containment relationships to CDA sections.

For example, a History-and-Physical document-level template might require that the patient's name be present, and that the document contain a Physical Exam section.

2.12.2. CDA Header Level

These templates constrain fields for parts of the CDA header, like the patient, the author, the service event or variants of participants

2.12.3. CDA Section Level

These templates constrain fields in the CDA section, and define containment relationships to CDA entries.

For example, a Physical-exam section-level template might require that the section/code be fixed to a particular LOINC code, and that the section contain a Systolic Blood Pressure observation.

2.12.4. CDA Entry Level

These templates constrain the CDA clinical statement model in accordance with real world observations and acts.

For example, a Systolic-blood-pressure entry-level template defines how the CDA Observation class is constrained (how to populate observation/code, how to populate observation/value, etc.) to represent the notion of a systolic blood pressure.

2.12.5. Message Level (V3 / v2.xml)

These templates constrain the HL7 V3 message model and may define containment relationships to wrapper and payload components including Common Message Element Types (CMETs). This is the counterpart to CDA document level templates. The type is also used to classify v2.xml messages/profiles, the XML representation of HL7 v2 messages.

2.12.6. Control Act Level

These templates constrain the HL7 V3 message model and may define containment relationships to control act wrapper components including Common Message Element Types (CMETs).

2.12.7. Payload Level

These templates constrain the HL7 V3 message model and may define containment relationships to payload components including Common Message Element Types (CMETs).

2.12.8. Clinical Statement Level

These templates constrain the HL7 V3 clinical statement model in accordance with real world observations and acts, used for example in Patient Care Messages. This is similar to CDA entry level templates.

Please note that Clinical Statement Level templates used in some V3 messaging artifacts may be equivalent to CDA entry level templates but not necessarily identical.

2.12.9. v2.xml Segments Level

These templates constrain segment structures of a v2.xml message/profile.

2.12.10. Data Type Level (= Data Type Flavors)

A data type may also be constrained by means of a template. In effect, the mechanism to handle data type constraints since Data Types Release 2 (ISO 21090) is to create and use a data type flavor, for example instead of “INT” for an integer use “INT.POS” if a positive integer is required.

2.12.11. FHIR Resource Profiles

The corresponding artifact to a template in the FHIR world is a profile. Expressing FHIR profiles as “templates” as defined here has been explored by both HL7 core groups but is not within the scope of this document yet.

2.12.12. vMR-CDS Profiles

The virtual medical record for clinical decision support (vMR-CDS) effort is also seeking to align its template definitions with this specification.

2.13. Template Design Considerations

An important consideration about a template design is how deep the template structure should go. A model is a collection of classes and the associations between them. When a model is expressed as a set of conformances in a template, the structure could be infinitely deep. To date, most templates are rather shallow, but some may include three or four levels of nested conformances. If one of the levels includes a previously defined template, then the nesting continues deeper based on the levels defined for that template.

This design feature creates a modularly recursive or "fractal-like" characteristic to a template. Being able to break a large model into a variety of smaller reusable sets of constraints has proven to be useful. However, it does add to the complexity of the template designs.

Please note that recursion cannot be prevented by design. An example: an organization has organizational parts that are again organizations.

Good tools detect that recursion and generate correct schematrons out of it. A recommendation is that inclusions may not be recursive (as they are virtual copies) but containments may (as they reference a template only). See section 2.10.2 for inclusion vs. containment.

When does a new design represent a new template as opposed to a new version of an existing template? If the intent/purpose of the template changes substantially or the design of template is substantively different, a new template needs to be created. Typical situations where a new template is required include:

- Different purpose, where the difference can be either a narrower or broader purpose from an existing template's definition, or a substantively different purpose
- Value Set Bindings,
- Cardinality / conformance of template elements,
- Containment of other templates
- Backward compatibility invalidation (see also section 3.4.8 about "backward compatibility").

2.14. Template Item Table Example

The following table is an example for a template item table. It is the start of the "Age Observation" Template from C-CDA [ccdar1].

Table 9: Example of a Template Item Table

Item	DT	Card	Conf	Description/Value
observation				
@classCode		**1..1**	**F**	OBS
@moodCode		**1..1**	**F**	EVN
templateId	II	**1..1**	**M**	Template Id of this template
@root		**1..1**	**F**	2.16.840.1.113883.10.20.22.4.31

Item	DT	Card	Conf	Description/Value
code	CE	**1..1**	**F**	
@code		**1..1**	**F**	445518008
@codeSystem		**1..1**	**F**	2.16.840.1.113883.6.96 (Snomed-CT)
value	PQ	**1..1**	**R**	
@value		**1..1**	**R**	The actual age
@unit		**1..1**	**R**	Code from value set : AgePQ_UCUM

Please note that from the table it is immediately obvious that all structures have fixed values, except the *value* itself. In other words: a corresponding instance can be created with most of the "HL7" structures already determined and the focus is on the value only, which carries the actual "age". Setting the focus on elements and attributes that actually carries a variable value can make implementation easier.

3. Template Relationships

3.1. What are template relationships?

The design of one template may depend on, build upon, or further constrain the design of another. Also, design interdependencies can define one template by referencing conformance expressed in another template or by including a component which has been defined separately in a different template. One template may have had several designs over time as the exchange requirements for the information evolved. A subsequent design of a template can include conformances that affect the ways in which a new design is similar or different from its prior version.

3.2. Why are template relationships useful?

Template relationships help us understand how information specifications are similar to or different from each other and enable us to logically deduce how data conforming to a template may or may not be interoperable. When data is conformant to certain templates, understanding the relationships between templates allows us to reason about the data that is exchanged.

3.3. How do template relationships work?

Template relationships enable processing logic to be applied to the instance data that is represented. If Template B is a further constraint on Template A, then by deductive reasoning, an instance of data that conforms to Template B can be processed according to rules established for processing data that conforms to Template A or Template B. An instance of data that conforms to Template A only can be processed according to rules established for processing data that conforms to Template A.

3.4. Types of relationships between templates and other artefacts

Based on the design techniques used to define templates, several types of relationships exist between different templates and between design versions of a single template.

3.4.1. Replacement (replaces)

A replacement is used when a new version of a template replaces older ones. With a replacement, the status of the older one typically gets retired (or inactive). This is inherent to versioning and could also indicate a new design replacing an old design (with a new id).

3.4.2. Version

A “version” relationship may be automatically derived (in a proper registry) or explicitly set to express the relationship to other (older or newer) versions of a template design. Versions typically have the same id but different effective dates.

Distinct versions of a template can be “active” or “draft” or “review” at the same time, e.g. version 2014-02-03 of Template X is “active” while another version 2014-05-07 of that template is still in “draft”.

3.4.3. Specializations / Generalizations (specializes, generalizes)

A specialized template is a narrower, more explicit, more constrained template based on a "parent" template. A generalized template is a broader, less restricted template.

Examples: A "Body Weight Observation" is a specialization of a "Simple Observation". A "Substance Administration Act" generalizes a "Prescription Act".

3.4.4. Design Copy (copies)

This is used to express that a copy of a template design from another governance group has been made without change, except the id.

A design copy implies the creation of a new template. The new template gets a new id, and all other properties including the effective date are copied from the previous template.

Foreign governance groups typically do this for later adaptation.

3.4.5. Adaptation (based on)

Copying a template design from another governance group (i.e. design copy/clone) and change it, i.e. change both id and design is called adaptation. The adapted template is "based on" the original template which means it can be an extension or a specialization (restriction) of the original template design.

This implies always the creation of a new: the adapted template gets a new id and a new effective date.

3.4.6. Equivalency (equals)

"Equivalency" indicates that two templates have the same purpose and the same design; however, their governance and/or metadata and/or details of their design may be different. They are meant to be semantically equivalent because a transformation exists without addition or loss of meaning.

Examples: use of different vocabulary (LOINC vs SNOMED), slightly different XML representation of CDA entries vs. Clinical Statements in Patient Care Messages.

3.4.7. Uses / Used by

A template may use other templates by inclusion or containment or is used by other templates. Although this is automatically detectable (and detected) in a proper registry there is the need to mention this relationship quality.

3.4.8. Backwards compatibility

Indicates that a template is considered to be backward compatible to a previous version of the template. A template version is backwards compatible if all instances of data which are compliant under the prior version also are compliant under the new version.

Determining true backward compatibility can be complex. A new version of a template defined as "open" may reject instances constructed using the older version as "open", which means other elements are allowed in an instance beyond those explicitly defined. In other words, older instances may contain legal additional input that fails to validate against a newer version of the template. On the other hand loosening constraints would generally be backward compatible.

True backward compatibility can be determined more reliably with templates defined as "closed" only:

3.4.9. Forward compatibility

This relationship indicates that a template version is compatible with a newer version of the template. Forward compatibility means that an instances of data created under the newer template can be used with applications designed to work with data created under the prior template.. Instances conforming to the new template should also conform to all the old templates.

3.4.10. Containment

This relationship indicates that a template has a reference to a contained template. In a proper registry/repository this relationship may be determined automatically (implicit relationship), see also section 2.10.2.

Examples: A CDA section level template has an entry that contains an *Age Observation* template. A CDA document level is composed out of several distinct section level templates in the CDA body, the CDA body contains section level templates.

3.4.11. Inclusion

This relationship indicates that a template makes use of virtually copying another template design into the template. This mechanism is elsewhere described as *transclusion*. In a proper registry/repository this relationship may be determined automatically (implicit relationship), see also section 2.10.2.

Examples: A CDA document level template includes a *Record Target* definition that is used in multiple templates. A *Substance Administration* act includes all definitions of various timing aspects expressed in a set of *effectiveTime* templates.

3.4.12. Derived from

This is a relationship that indicates a model (R-MIM or another "model" etc.) where the template is derived from or based on.

3.5. Relationships to other artifacts

3.5.1. Relationship to Underlying model(s)

This relationship is used to express the link to an underlying model. Examples are conceptual models, Detailed Clinical Models, HL7 R-MIMs, HL7 CMETs. In other words, the template is based on / derived from a (more generic) model.

3.5.2. Subscribe with "Adopter"

An "adopter" relationship expresses a kind of relationship to other individuals or governance groups in the sense of a subscription (a "template follower") on any changes or improvements of a template or to proactively offer participation in testing and reviewing new versions. Governance of the template remains untouched.

It is a template registry matter to record those interests and to signal the "adopters" on changes or the start of review phases.

4. Using Templates to Create, Validate and Consume Instances

4.1. Templates as Definition of Clinical Concepts

HL7 Models serve as a structured formalism through which human beings can unambiguously exchange agreed knowledge models that describe concepts they share. In this wider context, templates are often used to define knowledge models at a finer level of granularity than the HL7 interoperability definitions.

Since all HL7 V3 templates rely on Static Models, all templates are a formal definition of a particular concept. Templates may be used to further refine any models in any domain of interest to HL7, including messaging infrastructure, healthcare administration and especially for clinical concepts.

Templates are used for three areas in information exchange:

- Construction of Instances
- Validation of Instances
- Processing of Instances

These activities are done by systems/individuals in the following roles:

- Instance Content Creator
- Instance Content Consumer

Applications may need to use other resources to define how the information model described in the template is properly presented or applied in other contexts, such as input screens. These things are out of scope of this template specification.

However, some vendors already use templates expressed in the templates exchange format (see chapter 7) to build reference user input interfaces or to generate code for proper instance generation etc.

4.2. Construction of Instances

Models and templates are used to guide and direct information input for a message or document. This may take the form of a specification for a user input interface, or where a set of templates is used to guide an application in constructing a proper instance. Templates can be used to provide knowledge models that are applied on a context sensitive basis to a consistently applied more general model, and provide a coherent framework for customizing application behavior.

A good example is the Continuity of Care Document definition by HL7 as part of the Consolidated CDA Templates for Clinical Notes, where specific templates are used to specify header parts and a collection of sections with their corresponding entries.

It is important to recognize that templates do not carry semantics. An instance that is built using templates shall carry all semantics on its own and must be understandable even without any reference to or knowledge of the underlying templates.

Applications using templates for instance creation may choose to reference the template in the instance to support validation and processing and it is current practice to include all template references in an instance.

4.3. Instance Content Creators

Instance Content Creators use templates to assert the patterns and semantics present in the data instances they create.

If a system creates a document instance that conforms to all the specification of Template A, then the Instance Content Creator can assert Template A in the document. This assertion can be used by Instance Content Consumers to process the data contained in the document. The assertion can also be used by Content Validators to confirm that the document does in fact meet all the constraint requirements specified for the template.

4.4. Validation of Instances

Templates are used to support validation of instances. When a template is applied, either by definition or by reference, the instance can be validated against the constraints expressed in the template in addition to the base static model.

Template definitions may overlap as they apply to an instance. More than one template may be applied to a given class, or a template defined elsewhere in the instance may still apply when a new template is applied. When more than one template is asserted multiple constrained models apply. The set of valid instances is defined by the intersection of the set of conformances described by each template. Note, if the models contain incompatible constraints, there may be no instance of data which can satisfy all the applied templates. The set of valid instances is the empty set.

4.5. Processing of Instances

Template ids in validated instances may also be used to process the instance properly. The presence of a specific template reference makes sure that the document conforms to the specification of the asserted template and it can be used to trigger the extraction of the information accordingly.

4.6. Instance Content Consumers

If a document is valid, then a Content Consumer can be certain that the data in the document conforms to the specification of the asserted templates (see also section on Content Validators below).

If the Content Consumer application has the ability to process data that conforms to a certain template, and the document contains that template, then the Content Consumer can receive that information.

If the document contains information that conforms to a template, which the Content Consumer does not have the capability to process, then data conforming to that template may not get processed.

If there is no expressed requirement for a Content Consumer to process certain templates, then it is permissible for the Content Consumer to process only that data which conforms to the templates it claims to support.

Note that Content Consumers may want to validate an instance before consuming it.

4.7. Content Validator Methods

Regarding the validation of instances in a certain context and against the underlying models and templates it must be mentioned that some of the constraints are testable in the scope of an instance, while others require examination of the inputs and outputs of a system by humans to determine conformance.

For the testable constraints of an instance validation methods are available to do the test whether an instance is conformant to definitions. In essence a few methodologies are mainly used in practice:

- W3C schema validation, mainly derived from the underlying static model
- ISO schematron validation, derived from the template definitions,
- Use of object graphs created by model-driven technology

For the Template exchange format in chapter 7, a transformation exists to create schematrons directly out of the template definitions (see section 7.5). The underlying schematron engine creates ISO schematron rules and also compiles all value set references that are needed for a proper validation.

Please note that schematron is not the only validation mechanism, but by far the most broadly used one.

According to the definition of the nature of a constraint, “errors”, “warnings”, or just “information” can be the output of a validation process against an instance. Any type of constraint described above may determine the nature of the validation message.

A governance group determines the behavior of the validation process if an instance fragment is not conformant. They may decide to issue errors, or just warnings or informational text. This may not be in contradicting to possible existing requirements that come from the underlying model.

Regarding the use of vocabulary, a test of an instance fragment whether it uses the correct vocabulary depends on the type of binding. If a value set binding is specified and the value set is completely available for the validation process, it is no problem to check codes in instances. In all other cases it might be difficult to check codes against value sets or even abstract conceptual domains that are not physically available.

4.8. The use of multiple Template Ids

More than one template can be used at the same time in the same document, so long as all the rules defined by each template hold true in the document where the templates are asserted. Instance Content Creators can use the layering of templates when they create a document in the same way that Template Designers can layer templates to define a specific template.

Template layering supports incremental interoperability. Layers can be added when data needs to be further constrained for more specific purposes. The templates can be used to meet very specific exchange requirements without additional negotiation. By establishing standard templates to meet the data interoperability needs, Instance Content Creators and Instance Content Consumers simply utilize the templates to generate and process the documents as defined by those standard templates. The inherent nesting properties of template designs make it possible for the data to be created and understood at varying levels of specificity.

The order of multiple template identifiers for one instance fragment supplied in an instance is not significant. However, to understand the combined effect, it is useful to envision the data as constrained initially by the most constraining template. Assuming that the set of templates have designs that "nest together", it then follows that the data also conforms to templates which are increasingly more relaxed, or generalized.

For example, if a system supports template A and template B which is a further constraint on template A, then it can create a document which includes data that conforms to both Template B and Template A. A system which understands only template A could still process the data, but not at the level of specificity that could have been achieved if it understood template B.

When creating or processing a list of templates, it may be unclear which template is the most constrained or it might be even none of them being the "most" constraining template. Currently, there is no computational algorithm for determining this, but it can be deduced through analysis of the template definitions.

It is not required that a document include all templates to which it conforms. The Instance Content Creator asserts the intended templates in a document at the time it is created. It may be possible for a receiver to determine that there are other models to which the instance of data is also conformant, even though they are not identified in the instance.

4.9. Implementation principles and use of template ids in instances

A template is referenced in an HL7 Version 3 / CDA XML instance by the XML element templateId wherever appropriate along with at least the template id in the @root attribute, e.g.

Example

```
<templateId root="1.2.3.4.5"/>
```

There is the possibility to reference to a specific version of a template (for example review or trial use cases, connect-a-thons using previews of templates). In this case the *@extension* attribute of the *templateId* element extends the id to a specific version, e.g.

Example

```
<templateId root="1.2.3.4.5" extension="2013-10-12"/>
```

or

Example

```
<templateId root="1.2.3.4.5" extension="v1.0"/>
```

If *@extension* is omitted then the binding to the template is as defined by the governance group. This accommodates the migration phase of various versioning practices to this new specification.

Attribute *@extension* can be any string and corresponds either to the version date (effective date) or to the value label defined in the template metadata. This specification does not allow any other association or use of the *@extension* attribute other that this. From a datatype perspective *@extension* is typically used for non-numeric strings or

those who are against the OID-rules (e.g. leading zeroes). It is recommended to always use alphanumeric string, such as “2014-07-12”, “v1.01” or “V1” for version label definitions and *templateId/@extension*.

In V3 specifications using the HL7 Datatypes Release 2 (ISO 21090), one may add other properties of the datatype II, such as *identifierName*.

Example

```
<templateId root="1.2.3.4.5" identifierName="MyTemplate"/>
```

For other XML-based definitions, the template context may use the “path” context, see section 2.9.12.

5. Use Cases Demonstrating the Creation, Use, Maintenance and Governance of Templates

This chapter is non-normative and shows typical use-cases of the Creation, Use, Maintenance and Governance of Templates.

5.1. Creation and life cycle of template "Estimated Delivery Date"

This example is about the life cycle of template definitions to represent data about an Estimated Delivery Date (EDD), defined as an open template.

5.1.1. Step 1: initial draft

Designer perspective: The governance group / template designer decides to create a brand new (nascent) template for the purpose of representing data regarding the "Estimated Delivery Date (EDD)".

The template creation was started on 2013-05-02 (effective creation date) and the initial template received the OID 1.2.3.7 with status "new". After a while they decided to publish the first draft of the template, still finalizing some details on the design.

Table 10: Example template

Template "Estimated Delivery Date"

Metadata
• Identifier: 1.2.3.7
• Effective Date: 2013-05-02
• Status code: draft
• Name: EstimatedDeliveryDate
• Version label: 1.0
Design
• Observation
  • @classCode = OBS
  • @moodCode = EVN
  • code = LOINC 11778-8
  • value to carry a valid "estimated delivery date" (demotion to xsi:type = TS)

Content creator perspective: An application may start with the implementation of the template and test/review it but must be aware that a template with status "draft" may change, even substantially.

5.1.2. Step 2: comment and review phase

Designer perspective: The designer decides to publish the draft and start the test phase to get feedback from the governance group, external reviewers, or content creators or consumers.

While the design remains the same, the metadata status code may be changed to reflect the new status of the template: the template version is now pending.

Table 11: Example template

Template "Estimated Delivery Date"
Metadata • Identifier: 1.2.3.7 • Effective Date: 2013-05-02 • Status code: **pending** • Name: EstimatedDeliveryDate • Version label: 1.0

Reviewer group / content creator or consumer perspective: The template should be reviewed / tested.

5.1.3. Step 3: revision with a minor change

During the test/review phase it became obvious that an optional date of determination of the EDD is needed.

The original template version was designed as "open", so an optional additional element can be added without "breaking" the former template design, i.e. instances based on the new template version are still valid against the former version.

Designer perspective: At 2013-05-20, the designer adds an *effectiveTime* as TS to the design, the rest is unchanged. Because the template was in state "review" he decides to create a new version of the template.

The use of the former template version may be declared as "discouraged" with the changing the metadata status code to "inactive".

Table 12: Example template

Template "Estimated Delivery Date" (old version)
Metadata • Identifier: 1.2.3.7 • Effective Date: 2013-05-02 • Status code: **inactive** • Name: EstimatedDeliveryDate • Version label: 1.0

The new version has the following metadata and design

Table 13: Example template

Template "Estimated Delivery Date" (new version)

Metadata
• Identifier: 1.2.3.7
• Effective Date: **2013-05-20**
• Status code: **draft**
• Name: EstimatedDeliveryDate
• Version label: **1.1**
• **Relationships: replaces template 1.2.3.7 as of 2013-05-02**
 	
Design
• Observation
  • @classCode = OBS
  • @moodCode = EVN
  • code = LOINC 11778-8
  • **effectiveTime of type TS as "date of determination"**
  • value to carry a valid "estimated delivery date" (demoted to xsi:type = TS)

Content creator / consumer perspective: Due to the notification that the original template got a new version that does not necessarily imply a required change of their application code, the may or may not adopt the new version of the template.

5.1.4. Step 4: revision with substantial changes

The test and review phase came up with the change proposal to

- make a change to the allowed *Observation.code* so that it not only allows LOINC 11778-8 but also Snomed-CT code 161714006 (change of vocabulary binding)
- allow to convey the method of evaluation of the EDD as a required item, with a proposed list: last ovulation, last conception, last menstrual period, quickening, ultrasound.

The governance group decides to follow the suggestions.

Designer perspective: At 2013-05-30, the designer makes the two suggested changes by creating a new (major) version of the template, leaving the rest of the design unchanged. He also compiles a new value set *EDDMethod* to reflect the concepts in the proposed list.

In addition, the governance group may decide to set the status of template version 1.1 to "retired". Note that the status of predecessor templates is not necessarily or automatically influenced by the existence of a replacing template. It is the responsibility of the governance group to decide what happens.

Table 14: Example template

Template "Estimated Delivery Date"

Metadata

- Identifier: 1.2.3.7
- Effective Date: **2013-05-30**
- Status code: **draft**
- Name: EstimatedDeliveryDate
- Version label: **2.0**
- Relationships: replaces template 1.2.3.7 as of 2013-05-02
- **Relationships: replaces template 1.2.3.7 as of 2013-05-20**

Design

- Observation
  - @classCode = OBS
  - @moodCode = EVN
  - code = LOINC 11778-8 or Snomed-CT 161714006
  - effectiveTime of type TS as "date of determination"
  - **methodCode** as "method of evaluation of the EDD", shall be drawn from value set EDDMethod.
  - value to carry a valid "estimated delivery date" (demoted to xsi:type = TS)

6. Management and Governance of Templates

6.1. Role of governance groups

What is the role of a governance group in the management and governance of templates?

A governance group is the custodian of a set of templates. A governance group can act in any of these three roles: Template Creator, Template Adopter, and Template Adapter. It establishes a Template Repository to hold the template designs it creates and uses. It may maintain a Template Registry to track the template designs it develops and uses, and tracks their status and relationships. It set the policies and procedures it will follow when using, adding, and addressing the life cycle of each template design under its control (see also 2.7.2 Governance Group).

6.2. Template Repository

What is a Template Repository?

A template repository houses the definition of template designs. As templates are added or modified, their designs remained documented in the Template Repository maintained by the governance group which is the custodian of the designs (see also 2.7.3 Template Repository).

6.3. Template creation

How does a new template get created?

To create a new template or a new design for an existing template, a template designer works with a subject matter expert, or a group of subject matter experts, to consider a specific use case and representational requirements for the information. If no existing template can be used to meet the requirements, then a new design is developed. The template designer documents the needed conformance statements that make up the template (see 2.7.1 Template Designer).

6.3.1. Use cases / business requirements

What role do use cases or business requirements play in the management of templates?

Use cases and business requirements are used to establish the purpose of a template. As the use cases expand, or the business requirements change, the purpose that a template serves may change, a new template design may need to be developed, or a new template may need to be developed.

6.3.2. Template content provider skills

Who is the template content provider to provide the functional requirements for the purpose and clinical information design?

Template designers have the technical expertise to analyze use cases and business requirements in order to construct the needed conformance statements to constrain and apply an information model to meet the information exchange requirements. However, it requires subject matter expertise to provide the functional requirements which inform the design. It often requires a great deal of input from multiple clinical subject

matter experts and multiple technical template designers to establish the templates needed for a complex use case.

6.3.3. Mapping clinical requirements to technical artifacts

Based on existing HL7 models and representations, how do the clinical requirements map to the technical artifacts of the template design?

Each clinical requirement may generate one or more technical requirements for a template's design. Information models and the conformance statements which constrain them can be very complex. Multiple dimensions must be considered including aspects such as but not limited to the structure and semantics associated with representing the information, the relationships across information, the provenance of the data, and the longitudinal nature of the information.

6.4. Template endorsement, publication, testing and maintenance

How do templates get endorsed, published, tested, maintained over time?

Each governance group establishes the rules and best practices for reviewing, approving, publishing, testing and maintaining the set of templates it governs. This process often follows well-established practices that align with broader governance policies for the organization. The governance of versioning is part of how an organization manages the information it controls as it changes over time.

6.5. Re-use of templates

How do existing templates get reused or adapted to create a new design?

When an information model is first developed, the number of templates in existence to constrain and use it is small. As the information model becomes more adopted and more uses are identified, the number of templates increases. This creates the need to reuse and adapt existing templates, as a natural maintenance requirement for interoperability. Prior designs must always be considered when creating new designs or else the information conforming to these established patterns will not benefit from the comparability implied by the use of the templates.

6.5.1. Referencing templates

How do you reference an existing template?

When the design of a new template is related to the design of another template, it is best practice to establish a relationship to that template, so that you can see where the new design is derived or copied from. By establishing relationships between templates and stating the nature of that relationship (an adaptation or further constraint of a prior template design etc.), only the incremental differences need to be understood, assuming an understanding of the prior template already exists.

When you want to use a template from another governance group it is best practice to reference the other template rather than copying the template "into" your governance group (and may be even assign a new OID although the templates are identical).

6.5.2. Obtaining template designs

How do you obtain template designs?

Template definitions are published and can be read to gain an understanding of the design, but this is not an efficient way to work with the definition when developing additional templates. To exchange a set of template definitions and the detailed conformance statements each template comprises, a standard has been devised. The implementable technology specification (ITS) for this exchange standard allows systems that maintain a template repository to export the template designs it manages and import template designs which have been created in other systems (see chapter 7).

6.5.2.1. Licensing issues

What are the template licensing issues?

Like other types of designs, templates are intellectual property. As intellectual property, copyrights and other forms of licensing and legal protections may be applied.

6.5.2.2. Use of template design

How can I make use of the design?

Prior template designs can be referenced in accordance with the rules established by the governance group that controls them. Referencing a prior template using its identifier rather than copying its conformances into a new design is the best practice.

6.5.3. Publish and Subscribe

How does Publish and Subscribe functionality support template management?

When template designs include interdependencies, it becomes necessary to stay informed of changes in the related templates. Template Registries may support functionality which permits a system to declare its interest in a template or set of templates so that it may be informed when changes occur. Once a system has subscribed, the Template Registry can publish a change, and the needed notification happens automatically.

6.5.4. Benefit for implementers

How can an implementer benefit from a layered template definition approach?

Appropriate tooling should combine the layered template definitions into different comprehensive views.

7. Standards for Exchanging Template Definitions

7.1. The Template Definition Exchange Standard

The **Template Definition Exchange Standard** is an implementable technology specification (ITS) for the exchange and storage format of template artifacts.

7.1.1. Objectives

The main objective of the Templates ITS is to provide machine-processable template definitions for various purposes:

- Publication and Documentation, Browsing
- Exchange, Exposition by Template Repositories
- Generation of Validation Methods, e.g. ISO Schematron
- Others, e.g. Code Generation

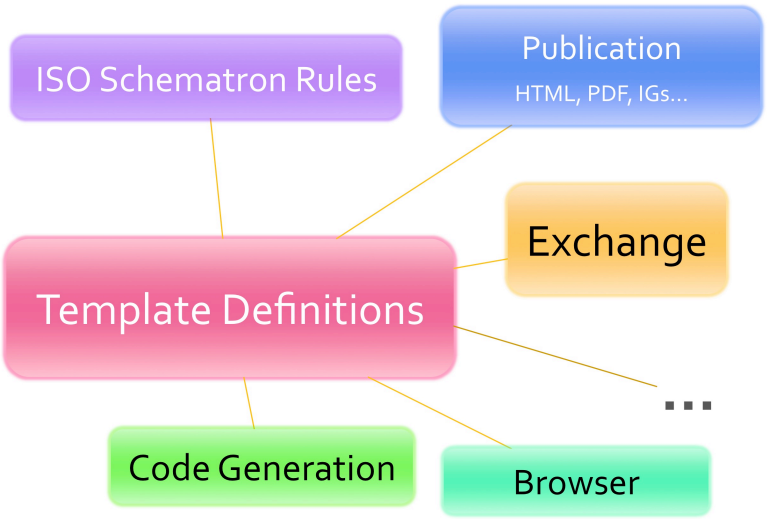


Figure 11: Template definitions and derivations for publication, validation, browsing, exchange and possible other purposes.

7.1.1.1. Template Publication and Documentation

Main objective is a proper documentation of the template definition. This means for example the availability of a template design browser that shows selected templates. The template definitions should also be available in other visual formats such as HTML or PDF or should be easily embeddable in implementation guides.

During the last years different governance groups published template definitions in various formats (see also Appendix C), using also different types of expressions for conformance requirements.

The definitions of a template are typically transformed into a tabular view (see example in section 2.14), but a governance group may decide to present them also in other formats or transform them into human language based conformance statements.

For the best practice examples in section 7.7 possible tabular view renditions are shown. In Appendix C other representations are shown.

7.1.1.2. Exchange Format

During the last years, a few template-authoring tools have been developed. While the underlying rationale and the details of how template design is acquired, presented and stored may be different, an exchange of template definitions between these tools is useful if not required.

This specification provides a standardized exchange and storage mechanism. In several proof-of-concept projects prior to the publication of this specification it could be shown, that in most cases conversion into the ITS here is possible with only little corrections by hand. It can be expected that – once this ITS is endorsed– tools may converge to read and write template specifications following this ITS without any manual interaction. This might include a transition to typical concepts described here, for example the choice mechanism or that a template must include all defined items that are allowed in order to be able to handle "closed" templates.

7.1.1.3. Source for Generation of Validation Methods

Template definitions serve three purposes regarding instances: creation, validation and processing of instances. Providing a standardized documentation format for template definitions allows to apply transformations on the templates to generate validation methods for instance testing.

During the last years schematron [isosch] has been established as the mainly used constraints language. It allows the definition of a set of rules which are applied to an XML instance. As a result, errors, warnings and informational notes may be presented coming from detected non-conformant fragments of the tested instance or an indication that the instance (fragment) is conformant (valid). A governance group may determine what kind of "violation" may lead to what kind of validation feedback (error, warning etc.). For example, SHALL, SHOULD and MAY conformance verbs may lead to errors, warnings and informational notes respectively. However, the kind of validation feedback may not be against conformance rules defined elsewhere (e.g. the underlying model), for example a missing element that is defined "mandatory" shall always throw an error.

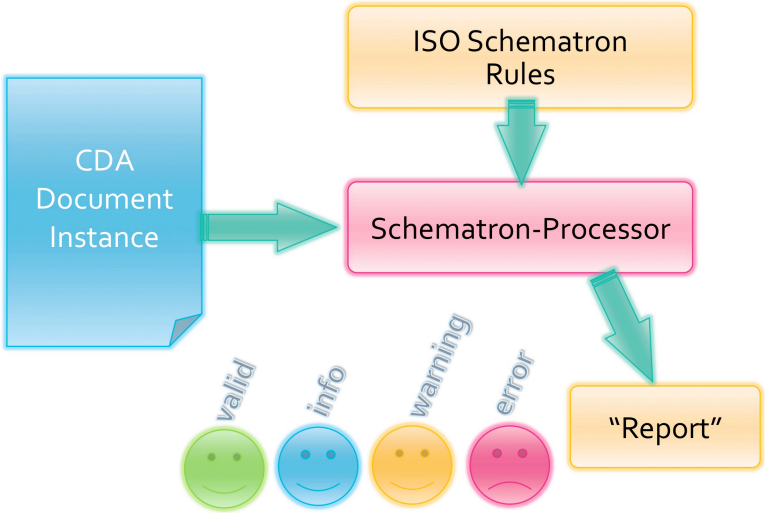


Figure 12: Schematic overwiew of a CDA instance validation with ISO schematron

For this ITS, a transformation suite (schematron engine) is available that converts the template definitions into schematrons. The results can be used to validate instance and test their conformance.

7.1.1.4. Easy Access to Templates by Repositories

Once a set of templates is ready to be published the aspect of a proper template registry and/or repository comes into play. This implies to not keep the template definitions locked up in a proprietary tool or published them in paper format only, but to allow other groups, e.g. template consumers (implementers), to access the template definitions electronically along with all other needed artifacts (e.g. value sets) for machine processing, code generation, validation and testing etc.

For that purpose, templates – once endorsed for use – ideally are exposed in a template repository/registry to allow access for the purposes mentioned above. In addition to that, a governance group may decide to make use of a template definition of another governance group by including endorsed templates in their specifications or to clone and refine or adapt them. This is easy by offering template definitions in a template repository/registry in a standardized format.

7.1.2. Out-of-scope

Other constraint languages than schematron, for example OCL or GELLO, are not covered in this guide.

7.1.3. Reference Installation

Reference Implementation

The Template Definition Exchange Standard (ITS) specified here has already been implemented in an open source project since the year 2009 and it could be proven that it fulfills all practical use cases of more than 50 clinical templates specification and implementation projects regarding template design, template versioning, template relationship and deployment as well as use of templates in creation, validation and processing XML instances including the usage in Template Repositories.

It was used in CDA R2 projects as well as in Clinical Statement (Care Provision) messages and non-HL7 XML specifications. Also, the following transformation scripts (XSLT and Xquery) and services are available

- *Template ITS → HTML tabular view (with a further path to PDF representation)*
- *Template ITS → ISO Schematron ("schematron-engine", including open / closed template behavior)*
- *RESTful repository functionalities to access for example value sets and templates*

7.2. Template Metadata and Design Body

According to the previous chapters in this specification, templates have meta information about their identity, effective date, status etc. referred to as the Template **Metadata** and the actual set of constraint definitions referred to as the Template Design **Body**.

The following figure gives an overview about the template "model" and the following sections specify the template metadata and design body elements that are used to define a whole template.

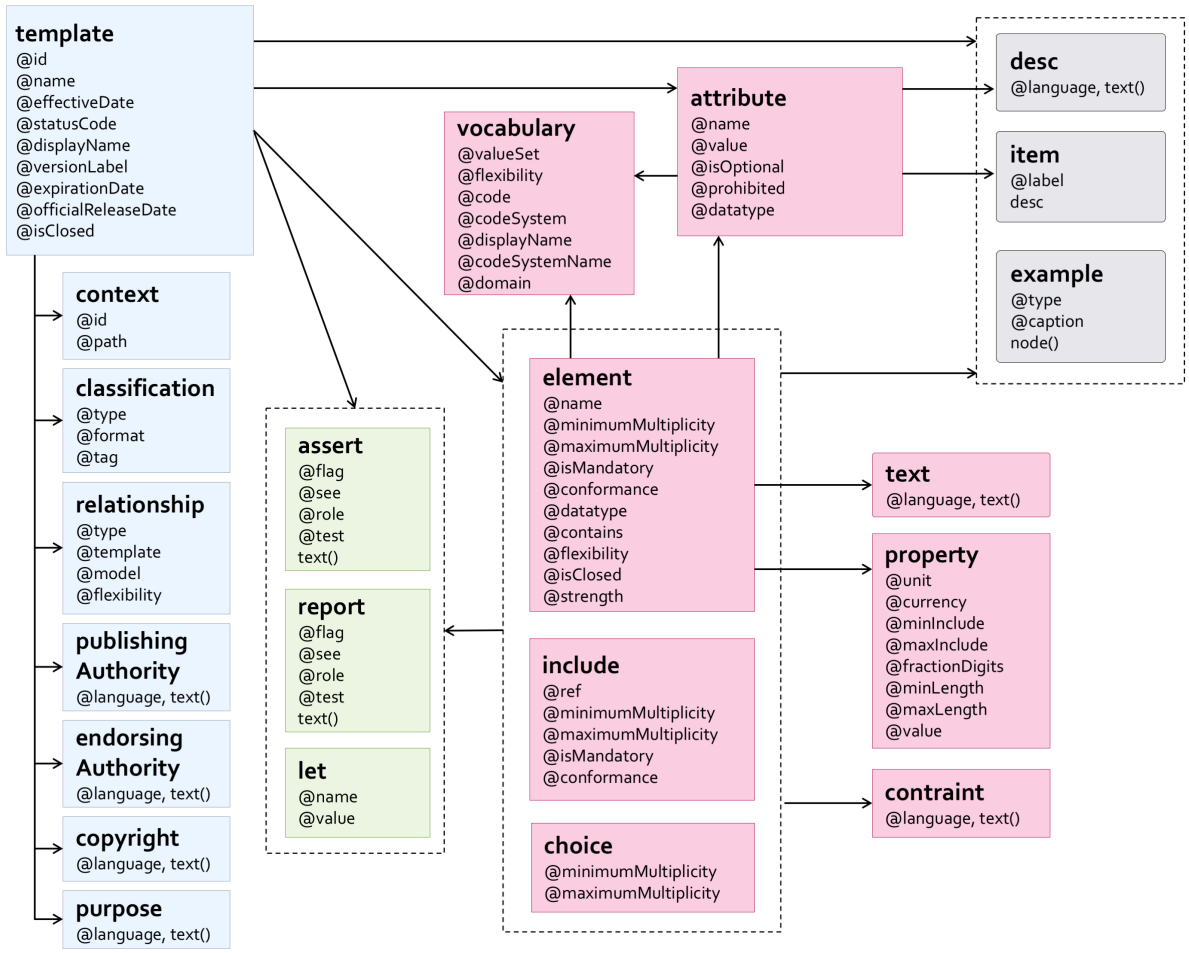


Figure 13: Schematic overview of "Template Model": left hand classes in blue belong to the metadata, mid red and green classes are part of the design body and the gray classes on the right may be part of the metadata or the design body.

7.2.1. Data types used for this specification

The elements and attributes of the ITS are associated with data types. Most of these data types are drawn from the MIF data types [mif] and are mainly restrictions of string types. The list of used data type definitions is as follows.

Table 15: Used data types used for templates definition elements and attributes

Data type	Description
AnyURI	Equivalent to the W3C schema type xs:anyURI
BasicIdOrOid	Either a string or an OID in order to identify the object
Boolean	Boolean "true" or "false"
StaticOrDynamic Flexibility	Either the fixed string "dynamic" or a valid W3C schema type xs:dateTime as an effective date of the version of the object

Data type	Description
Enumeration	Used as the base class for all "codes" which qualify or shall be handled as a W3C schema NMTOKEN
FreeText	Allows (unconstrained) markup and identifies the language in which it is expressed in the @language attribute of the element of that type; it is recommended that authoring systems restrict the markup of the content to the following • bold, italic, underline, strikethrough • header 1 to 5 • subscript, superscript • indentation • font color, family and size • text alignment, line breaks, paragraphs • whitespace interpretation • unordered and ordered lists • images • HTML links
ShortFormalName	Used when names are intended to be non-empty, short and without "weird" characters
NonEmptyString	A non-empty string
Oid	ISO Object Identifier, used for identifiers
SelfReference TemplateIdOrOid	A string "*" or "**" or a valid OID
TimeStamp	A time stamp according W3C schema type xs:dateTime
SmallNonNegative Integer	Integer number up to 6 characters long
UnlimitedInteger	Integer number up to 6 characters long or "*" to indicate "unbounded", used for multiplicity definitions

7.2.2. Overview

The following table gives an overview of the template metadata structure.

Table 16: Template Metadata elements overview

Item	DT	Card	Conf	Description
template				Root element
@id	Oid	1..1	M	Identification
@name	ShortFormal Name	1..1	M	Business name
@effectiveDate	TimeStamp	1..1	M	Creation Date

Item	DT	Card	Conf	Description
@statusCode	Enumeration	1..1	M	Status
@displayName	NonEmpty String	0..1	R	Print Name
@versionLabel	NonEmpty String	0..1	R	Version Label
@expirationDate	TimeStamp	0..1	R	Expiration Date
@officialReleaseDate	TimeStamp	0..1	R	Official Release Date
@isClosed	Boolean	0..1	R	Open/Closed Template
desc	FreeText	0..*	R	Multilingual description of purpose and scope
classification		0..*	R	Classification
@type	Enumeration	0..1	R	Type of Template
@format	Enumeration	0..1	R	Format of Template
tag	NonEmpty String	0..*	R	Tags for search purposes
property	NonEmpty String	0..*	R	Tags to specifiy any number of properties not part of the standard metadata, e.g. for operationalizations
relationship		0..*	R	Relationships
@type	Enumeration	1..1	M	Type of Relationship
@template	BasicIdOrOid	0..1	C	Related Template name or id (preferred: id)
@model	NonEmpty String	0..1	C	Related Template name or id (preferred: id)
@flexibility	StaticOr Dynamic Flexibility	0..1	O	Static or dynamic flexibility indicator
	Conditions			@template or @model
publishingAuthority	NonEmpty String	0..1	R	A statement about the publishing authority
endorsingAuthority	NonEmpty String	0..1	R	A statement about of the endorsing authority
purpose	FreeText	0..*	R	Multilingual statement about the purpose of the template

Item	DT	Card	Conf	Description
copyright	FreeText	0..*	R	Multilingual copyright statement(s) relating to the template and/or its contents
context		0..1	R	
@id	SelfReference TemplateIdOr Oid	0..1	C	Type of template id as context, either "*" or "**" or an OID
@path	AnyURI	0..1	C	An Xpath expression
	Conditions			Either @id or @path
item		0..1	R	Template item label and description
@label	NonEmpty String	1..1	M	String to identify item during validation
desc	FreeText	0..1	R	Further explanation of the item label
example		0..*	R	XML example
@type	Enumeration	0..1	R	Type of example, valid or error
@caption	NonEmpty String	0..1	R	Caption for the example
(element content)	(any well-formed XML)	1..1	M	The example XML fragment

7.3. Template elements and their attributes

7.3.1. template/@id

A mandatory globally unique, non-semantic, identifier for the template as the primary identifier. It is the identification of the purpose or intent of the template and is typically an OID. The id or the name of a template is chosen when referencing the template in another design, such as inclusion or containment.

Example fragment

```
<template id="2.16.840.1.113883.10.20.22.4.31" …
```

7.3.2. template/@name

A required name as a business name for the template.

Please note that there is no guarantee that the name is globally unique but it shall be unique within a governance group. Id of a template is chosen when referencing the template in another design, such as inclusion or containment.

Example fragment

```
<template … name="AgeObservation" …
```

7.3.3. template/@effectiveDate

The template has a mandatory timestamp (date and optional time) after which the template existed regardless of its state (e.g. still in design phase or ready for use). See also definitions in section 2.9.3.

Example fragment

```
<template … effectiveDate="2013-11-13T00:00:00" …
```

7.3.4. template/@statusCode

The template has a mandatory status code. It is one of the following codes.

Table 17: Enumeration values for the status code of templates

Code	Description
draft	draft
pending	pending
active	active
review	review
retired	retired / deprecated
cancelled	cancelled
rejected	rejected
terminated	terminated

Example fragment

```
<template … statusCode="active" …
```

See also definitions in section 2.9.6

7.3.5. template/@displayName

The optional human readable print name for the template for orientation purposes. This is not intended for machine processing and typically created in the project's language or in English.

7.3.6. template/@versionLabel

A version of a template may contain an optional human readable version label for the template to be able to determine the correct version of a template.

See also definitions in section 2.9.4.

Example fragment

```
<template … versionLabel="v2.0" …
```

7.3.7. template/@expirationDate

The optional date at which the design represented by this template version becomes stale, and for example should be reviewed for (clinical) relevance and accuracy. A new version should be considered. The expiration date is set to indicate that it should/shall no longer be used in the design of other templates (by anyone). Typically the status of a template with an expiration date has status "retired" on and after that date.

See also definitions in section 2.9.7.

7.3.8. template/@officialReleaseDate

An optional official release date of the template.

See also definitions in section 2.9.8.

7.3.9. template/@isClosed

An optional Boolean value that indicates whether the template is defined as closed or not. The default is "false" (i.e. template is defined as "open").

An "open" template should not have *@isClosed* or *@isClosed* set to "false". A "closed" template shall have an *@isClosed* set to "true".

Example fragment

```
<template … isClosed="true" …
```

On discussion and further explanation on "open" vs. "closed" template definitions refer to section 2.11.

7.3.10. template/desc

An optional free text natural language description of the intent and scope of the template. The purpose is to provide the author's initial intent for the template. The format is string with optional HTML tagging (XHTML). Please note that this element is repeatable, one per language.

7.3.11. template/desc/@language

The language indicator in which language the description is done. Examples are "en-US", "de-DE" etc.

7.3.12. template/classification

The classification is an optional and repeatable element that allows the classification of the template (see also section 2.9.10).

7.3.13. template/classification/@type

This attribute is an optional indication of the type of the template. For types of template refer to section 2.12. The type is a code (enumeration) with a choice one of the following core items:

Table 18: Enumeration values for the types of templates (extensions allowed)

Code	Description
cdadocumentlevel	CDA document level
cdaheaderlevel	CDA header level
cdasectionlevel	CDA section level
cdaentrylevel	CDA entry level
messagelevel	V3 or v2.xml message level
controlactlevel	V3 message level control act
payloadlevel	V3 message level payload
clinicalstatementlevel	In a V3 message, a clinical statement constraint
segmentlevel	v2.xml segment level
datatypelevel	Data type (flavor) level
fhirprofile	FHIR profile
vmrcdsprofile	vMR-CDS profile

Example fragment

```
<classification type="cdaheaderlevel"/>
```

It should be mentioned that there are templates with the same root element name, e.g. like participant, that can act as header or entry level template. The template type should be set appropriately.

7.3.13.1. template/classification/@format

The format of the instance the template constrains. The format is a code (enumeration) with a choice one of the following core items:

Table 19: Enumeration values for the format of templates (extensions allowed)

Code	Description
hl7v3xml1	HL7 Version 3 XML ITS R 1, including CDA; this is the default
hl7v2.5xml	HL7 Version 2.5 XML
fhirxml	FHIR expressed in XML
vmrxml	vMR-CDS XML

In the future, there might be other types of instances, e.g. “greencda” for green CDA instances or “genxml” for generic XML etc.

7.3.13.2. template/classification/tag

Tag elements could be incorporated to specify any number of tags for search purposes.

Example fragment

```
<classification type="cdaentrylevel">
  <tag>Blood pressure</tag>
</classification>
```

7.3.13.3. template/classification/property

Tag elements could be incorporated to specify any number of properties not part of the standard metadata, e.g. for operationalization instructions.

Example fragment

```
<classification type="messagelevel" format="vmrxml">
  <property>Data Expected if No Restriction</property>
</classification>
```

7.3.13.4. template/relationship

This optional and repeatable element defines relationships of the template regarding other templates or model artifacts. The type of relationship is indicated in the @type attribute.

The relationship element shall have a value in either @template or @model, but not both. The attribute @flexibility reflects the binding type for @template or @model, depending on which is valued.

It is recommended to define at least a relationship to the underlying model.

7.3.13.5. template/relationship/@type

The type of the relationship of the template, at this point in time one of the following.

Table 20: Enumeration values for the types of templates

Code	Description
Template-Template-Relationships	
REPL	This template replaces @template
VERSION	This template is a version of @template*
SPEC	This template specializes @template
GEN	This template generalizes @template
COPY	This template is a design copy of @template
ADAPT	This template is an adaptation of @template
EQUIV	This template is equivalent to @template regarding design
BACKWD	This template is backward compatible to @template
FORWD	This template is forward compatible to @template
USES	This template uses @template*

Code	Description
USEDBY	This template is used by @template*
Template-Model-Relationships	
DRIV	This template is derived from @model
* in a proper registry, this relationship is derived automatically	

Example fragment

```
<relationship type="SPEC" template="2.16.840.1937.10.101"
   flexibility="dynamic"/>

<relationship type="ADAPT" template="1.3.1937.10.102"
   flexibility="2013-02-10T00:00:00"/>

<relationship type="DRIV" model="some-model-name-or-id"/>
```

For further discussion refer to chapter 3.

7.3.13.6. template/relationship/@template

A reference by id to a template identifier, used for template-template relationships. It is required to use *@template* with an id but a template viewer may render business name or display name as well for convenience.

7.3.13.7. template/relationship/@model

A reference to a model artifact identifier or description specifying from what this template is derived from or based on, e.g. an HL7 R-MIM, a DCM, used for template-model relationships.

7.3.13.8. template/relationship/@flexibility

Static or dynamic binding for the template or model that is referred to. Default value is "dynamic". *@flexibility* takes either the keyword "dynamic" or – in order to define a static binding – a version date.

7.3.14. template/context

An optional context of a template, i.e. where in an XML instance the template rules are considered to apply to.

For further discussion refer to section 2.9.12. Typically, there are three types of context specifications.

7.3.14.1. Sibling node context

The template rules apply to all sibling elements and descendent nodes in the instance.

Example fragment

```
<context id="*"/>
```

7.3.14.2. Parent node context

The template rules apply to the siblings of the parent node and all descendent nodes in the instance. CDA entry level templates are often defined in parent node context.

Example fragment

```
<context id="**"/>
```

7.3.14.3. Pathname specified context

A pathname (making use of XPath expressions) is specified, which allows to trigger templates in an instance without the need to have template id elements in the instance.

Example fragment

```
<context path="cda:recordTarget"/>
```

The path context is especially useful for non HL7 Version 3 XML definitions, such as v2.xml, SOAP, ebXML etc.

7.3.15. template/item

The optional item is used as an identification mechanism (aka conformance labels) when it comes to error or warning or information messages by derived validation scripts. For example, if each constraint has a (unique) number, it may be used to precede every validation message.

Example

A CDA "encounter" section level template defines the section code to be 46240-8 drawn from LOINC. This constraint is uniquely identified by an identifier at or near the end of the constraint (e.g., CONF:7941). In the template definition, this is done specifying:

```
<item label="CONF:7941"/>
```

Every error, warning or information that is raised during validation carries this constraint item identifier.

This item is optional but it is recommended that a template item is specified in the metadata (or one is generated as a default).

Please note: If you clone a template and create an adaptation of it, it is best practice to delete the original item / conformance labels, since conformance labels are not part of the normative specification.

7.3.15.1. template/item/@label

Template item label as a string. The default shall be the template name or id (preferred: id).

7.3.15.2. template/item/desc

Further explanation of the item label.

7.3.15.3. template/example

Templates may have zero to many example fragments to illustrate valid (or explicitly invalid) instances. In addition, each template element definition (see below) may have also an example fragment.

7.3.15.4. template/example/@type

The optional type of the example, see the following table for valid codes.

Table 21: Enumeration values for the types of template examples

Code	Description
valid	The example is explicitly marked as valid. This is the default.
error	The example is explicitly marked as invalid

7.3.15.5. template/example/@caption

An optional text used as caption for the example.

The following statements could for example be rendered appropriately in the template documentation

```
<example type="valid" caption="A valid example with a piece of text">
  <root>
    <element value="abc"/>
    <text mediaType="text/plain">I am the text example</text>
  </root>
</example>
```

Example — A valid example with a piece of text

```
<root>
   <element value="abc"/>
   <text mediaType="text/plain">I am the text example</text>
</root>
```

```
<example type="error" caption="value/@unit must be a proper UCUM unit">
  <observation classCode="OBS" moodCode="EVN">
    <value xsi:type="PQ" value="6.7" unit="inch"/>
  </observation>
</example>
```

Example — value/@unit must be a proper UCUM unit

```
<observation classCode="OBS" moodCode="EVN">
   <value xsi:type="PQ" value="6.7" unit="inch"
</observation>
```

7.4. Template design body definitions

The actual template design, also seen as the "body" of the template, is a collection of constraints that ideally describes the structure and semantics of all instance elements. It contains the following items

- template **element definitions** with possible vocabulary, property or element content (text) constraints,
- template element **attribute definitions** with possible vocabulary constraints,
- **choices** of template elements,
- **inclusions** of other templates and
- to support at least one typical constraint language **schematron** rules (assert, report, let) are allowed.

The following table gives an overview over all possible template design body elements.

Table 22: Template Design Body elements overview

Item	DT	Card	Conf	Description
element				An XML element definition
@name	AnyURI	1..1	M	The name of the XML element
@minimum Multiplicity	SmallNon Negative Integer	0..1	R	The minimum multiplicity of the element
@maximum Multiplicity	Unlimited Integer	0..1	R	The maximum multiplicity of the element or "*" to express unbounded multiplicity
@isMandatory	Boolean	0..1	R	Whether the element is mandatory
@conformance	Enumeration	0..1	R	The conformance type of the element
@datatype	Enumeration	0..1	R	The data type of the element
@contains	AnyURI	0..1	R	A containment of another template
@flexibility	StaticOr Dynamic Flexibility	0..1	R	The associated flexibility of the contained template
@strength	Enumeration	0..1	R	Coding strength of a vocabulary binding, either CNE (Coded with no Exceptions) or CWE (Coded with Exceptions)
@isClosed	Boolean	0..1	R	Whether the element and all subsequent definitions are defined as open or closed

Item	DT	Card	Conf	Description
@id	AnyURI	0..1	O	A unique identifier of this template element
desc		0..*	R	Multilingual description of the element
item		0..1	R	Element item label and description
example		0..*	R	XML example
vocabulary		0..*	R	Vocabulary binding specification for that element
property		0..*	R	Property specification for that element
text		0..*	R	Element content specification for that element
Following allowed child elements of "element" are • **element** • **attribute** • **choice** • **include** • **assert** • **report** • **let** • **constraint**				
attribute				An XML attribute definition
@name	Enumeration Token	1..1	M	
@value	NonEmpty String	0..1	R	A fixed value for the attribute
@isOptional	Boolean	0..1	R	Attribute is optional
@prohibited	Boolean	0..1	R	Attribute is prohibited
@datatype	Enumeration	0..1	R	The data type of the attribute
@id	AnyURI	0..1	O	A unique identifier of this template attribute
desc		0..*	R	Multilingual description of the attribute

Item	DT	Card	Conf	Description
item		0..1	R	Attribute item label and description
vocabulary		0..*	R	Vocabulary binding specification for that attribute
choice				A choice
@minimumMultiplicity	SmallNonNegativeInteger	1..1	M	The minimum multiplicity of the elements in choice
@maximumMultiplicity	UnlimitedInteger	1..1	M	The maximum multiplicity of the elements in choice or "*" to express unbounded multiplicity
Following allowed child elements of "choice" are • **element** • **choice** • **include** • **constraint**				
include				Inclusion of another template
Following allowed child elements of "include" are • **element** • **attribute** • **choice** • **include** • **constraint**				
assert				Schematron assert
@flag	NonEmptyString	0..1	R	Schematron flag: to convey state or severity information to a subsequent process
@see	NonEmptyString	0..1	R	URI value of the external information of interest
@role	Enumeration	1..1	R	"error", "warning" or "information"
@test	NonEmptyString	1..1	R	The test expression
(element content)		0..1	R	The schematron message

Item	DT	Card	Conf	Description
report				Schematron report
@flag	NonEmpty String	0..1	R	Schematron flag: to convey state or severity information to a subsequent process
@see	NonEmpty String	0..1	R	URI value of the external information of interest
@role	Enumeration	1..1	R	“error”, “warning” or “information”
@test	NonEmpty String	1..1	R	The test expression
(element content)		0..1	R	The schematron message
let				Schematron let
@name	NonEmpty String	1..1	R	Schematron let name of the variable
@value	NonEmpty String	1..1	R	Schematron let value
constraint				Constraint
(element content)		0..1	R	A constraint in natural language, e.g. a predication table

7.4.1. element

7.4.1.1. element/@name

The @name attribute carries the name of the element, typically prefixed by the default project namespace (e.g. “hl7:” or “cda:”).

Example fragment

```
<element name="hl7:recordTarget" … >
```

The name may be an Xpath expression with predicates.

7.4.1.2. element/@minimumMultiplicity

This optional attribute identifies the minimum number of repetitions of this element that may occur. This is a “small” non-negative integer.

7.4.1.3. element/@maximumMultiplicity

This optional attribute identifies the maximum number of repetitions of this element that may occur. This is either a "small" non-negative integer or "*" to indicate "unbound" repetitions.

Example fragments

```
<element … maximumMultiplicity="2" … />
<element … maximumMultiplicity="*" … />
```

7.4.1.4. element/@isMandatory

This optional boolean attribute identifies that this element is mandatory, i.e. in an instance it shall be populated with a valid value.

Please note that the mandatory elements have a minimum cardinality of 1.

7.4.1.5. element/@conformance

This optional attribute identifies the conformance of this element. Allowed values are shown in the following table.

Table 23: Enumeration values for the conformance of an element

Code	Description
R	Required
NP	Not present
O	Optional
C	Conditional

Please note that the conformance cannot conflict with the cardinality, for example, NP cannot be combined with a cardinality of 1..1.

7.4.1.6. element/@datatype

This optional attribute specifies the data type or a data type flavor for this element.

Examples
CD
CE
ST
TS
IVL_TS
II.BSN.NL
TS.DATE.MIN

Please note that this specifies the data type or flavor of the element for documentation purposes but schematron engines may produce appropriate tests on the definitions per element based on these declarations.

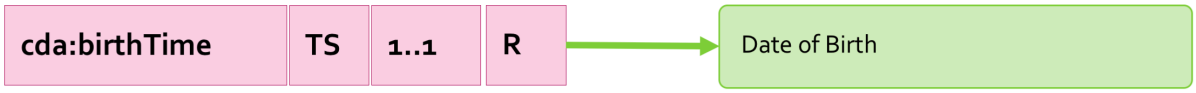
7.4.1.7. element/@id

This optional attribute is an id that allows the element to be uniquely identified. This is typically done by an OID. The id can be used to link between a template element and an associated concept for example in a data set or functional model.

Examples

A CDA recordTarget has an element "cda:birthTime" that is associated with a model attribute representing the birth time of the patient. If the birthTime template element gets an element id, it can be linked to a concept representation for example in a data set that shows the functional perspective.

The following figure shows a possible representation of a link between the birthTime element in the template definition (with datatype and cardinality) and a data element in a healthcare provide oriented model called “Date of birth”.



7.4.1.8. element/@contains

A *@contains* attribute indicates that the element contains the template specified by *@contains*. This may be a valid template name or template id. It is required to use *@template* with an id but a template viewer may render business name or display name as well for convenience.

Please note that a contained template typically influences the predicate of the element. See sections 6.5 about re-use of templates and section 2.10.2, Inclusion and Containment, for further information.

Example fragment: the CDA entry contains a template with id *1.3.1937.10.102*

```
<element name="cda:entry" contains="1.3.1937.10.102" …
```

7.4.1.9. element/@flexibility

An optional static or dynamic binding for the rule set that is referred to by *@contains*. Default value is “dynamic”, i.e. use the most recent version of the specified contained template. *@flexibility* takes either the keyword “dynamic” or – in order to define a static binding – a version date.

Example fragment: the cda entry contains a template with id 1.2.3.4.5 as of 2013-11-23 (the version of that template)

```
<element name="cda:entry" contains="1.2.3.4.5" flexibility="2013-11-23"…
```

7.4.1.10. element/@strength

This optional attribute specifies the vocabulary binding strength for data types that support that feature. Examples include CD, CE, CV and CO. Values are *required*, *extensible*, *preferred* and *example* (see table below). The default is *required*. Mismatches with required typically lead to validation errors (SHALL logic). Mismatches with extensible typically lead to validation warnings (SHOULD logic). Mismatches with preferred typically lead to validation information messages (MAY logic). Mismatches with example should not have any validation consequences.

Table 24: Enumeration values for the format of templates (extensions allowed)

Code	Display	Description
required (or CNE*)	Required/ CNE	Coded with no exceptions; this element SHALL be from the specified value set
extensible (or CWE*)	Extensible/ CWE	Coded with Exceptions; this element SHALL be from the specified value set if any of the codes within the value set can apply to the concept being communicated. If the value set does not cover the concept (based on human review), alternate codings (or, data type allowing, text) may be included instead.
preferred	Preferred	Instances are encouraged to draw from the specified codes for interoperability purposes but are not required to do so to be considered conformant.
example	Example	Instances are not expected or even encouraged to draw from the specified value set. The value set merely provides examples of the types of concepts intended to be included.

* CNE and CWE are deprecated

7.4.1.11. element/@isClosed

This optional Boolean attribute allows specifying that the element and subsequent child nodes are considered as open (other elements than specified allowed) or closed (no other elements than specified allowed). The rationale is the same as for open or closed templates, except that this definition applies to element and its child elements only.

See also section 2.11 on open vs. closed templates.

7.4.2. element/desc

This optional sub-element carries the multilingual textual description or further explanation on the use of the element.

7.4.3. element/item

Indicates the item label of the element for errors, warnings etc., see also section 7.3.15 about Template items.

7.4.4. element/example

An template element may have zero or more examples or example fragments for the element, see also section 7.3.15.3 about Template examples.

7.4.5. element/vocabulary

This optional sub-element is used for elements with "coded" data types. It allows the assertion of one or more codes, code systems, value sets or concept domains.

Valid combinations

- A @*valueSet* (by name or id, preferred: id) plus an optional @*flexibility* attribute which is either the fixed string "dynamic" or a valid date and time as an effective date of the version of the value set (static).
- A @*code* and (depending on the data type) a @*codeSystem*; it may contain human readable attributes such a @*displayName* and @*codeSystemName* which, if present, constraints the presence of these attributes in instances.
- An @*domain* specification by name, defining the abstract concept domain(s) from which proper codes may be drawn.

Example of value set bindings, dynamic and static

```
<element name="hl7:code" … datatype="CV">
  <vocabulary valueSet="MyValueSet" flexibility="dynamic"/>
</element>

<element name="hl7:code" … datatype="CE">
  <vocabulary valueSet="MyValueSet" flexibility="2013-11-24"/>
</element>
```

Example of a fixed code/code system binding

```
<element name="hl7:code" … datatype="CE">
  <vocabulary code="1234" codeSystem="1.2.3.4.5"/>
</element>
```

If multiple vocabulary elements are specified the definitions are logically connected by an "OR".

Example: Code must come from Value Set AAA or BBB or shall be code '1234' from codeSystem '1.2.3.4.5'

```
<element name="hl7:code" … datatype="CE">
  <vocabulary valueSet="AAA" flexibility="dynamic"/>
  <vocabulary valueSet="BBB" flexibility="dynamic"/>
  <vocabulary code="1234" codeSystem="1.2.3.4.5"/>
</element>
```

7.4.5.1. element/vocabulary/@valueSet

This attribute specifies the value set by name or id (preferred: id) where codes shall be drawn from. It is required to use *@valueSet* with an id but a template viewer may render business name or display name as well for convenience.

7.4.5.2. element/vocabulary/@flexibility

This attribute specifies the flexibility for the value set. *@flexibility* takes either the keyword "dynamic" or – in order to define a static binding – a version date.

7.4.5.3. element/vocabulary/@code

This attribute specifies the code that shall be used in the instance.

Example: Code for "procedure indication" must be code 429969003 from Snomed-CT

```
<element name="hl7:code" … datatype="CE">
  <vocabulary code="429969003"
     codeSystem="2.16.840.1.113883.6.96"/>
</element>
```

Example: Code for "procedure indication" must be code 429969003 from Snomed-CT or a post-coordinated term 444783004:246513007=134433005 from Snomed-CT

```
<element name="hl7:code" … datatype="CE">
  <vocabulary code="429969003"
     codeSystem="2.16.840.1.113883.6.96"/>
  <vocabulary code="444783004:246513007=134433005"
     codeSystem="2.16.840.1.113883.6.96"/>
</element>
```

7.4.5.4. element/vocabulary/@codeSystem

This attribute specifies the code system (e.g. an OID) that shall be used in the instance.

7.4.5.5. element/vocabulary/@displayName

This attribute specifies the human readable display name that shall be used in the instance.

7.4.5.6. element/vocabulary/@codeSystemName

This attribute specifies the human readable code system name that shall be used in the instance.

7.4.5.7. element/vocabulary/@domain

This attribute specifies the conceptual domain by name where codes shall be drawn from.

Example of a conceptual domain binding

```
<element name="hl7:code" … datatype="CE">
  <vocabulary domain="TheConceptDomain"/>
</element>
```

Please note that specifying a domain typically leads to no testable properties in an instance. A conceptual domain is used for coded elements in templates that are typically further constrained or that leaves a concrete binding to a vocabulary open by intention.

7.4.6. element/property

The property sub-element is used for elements of type quantity, string or boolean. It allows the assertion of one or more units, ranges, fraction digits or fixed values.

Table 25: Property definitions for an element

Property attribute	Attribute type	Description	Examples
@unit	string (code)	A proper (UCUM) unit.	cm mm[Hg]
@currency	string (code)	A currency unit.	EUR
@minInclude	integer real	An integer or decimal number to specify the lower inclusive interval boundary	1.00
@maxInclude	integer real	An integer or decimal number to specify the upper inclusive interval boundary	38
@fractionDigits	positive integer	A small positive integer that specifies the number of fractional digits. A fraction digit "2" requires at least two fraction digits to be present in the instance, otherwise the instance is in error. If fraction digit is suffixed with a "!", e.g. "2!" this means the presence of exactly two fraction digits is required	2 2!
@minLength	positive integer	A positive integer number to specify the minimal inclusive length of a string	1
@maxLength	positive integer	A positive integer number to specify the maximal inclusive length of a string	10
@value	string	A string as a fixed value. In the future also regular expressions may be allowed here.	XYZ

Valid combinations of attributes are:

- @value @unit (for physical quantities)
- @unit @minInclude @maxInclude @fractionDigits (for physical quantities in @value)
- @currency @minInclude @maxInclude @fractionDigits (for monetary amounts in @value)
- @minLength @maxLength (for strings)
- @value (fixed values in @value)
- @value @currency(for monetary amounts)

Example of specifying that the value PQ shall be 0..200 and unit shall be 'cm'

```
<element name="hl7:value" … conformance="R" datatype="PQ">
  <property minInclude="1" maxInclude="200" unit="cm" />
</element>
```

If multiple property elements are specified the definitions are logically connected by an "OR".

Example of specifying that the value PQ shall be 1 .. 200 cm or 0.01 .. 2.00 m with exactly 2 fraction digits

```
<element name="hl7:value" … datatype="PQ">
  <property minInclude="1" maxInclude="200" unit="cm"/>
  <property minInclude="0.01" maxInclude="2.00" unit="m"
     fractionDigits="2!"/>
</element>
```

7.4.7. element/text

This optional sub-element specifies the element content in an instance. This is used for elements of type ST, ED etc.

Example of specifying a fixed text for the element content in the instance

```
<element name="hl7:text" … datatype="ST">
  <text>Fixed text to be used in the instance<text/>
</element>
```

If multiple text elements are specified the definitions are logically connected by an "OR".

7.4.8. Attribute

This definition allows to specify attributes for template elements and their properties.

7.4.8.1. attribute/@name

Specifies the name of the XML attribute. It is typically used in conjunction with a *@value*.

Example: a template element definition with attribute 'classCode' to be valued 'OBS'

```
<element name="hl7:observation" … >
```

```
  <attribute name="classCode" value="OBS"/>
</element>
```

Example: attribute 'negationInd' shall be set to 'true'

```
<attribute name="negationInd" value="true"/>
```

7.4.8.2. attribute/@value

Specifies the value of the XML attribute given in *@name*.

7.4.8.3. attribute/@isOptional

This option attribute determines that the attribute is required to be valued in the instance.

Default is 'false', meaning that the attribute is NOT optional, and thus required.

Example: Defining *@classCode* to be 'OBS' if present

```
<attribute name="classCode" value="OBS" isOptional="true"/>
```

7.4.8.4. attribute/@prohibited

Determines that the attribute is prohibited to be in the instance.

Example: A nullFlavor is not allowed

```
<attribute name="nullFlavor" prohibited="true"/>
```

7.4.8.5. attribute/@datatype

This optional attribute allows specifying the data type of the XML attribute. Valid types are shown in the table below.

Table 26: Valid data types for an attribute of an element (extensions allowed)

Attribute data type	Meaning	Attribute Value Example
st	string, the default	"a string"
bl	boolean	"true"
ts	timestamp	"20130630"
int	integer	"1"
real	real	"0.4"
cs	simple code (cannot contain spaces)	"B64"
set_cs	set of codes	"HP WP"

Example

```
<attribute name="mediaType" datatype="cs"/>
```

7.4.9. attribute/desc

This optional sub-element carries the multilingual textual description or further explanation on the use of the element.

7.4.10. attribute/item

Indicates the item label of the element for errors, warnings etc., see also section 7.3.15 about Template items.

7.4.11. attribute/vocabulary

This optional sub-element is used for attributes with "coded" data types, i.e. "cs" or "set_cs". It allows the assertion of one or more value sets.
In validation it is assumed that multiple codes may be used as is possible in e.g. *@use* on datatypes AD, EN and TEL. In order to constrain to only one possible code, the ''@datatype'' attribute must be set to "cs" (only one single code allowed). Alternatively, and also the default, use "set_cs" (one or multiple space delimited codes allowed).

Example: required attribute containing one or more codes as listed for @use

```
<attribute name="use" datatype="set_cs">
  <vocabulary code="TMP"/>
  <vocabulary code="H"/>
  <vocabulary code="WP"/>
</attribute>
```

7.4.11.1. attribute/vocabulary/@valueSet

The value set where the code shall be drawn from, referenced by name or id (preferred: id) of the value set.

Example: required attribute containing one code from a value set

```
<attribute name="mediaType" datatype="cs">
  <vocabulary valueSet="MyMediaTypes" flexibility="dynamic"/>
</attribute>
```

Example: optional attribute containing a list of one to many codes from a value set

```
<attribute name="use" datatype="set_cs" isOptional="true">
  <vocabulary valueSet="AddressUse" flexibility="dynamic"/>
</attribute>
```

Another example is how to determine the set of allowed nullFlavors for any element. Assume the following definition of a *doseQuantity* element, then the included attribute definition constrains the set of allowed nullFlavors (defined in an value set shown below as well).

Example: null flavors for the *doseQuantity* element must be drawn from value set 'AllowableNulls' with OID 1.2.3.4.5

```
<element name="hl7:doseQuantity" minimumMultiplicity="1"
  maximumMultiplicity="1" conformance="R" datatype="IVL_PQ">
  <attribute name="nullFlavor" datatype="cs" isOptional="true">
    <vocabulary valueSet="1.2.3.4.5"/>
  </attribute>
</element>
```

7.4.12. choice

In some cases, a designer wants to offer a choice of template elements and restrict the choice to be n out of m (cardinality constraint). In order to be able to specify choices of elements the choice element can be used.

Examples

A typical example is the CDA header author definition where in the underlying model a choice is defined between an *assignedPerson* or an *assignedAuthoringDevice* playing the role of the author and where 0..1 playing entities may be chosen.

This can be expressed by specifying the following constraint

```
<choice minimumMultiplicity="0" maximumMultiplicity="1">
  <element name="assignedPerson">...</element>
  <element name="assignedAuthoringDevice">...</element>
</choice>
```

This offers the choice to select 0 or 1 out of the element *assignedPerson* or *assignedAuthoringDevice*.

If the minimumMultiplicity of the choice were 1, then it would basically say that either *assignedPerson* or *assignedAuthoring* device must be chosen, but it is an error to include neither or both. The definitions of *assignedPerson* or *assignedAuthoring* are not shown here.

Note, that the elements in the choice may have their own cardinality and it shall be possible to fulfil the choice's cardinality, e.g. if choice is 0..1, none of the elements may carry 1 as their minimum cardinality as this would make a legal empty choice impossible or would exceed the maximum elements in the choice.

If a choice has no cardinality definitions, the default is 0..*.

7.4.12.1. choice/@minimumMultiplicity

Optional, identifies the minimum number of elements out of the choice.

7.4.12.2. choice/@maximumMultiplicity

Optional, identifies the maximum number of elements out of the choice.

7.4.13. include

A template may reference another defined template by inclusion.

An inclusion within a template design makes use of another template by “virtually” copying the included template definitions, also known as transclusion. In essence this means that template definitions are included by reference and shown as-is on demand, i.e. at time of displaying the template or using it for the creation of validation scripts. Inclusion is automatic and transparent to the user.

Examples

A CDA document level template includes the definitions of a CDA *recordTarget author*, *documentationOf* and *setId+versionNumber*. These items are defined as "includable" templates because they may be used in other document level templates as well.

7.4.13.1. include/@ref

Mandatory reference to the template to be included, either by name or id (preferred: id).

Example: inclusion of a template with id *1.2.3.4.5*

```
<include ref="1.2.3.4.5"/>
```

Example: Typical start of a Clinical Document definition with includes

```
<element name="hl7:ClinicalDocument">
    <include ref="1.2.3.4.10.9.1"/>
    <element name="hl7:templateId" minimumMultiplicity="1"
        maximumMultiplicity="1" isMandatory="true" datatype="II">
        <desc language="en-US">CDA document template id
           for this kind of document</desc>
      <attribute root="2.16.840.1.113883.10.1"/>
    </element>
    <include ref="1.2.3.4.10.9.2"/>
    <element name="hl7:code"
        minimumMultiplicity="1" maximumMultiplicity="1"
        isMandatory="true" datatype="CE">
        <example>
            <!-- document type -->
            <code code="11524-6" codeSystem="2.16.840.1.113883.6.1"
               codeSystemName="LOINC" displayName="EKG study report"/>
        </example>
        <vocabulary code="11524-6" codeSystem="2.16.840.1.113883.6.1"/>
    </element>
    <include ref="1.2.3.4.10.9.3" minimumMultiplicity="0"
```

```
        maximumMultiplicity="1"> <!-- title -->
        <example>
            <title>EKG Report as of 1 February 2013</title>
        </example>
    </include>
    <include ref="1.2.3.4.10.9.4"/>  <!-- effectiveTime -->
    <include ref="1.2.3.4.10.9.5 "/> <!-- confidentialityCode -->
    <include ref="1.2.3.4.10.2.1" minimumMultiplicity="1"
        maximumMultiplicity="1"/>    <!-- recordTarget -->
    <include ref="1.2.3.4.10.2.2" minimumMultiplicity="1"
        maximumMultiplicity="1"/>    <!-- author -->
    <include ref="1.2.3.4.10.2.3"/>  <!-- custodian -->
    ...
</element>
```

7.4.13.2. include/@flexibility

An optional static or dynamic binding for the rule set that is referred to by *@ref*. Default value is "dynamic", i.e. use the most recent version of the specified referenced template. *@flexibility* takes either the keyword "dynamic" or – in order to define a static binding – a version date.

Example fragment: a template with id 1.2.3.4.5 as of 2013-11-23 (the version of that template) is to be included

```
<include ref="1.2.3.4.5" flexibility="2013-11-23"…
```

7.4.13.3. include/@minimumMultiplicity

The optional attribute identifies the minimum number of repetitions of all elements at top level of the included template.

The include statement may specify overriding cardinalities/conformances. If the included template has only one root element the cardinality/conformance of this element gets overridden by the cardinality/conformance specified in the calling include statement. If there are more than one root element in the included template, all element's cardinalities/conformances are overridden. The override does not affect attribute or choice definitions.

Examples

In the following situation, a *templateX* is included. Please note that regardless of the cardinality/conformance definitions of the root element *hl7:entry* in *templateX*, it gets overridden by 1..1 as specified in the include statement. Also note that this is an override, not a formally checked restriction and it shall be in alignment with a possibly underlying standard.

```
<template name="someTemplate" … >
  <element name="hl7:id" … />
  <include ref="templateX"
      minimumMultiplicity="1" maximumMultiplicity="1">
  …
</template>

<template name="templateX" … >
  <element name="hl7:entry"
      minimumMultiplicity="0" maximumMultiplicity="*" … />
</template>
```

7.4.13.4. include/@maximumMultiplicity

The optional attribute identifies the maximum number of repetitions of all elements at top level of the included template.

7.4.13.5. include/@isMandatory

This optional Boolean attribute identifies that all elements at top level of the included template are mandatory, i.e. in an instance it shall be populated with a valid value.

7.4.13.6. include/@conformance

This optional attribute identifies the conformance of all elements at top level of the included template.

7.4.13.7. include/desc

This optional sub-element carries the multilingual textual description or further explanation on the use of the *include*.

7.4.13.8. include/item

Indicates the item label of the include statement for errors, warnings etc., see also section 7.3.15 about Template items.

7.4.13.9. include/example

An include may have zero or more examples or example fragments for the *element*, see also section 7.3.15.3 about Template examples.

7.4.14. Schematron statements

To support at least one typical constraint language schematron rules (*assert, report, let*) are allowed in template definitions. They may be interspersed at defined locations and a generated in addition to the validation statements derived from the other template definitions, e.g. cardinality constraints, datatype restrictions etc.

Please note that the Templates ITS allows to intersperse constraint language schematron rules as specified below, i.e. *assert, report* and *let*, in a template design. By positioning *assert, report* and *let* somewhere in the template design

hierarchy in a context of an element, the according schematron *rule* statement that contains the *context* path is auto determined. In other words, you cannot "superimpose" an *assert* or *report* by a hand-written *rule* that gives another *context* than the position they appear within the template design.

In addition, schematron engine implementations typically and automatically determine the context by position of the *assert* or *report* within the template design.

Please note also, that ISO schematron *context* paths may not be appear more than once in a schematron file, otherwise one the first rule will be fired instead of all with the same context. To prevent clashes, no schematron phases, patterns or rules are allowed.

Schematron *pattern* can be handled by the conversion mechanism (schematron engine) that converts the template design into schematron files. Allowing *pattern* statements is out-of-scope for the Templates ITS.

7.4.14.1. assert

This statement will be converted to an appropriate schematron assert statement. They are often used to express co-constraints.

Example

```
<assert role="error" test="@negationInd or hl7:value">
  If the observation is not negated a value shall be present
</assert>
```

The assert statement is identically defined to the original schematron assert.

assert/@flag

Schematron flag to convey state or severity information to a subsequent process.

assert/@see

URI value of the external information of interest.

assert/@role

One of the following: "error", "warning" or "information".

assert/@test

Schematron test expression.

7.4.14.2. report

This statement will be converted to an appropriate schematron report statement. They are often used to express co-constraints.

The report statement is identically defined to the original schematron report.

report/@flag

Schematron flag to convey state or severity information to a subsequent process.

report/@see

URI value of the external information of interest.

report/@role

One of the following: “error”, “warning” or “information”.

report/@test

Schematron test expression.

7.4.14.3. let

A let statement defines a schematron runtime variable that can be used in assert or report or let statements.

Example: get the @code in the value element of the observation classified by code/@code 123-4

```
<let name="lepcode"
   value="//hl7:observation[hl7:code[@code='123-4']]/hl7:value/@code"/>
```

let/@name

Schematron let name of the variable

let/@value

Schematron let value

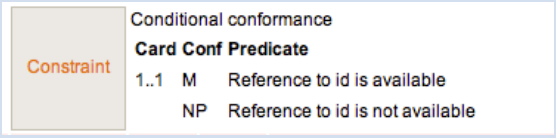
7.4.14.4. constraint

A constraint in natural language, e.g. a predication table for a conditional statement. If it is possible to compute the constraint, it is typically expressed as a schematron rule.

Example of a textual constraint in a template definition and a possible rendition

```
<constraint language="en-US">Conditional conformance
    <table>
        <tr>
            <th>Card</th>
            <th>Conf</th>
            <th align="left">Predicate</th>
        </tr>
        <tr>
            <td>1..1</td>
            <td>M</td>
            <td>Reference to id is available</td>
        </tr>
        <tr>
            <td/>
            <td>NP</td>
            <td>Reference to id is not available</td>
```

```
        </tr>
    </table>
</constraint>
```



Example of a computable constraint in a template definition and the schematron rule

```
<constraint ...>Follow the rules for addresses</constraint>
<assert role="error"
  test="hl7:streetAddressLine or (hl7:streetName and hl7:houseNumber)">
  Either streetAddressLine or streetName+houseNumber</assert>
```

7.5. How to Create, Validate and Consume Template Definitions

The following section describes actions on Template Definition Instances not the XML instances of e.g. a Clinical Document.

7.5.1. Creation

There are multiple ways to create a Template Definition Instance.

The simplest way is to craft the definitions using an XML editor. To support this method, a W3C schema and accompanying schematron rules are available (see following section).

Meanwhile template authoring tools are available. The underlying rationale and the details of how template design is acquired, presented and stored may be different in these tools, but it is expected that – once this specification has been endorsed – an exchange of template definitions between these tools is easily possible.

7.5.2. Validation

For the validation of the template definitions itself a W3C schema with embedded schematron is available.

A Wiki page has been established in order to point to all recommended artifacts, see

http://wiki.hl7.org/index.php?title=Template_Instance_Validation

7.5.3. Consume

Regarding consuming template definitions several areas have to be mentioned:

- Documentation and publication of template definitions
- Using template definitions for XML instance validation including testing and certification purposes
- Creation of XML instance (fragments) according to the template definition, e.g. code generation for interfaces of applications
- Exposing template definitions in appropriate Template Repositories.

It is obvious that the template design has to be presented in an appropriate format for the tools they were designed with. In addition, for **documentation and publication**

purposes textual representations like in implementations guides may be required. Some governance groups use wiki pages to publish their template designs, in other cases extracts of the definitions are done by generating documents for text processors or HTML pages.

As of today the most prominent purpose of templates beside documentation and publication in various formats, is XML **instance validation**.

In order to be able to validate instances against a template, a transformation exists from the template exchange format to schematrons directly out of the template definitions. The underlying schematron engine creates ISO schematron rules and also compiles all value set references that are needed for a proper validation.

In some projects the template definitions have also been used to create XML instance fragments by means of XSLT transformations or to generate code out of the definitions.

An appropriate tool suite should have the ability to expose template definitions in a repository so that templates can be referred to or re-used in other template definitions, even between different governance groups.

7.6. How to exchange Template Definitions between Systems

One major objective of this ITS is to enable the exchange of template definitions between systems, either in terms of an export and import functionality or to expose template definitions in a Template Repository.

There are some typical functions, a Template Repository must provide. The following list is an example.

Examples of Template Repository Functions regarding Exchange of Template Definitions

Typical Functions

- Provide a **list of all templates** available in the Repository
- **Search** for a specific Template based on Metadata information
- **Retrieve** a complete Template in the Exchange Format
-

Additional Functions

- **List** Template **Metadata**
- **Allocate Template Id** for a new Template
- **Submit Template** to the Repository
- **Update Template** in the Repository
- **Remove Template** from the Repository

7.7. Best practice examples for CDA definitions

The following examples are derived from real projects using this ITS and show best practice for simple CDA template definitions. This document starts with simple section level template definitions, continues with entry and header level templates and ends with a whole document level template specification. If any examples are interspersed in the definition they are shown in italic font to make it easier to identify these parts as examples.

7.7.1. Example of a CDA section level template

A section level template "EKG Impression Section" is defined to capture the narrative results of an EKG study. The definition includes the corresponding *templateId*, the *section.code* as a LOINC code and declares both *section.title* and *section.text* as mandatory.

Example: EKG Impression Section

```
<template id="2.16.840.1.113883.3.1937.99.61.3.10.3001"
  name="EKGImpressionSection" displayName="EKG Impression Section"
  effectiveDate="2013-02-10T00:00:00" statusCode="active">
    <desc language="en-US">This section describes the impression
      (findings) of an EKG study of a patient</desc>
    <classification type="cdasectionlevel"/>
    <context id="**"/>
    <example>
        <section classCode="DOCSECT">
            <!-- Template id for EKG measurements -->
            <templateId root="2.16.840.1.113883.3.1937.99.61.3.10.3001"/>
            <code code="18844-1" codeSystem="2.16.840.1.113883.6.1"
               codeSystemName="LOINC"/>
            <title>Impression</title>
            <text>Normal sinus rhythm<br/> Ischemic ST-T changes in
               anterior leads<br/> Poor R Progression in right
               precordial leads</text>
        </section>
    </example>
    <element name="hl7:section">
        <attribute name="classCode" value="DOCSECT" isOptional="true"/>
        <!-- Element templateId -->
        <element name="hl7:templateId" minimumMultiplicity="1"
          maximumMultiplicity="1" datatype="II">
            <attribute name="root"
               value="2.16.840.1.113883.3.1937.99.61.3.10.3001"/>
        </element>
        <!-- Element code -->
        <element name="hl7:code" minimumMultiplicity="1"
          maximumMultiplicity="1" isMandatory="true" datatype="CD">
            <vocabulary code="18844-1"
               codeSystem="2.16.840.1.113883.6.1"/>
        </element>
        <!-- Element title -->
        <element name="hl7:title" minimumMultiplicity="1"
          maximumMultiplicity="1" isMandatory="true" datatype="ST"/>
        <!-- Element text -->
```

```
        <element name="hl7:text" minimumMultiplicity="1"
          maximumMultiplicity="1" isMandatory="true" datatype="SD.TEXT"/>
    </element>
</template>
```

The specifications could be easily rendered to HTML or a PDF representation as some tools do already.

If a section contains an entry this is typically expressed as the following.

Example fragment: the cda entry contains a template with id 1.2.3.4.5

```
<element name="cda:entry" contains="1.2.3.4.5" …
```

It is also possible to state static or dynamic binding for the template that is referred to by @contains.

Example fragment: the cda entry contains a template with id 1.2.3.4.5 as of 2013-11-23 (the version of that template)

```
<element name="cda:entry" contains="1.2.3.4.5" flexibility="2013-11-23"…
```

The template itself has to start as the example for an entry level template below shows.

7.7.2. Example of a CDA header level template

A header level template "CDA custodian" is defined to capture the custodian of a document. It is defined as a re-usable component that is normally used by inclusion in a document level template (see below). Therefore, it has no "context" as the context is determined by the point of inclusion in the including template.

The items defined for the custodian in this example includes an *assignedCustodian.representedCustodianOrganization.id* only.

Example: CDA custodian

```
<template id="2.16.840.1.113883.3.1937.99.61.3.10.2003"
  name="CDAcustodian" displayName="CDA custodian"
  effectiveDate="2013-12-05T00:00:00" statusCode="active">
    <desc language="en-US">Custodian of the document</desc>
    <classification type="cdaheaderlevel"/>
    <example>
        <custodian>
            <assignedCustodian>
                <representedCustodianOrganization>
                    <id root="2.16.840.1.113883.3.1937.99.3.2.997788"/>
                </representedCustodianOrganization>
            </assignedCustodian>
        </custodian>
    </example>
    <element name="hl7:custodian" minimumMultiplicity="1"
      maximumMultiplicity="1" isMandatory="true">
```



```
        <attribute name="typeCode" value="CST" isOptional="true"/>
        <element name="hl7:assignedCustodian" minimumMultiplicity="1"
          maximumMultiplicity="1" isMandatory="true">
            <attribute name="classCode" value="ASSIGNED"/>
            <element name="hl7:representedCustodianOrganization"
              minimumMultiplicity="0" maximumMultiplicity="1">
                <element name="hl7:id" minimumMultiplicity="1"
                  maximumMultiplicity="1" conformance="R" datatype="II"/>
            </element>
        </element>
    </element>
</template>
```

7.7.3. Example of a CDA entry level template

An entry level template "Age Observation" is defined to capture the age of a patient. The only variable spot in the template is the actual value of the observation and the @unit is also bound to the value set *AgePQ_UCUM* that contains typical units for age like "a" for year or "mo" for month.

Example: Age Observation

```
<template id="2.16.840.1.113883.3.1937.99.61.3.10.4001"
  name="AgeObservation" effectiveDate="2013-01-31T00:00:00"
  statusCode="draft" displayName="Age Observation">
    <desc language="en-US">This Age Observation represents...</desc>
    <classification type="cdaentrylevel"/>
    <context id="**"/>
    <example>
        <observation classCode="OBS" moodCode="EVN">
            <templateId root="2.16.840.1.113883.10.20.22.4.31"/>
            <code code="397659008" codeSystem="2.16.840.1.113883.6.96"
              displayName="Age"/>
            <statusCode code="completed"/>
            <value xsi:type="PQ" value="57" unit="a"/>
        </observation>
    </example>
    <element name="hl7:observation">
        <attribute name="classCode" value="OBS"/>
        <attribute name="moodCode" value="EVN"/>
        <element name="hl7:templateId" minimumMultiplicity="1"
          maximumMultiplicity="1" isMandatory="true">
            <attribute name="root"
              value="2.16.840.1.113883.10.20.22.4.31"/>
        </element>
        <element name="hl7:code" minimumMultiplicity="1"
          maximumMultiplicity="1" isMandatory="true">
```

```
            <vocabulary code="445518008"
               codeSystem="2.16.840.1.113883.6.96"/>
        </element>
        <element name="hl7:statusCode" minimumMultiplicity="1"
          maximumMultiplicity="1" isMandatory="true">
            <vocabulary code="completed"/>
        </element>
        <element name="hl7:value" minimumMultiplicity="1"
          maximumMultiplicity="1" isMandatory="true" datatype="PQ">
            <attribute name="unit">
                <!-- Age_PQ_UCUM value set
                     with id 2.16.840.1.113883.11.20.9.21 -->
                <vocabulary valueSet="2.16.840.1.113883.11.20.9.21"/>
            </attribute>
        </element>
    </element>
</template>
```

7.7.4. Example of a CDA document level template

A document level template "Minimal CDA" is defined. It defines the minimal requirements of a CDA document. A *CDATypeId* template is included (re-usable template for all kinds of document level templates), along with a template id, document id, document type code, title, effective time and confidentiality code. Subsequently the record target, author and custodian definitions are included, followed by the structured body that contains a single section template.

Example: Document Level Template of a minimal CDA document

```
<template id="2.16.840.1.113883.3.1937.99.61.3.10.1"
  name="MinimalCDAdocument" displayName="Minimal CDA document"
  effectiveDate="2013-12-05T00:00:00" statusCode="active">
    <desc language="en-US">A minimal CDA Release 2 document, ...</desc>
    <classification type="cdadocumentlevel"/>
    <context path="/"/>
    <example>
        <ClinicalDocument xmlns="urn:hl7-org:v3">
            <!-- CDA Header -->
            <typeId root="2.16.840.1.113883.1.3"
              extension="POCD_HD000040"/>
            <templateId root="2.16.840.1.113883.3.1937.99.61.3.10.1"/>
            <id extension="123456789"
              root="2.16.840.1.113883.3.1937.99.3.2.997788.1"/>
            <code code="11524-6" codeSystem="2.16.840.1.113883.6.1"
              codeSystemName="LOINC" displayName="EKG study report"/>
            <effectiveTime value="20131020122709"/>
            <confidentialityCode code="N"
```

```
            codeSystem="2.16.840.1.113883.5.25"/>
        <recordTarget><!-- .. --></recordTarget>
        <author><!-- .. --></author>
        <custodian><!-- .. --></custodian>
        <!-- CDA Body -->
        <component>
            <structuredBody>
                <component><!-- .. --></component>
            </structuredBody>
        </component>
    </ClinicalDocument>
</example>
<element name="hl7:ClinicalDocument">
    <include ref="1.2.3.4.10.9.1"/>
    <!-- template id for this document -->
    <element name="hl7:templateId" minimumMultiplicity="1"
      maximumMultiplicity="1" isMandatory="true" datatype="II">
        <desc language="en-US">CDA document template id for
          this kind of document</desc>
      <attribute name="root"
        value="2.16.840.1.113883.3.1937.99.61.3.10.1"/>
    </element>
    <!-- document id -->
    <element name="hl7:id" minimumMultiplicity="1"
      maximumMultiplicity="1" isMandatory="true" datatype="II"/>
    <!-- document type code -->
    <element name="hl7:code" minimumMultiplicity="1"
      maximumMultiplicity="1" isMandatory="true" datatype="CE">
        <example>
            <code code="11524-6" codeSystem="2.16.840.1.113883.6.1"
              codeSystemName="LOINC" displayName="EKG study report"/>
        </example>
        <vocabulary code="11524-6"
          codeSystem="2.16.840.1.113883.6.1"/>
    </element>
    <include ref="1.2.3.4.10.9.2" minimumMultiplicity="0"
      maximumMultiplicity="1">
        <example>
            <title>EKG Report as of 1 February 2013</title>
        </example>
    </include>
    <element name="hl7:effectiveTime" minimumMultiplicity="1"
      maximumMultiplicity="1" isMandatory="true"
      datatype="TS.DATETIME.MIN"/>
```



```
        <element name="hl7:confidentialityCode" minimumMultiplicity="1"
          maximumMultiplicity="1" isMandatory="true" datatype="CE">
            <example>
                <confidentialityCode code="N"
                  codeSystem="2.16.840.1.113883.5.25"/>
            </example>
            <vocabulary valueSet="BasicConfidentialityKind"/>
        </element>
        <include ref="1.2.3.4.10.2.1"/>
        <include ref="1.2.3.4.10.2.2"/>
        <include ref="1.2.3.4.10.2.3"/>
        <element name="hl7:component">
          <attribute name="typeCode" value="COMP"
            contextConductionInd="true" isOptional="true"/>
            <element name="hl7:structuredBody">
              <attribute name="classCode" value="DOCBODY"/>
              <attribute name="moodCode" value="EVN" isOptional="true"/>
                <!-- EKG Impression section, reuqired
                  (otherwise this report is useless) -->
                <element name="hl7:component" minimumMultiplicity="1"
                  maximumMultiplicity="1" isMandatory="true"
                  contains="1.2.3.4.10.3.1">
                    <attribute name="typeCode" value="COMP"
                      contextConductionInd="true" isOptional="true"/>
                </element>
            </element>
        </element>
    </element>
</template>
```

8. Appendix A – Glossary

Term	Definition
Attribute Value Domain	The set of possible values that can be assigned to an data element, i.e. a specific set of codes, valid numbers or ranges thereof etc.
Cardinalities	The cardinality indicator (0..1, 1..1, 1..*, etc.) specifies the allowable occurrences within a document instance.
Closed Template	A closed template defines what's relevant for that template and leaves no room for any other contents in the instance.
Co-occurrence constraint	Describes rules which govern what kind of data may or shall occur together in an XML instance; “conditional” is as a related term: if data X has value Y (condition) then data Z must be present/populated (co-occurrence)
Concept Domain	A named category of concepts, e.g. “person gender” for concepts representing the gender of a person
Conformance Verbs	SHALL, SHALL NOT, SHOULD, SHOULD NOT, MAY, MAY NOT
Content Consumer	Content Consumers use templates to receive the patterns and semantics presented to them by Content Creators.
Content Creator	Content Creators use templates to assert the patterns and semantics present in the data instances they create.
Content Validator	An assertion that confirms that the document does in fact meet all of the constraint requirements specified for the template.
(Type) Demotion	A term known in programming languages and V3: a datatype conversion can be a promotion or demotion, see also http://stackoverflow.com/questions/21307485/data-type-promotion-or-demotion-in-c
Dynamic Binding	A DYNAMIC binding implies a need for a look-up of the (most recent) artifact at runtime.
Open Template	An open template defines what's relevant for that template while allowing undefined contents in the instance.
Static Binding	A STATIC binding is a fixed binding at design time.
Template	A template is a set of conformance statements which further constrain an existing information model.
Template Body	A collection of constraints that ideally describes the structure and semantics of all instance elements.
Template Containment	Inclusion in a template definition of a component defined by another template.

Template Design	A template design can be defined in a number of ways. They can be created as human-written specifications or they can be developed using modeling / design tools which automate the process for greater precision and consistency. Templates can be published in human-readable or in machine processable forms. They can be managed in documents where they are defined or in repositories deployed through technology tools developed specifically for the management of templates.
Template Designer	a person who develops the set of conformance statements which make up a template.
Template Governance Group	A template governance group establishes the rules and best practices associated with their creation and management of templates.
Template Inclusion	"Inheritance" of the conformances defined in another template
Template Metadata	Version-related metadata that exists for each design of a template. Ex: Identifier, Name, Effective Date, Version Label, Expiration Date, Official Release Date, Status, and Additional metadata.
Template Registry	A template registry tracks the existence of templates and their designs from one or more template repositories. When template designs include interdependencies, it becomes necessary to stay informed of changes in the related templates. Template Registries may support functionality which permits a system to declare its interest in a template or set of templates so that it may be informed when changes occur. Once a system has subscribed, the Template Registry can publish a change, and the needed notification happens automatically.
Template Repository	A template repository houses the definition of template designs.
Template Types	Document level, Header level, Section level, Entry level, etc.
Vocabulary Binding	There are three practically used types of vocabulary bindings. • A coded element can be bound to a specific code from a specific code system and the population of the element can be considered as a “constant”; also a set of specific codes are possible. • A coded element can be bound to a specific value set by value set id (or name). A coded element can be bound to a certain concept domain by domain id or name.

9. Appendix B – Acronyms and Abbreviations

CDA	Clinical Document Architecture
DAM	Domain Analysis Model
DSTU	Draft Standard for Trial Use
H&P	History and Physical
HIT	healthcare information technology
HL7	Health Level Seven
HTML	Hypertext Markup Language
IG	implementation guide
IHE	Integrating the Healthcare Enterprise
IHTSDO	International Health Terminology Standard Development Organisation
LOINC	Logical Observation Identifiers Names and Codes
PDF	portable document format
RIM	Reference Information Model
SDWG	Structured Documents Working Group
SDO	Standards Development Organization
SNOMED CT	Systemized Nomenclature for Medicine – Clinical Terms
UCUM	Unified Code for Units of Measure
URL	Uniform Resource Locator
XPath	XML Path Language

10. Appendix E – References

Extensible Markup Language (XML) 1.0 (Fifth Edition), http://www.w3c.org/TR/2008/REC-xml-20081126/

HL7 Clinical Document Architecture (CDA Release 2). http://www.hl7.org/implement/standards/cda.cfm

HL7 Version 3 Interoperability Standards, Normative Edition 2010. http://www.hl7.org/memonly/downloads/v3edition.cfm - V32010

[ccdar1] HL7 Implementation Guide for CDA® Release 2: IHE Health Story Consolidation, DSTU Release1.1 (US Realm) Draft Standard for Trial Use, January 2013

[ccdar2] HL7 Implementation Guide for CDA® Release 2: Consolidated CDA Templates for Clinical Notes (US Realm) Draft Standard for Trial Use Release 2, HL7 Draft Standard for Trial Use (DSTU) Ballot, September 2013

[v2xml] XML Encoding Rules (HL7 Version 2: XML Encoding Syntax, Release 1) http://www.hl7.org/implement/standards/product_brief.cfm?product_id=83

[fhir] FHIR® – Fast Health Interoperable Resources, http://hl7.org/fhir

[isosch] ISO/IEC 19757-3:2006 Information technology -- Document Schema Definition Language (DSDL) -- Part 3: Rule-based validation – schematron, http://www.schematron.com

[isooid] ISO/DTS 13582:2012 Health Informatics — Sharing of OID registry information

[mif] HL7 Version 3 Standard: Model Interchange Format, Release 1, http://www.hl7.org/implement/standards/product_brief.cfm?product_id=101

[hl7v3pfg] HL7, Version 3 Publishing Facilitator's Guide. http://www.hl7.org/v3ballot/html/help/pfg/pfg.htm

[rfc2119] RFC 2119 https://www.ietf.org/rfc/rfc2119.txt