Leading healthcare terminology, worldwide

SNOMED CT Template Syntax Specification

Version 1.1

Publication date: 2020-09-08

Web version link: http://snomed.org/sts

SNOMED CT document library: http://snomed.org/doc

This PDF document was generated from the web version on the publication date shown above. Any changes made to the web pages since that date will not appear in the PDF. See the web version of this document for recent updates.

© Copyright 2020 International Health Terminology Standards Development Organisation

Leading healthcare
terminology, worldwide

© Copyright 2020 International Health Terminology Standards Development Organisation

SNOMED International

Table of Contents

1. Introduction....2
2. Use Cases....4
   - 2.1. Authoring of Pre-coordinated Concepts....4
   - 2.2. Defining Post-coordinated Clinical Meanings....5
3. Requirements....8
   - 3.1. General SNOMED CT Language Requirements....8
   - 3.2. Template Design Requirements....8
   - 3.3. Template Processing Requirements....9
4. Logical Model....10
   - 4.1 UML Class Diagram....12
5. Syntax Specification....14
   - 5.1. Normative Specification....14
   - 5.2. Informative Comments....16
6. SNOMED CT Language Templates....19
   - 6.1. Expression Template Language....19
7. Processing Expression Templates....23
   - 7.1. Preparing Input Data....23
   - 7.2. Template Processing....37
   - 7.3. Post-processing Validation....39
8. Expression Template Examples....41
   - 8.1. Simple Replacement Slots....41
   - 8.2. Typed Replacement Slots....43
   - 8.3. Constrained Replacement Slots....46
   - 8.4. Named Replacement Slots....49
   - 8.5. Information Slots....50
   - 8.6. Advanced Expression Templates....52

© Copyright 2020 International Health Terminology Standards Development Organisation

Leading healthcare terminology, worldwide

The *SNOMED template syntax specification* defines the formal rules for representing *slots* in SNOMED CT expressions, expression constraints or queries. A *slot* either provides information as to how the template should be processed, or represents a placeholder for a value that is not known at the time of authoring. These placeholders can be completed at a later time using data recorded elsewhere (such as in an information model or entered into a data entry form).

The *template syntax* may be used in conjunction with any of the computable SNOMED CT languages to represent templates of various kinds. For example, using the template syntax with *compositional grammar* enables the representation of *expression templates*, while using the template syntax with the *expression constraint language* enables the representation of *expression constraint templates*.

SNOMED templates can be used for a number of purposes, including to define reusable patterns for authoring and validating precoordinated concept definitions and postcoordinated expressions.

Web browsable version: http://snomed.org/sts

SNOMED CT Document Library: http://snomed.org/doc

© Copyright 2020 International Health Terminology Standards Development Organisation, all rights reserved.

This document is a publication of International Health Terminology Standards Development Organisation, trading as SNOMED International. SNOMED International owns and maintains SNOMED CT®.

Any modification of this document (including without limitation the removal or modification of this notice) is prohibited without the express written permission of SNOMED International. This document may be subject to updates. Always use the latest version of this document published by SNOMED International. This can be viewed online and downloaded by following the links on the front page or cover of this document.

SNOMED®, SNOMED CT® and IHTSDO® are registered trademarks of International Health Terminology Standards Development Organisation. SNOMED CT® licensing information is available at http://snomed.org/licensing. For more information about SNOMED International and SNOMED International Membership, please refer to http://www.snomed.org or contact us at info@snomed.org.

1. Introduction

Background

SNOMED CT is a clinical terminology with global scope covering a wide range of clinical specialties and requirements. The use of SNOMED CT expressions in electronic health records (EHRs) provides a standardized way to represent clinical meanings captured by clinicians and enables the automatic interpretation of these meanings. SNOMED CT expression constraints provide a computable rule that can be used to define a bounded set of clinical meanings for the purpose of constraining the contents of a data element in an electronic health record (EHR), intensionally defining a concept-based reference set, querying SNOMED CT content in a machine processable way, or restricting the range of an attribute defined in the SNOMED CT concept model.

In some situations, however, the specific concepts or values are not known at the time of authoring. In these cases, one or more *slots* may be used within an expression or expression constraint to create an *expression template* or *expression constraint template* (respectively). S*lots* provide a placeholder in the expression or expression constraint, whose specific value can be completed at a subsequent time using a concept recorded within an information model, entered into a data entry form, or sourced by some other means. When each slot in the template has been replaced with a specific value, the result should be a syntactically correct expression or expression constraint. SNOMED CT templates can be used for a number of purposes, including to define reusable patterns for authoring and validating precoordinated concept definitions and postcoordinated expressions.

History

The SNOMED CT Template Syntax (v1.0) was first published in July 2017. In 2020, the syntax was updated (v1.1) to support boolean attribute values.

Purpose

The purpose of this document is to define and describe a consistent mechanism for SNOMED CT templates, which can be used to convert any computable SNOMED CT language into a SNOMED CT template language. For example, by using the syntax provided in this document, SNOMED CT compositional grammar can be used as an expression template language, and the SNOMED CT expression constraint Language can be used as an expression constraint template language. This guide also provides examples and guidance to assist in the processing of templates.

Scope

This document presents the specification of a SNOMED CT template syntax, which can be used together with other formal SNOMED CT languages to develop SNOMED CT Templates. The SNOMED CT Template Syntax is part of a consistent set of computer processable language syntaxes designed to support a variety of use cases involving the use of SNOMED CT. Other computable SNOMED CT languages that are either complete or under development include:

- Compositional Grammar: designed to represent SNOMED CT expressions;
- **Expression Constraint Language**: designed to represent a bounded set of clinical meanings represented using SNOMED CT; and

This document provides a specification, examples and general guidance to assist in the representation and processing of SNOMED CT templates.

However, this document does not include a full description of how to implement a template parser or interpreter. It does not describe how to transform a template into other languages (such as OWL, SPARQL or SQL), or how to determine whether two templates are equivalent. It also does not describe how to implement a terminology server or an EHR which uses templates to constrain or query its content.

© Copyright 2020 International Health Terminology Standards Development Organisation

Audience

The target audiences of this document include:

- SNOMED International National Release Centres;
- SNOMED CT designers and developers, including designers and developers of EHR systems, information models, data entry interfaces, storage systems, decision support systems, retrieval and analysis systems, communication standards and terminology services;
- SNOMED CT terminology developers, including concept model designers, content authors, map developers, subset and constraint developers and release process managers.

It should be noted that this document contains both technical and non-technical content. In particular, the detailed logical model and formal syntax is specifically focussed at more technical readers. Less technical readers are encouraged to read the introductory material and examples.

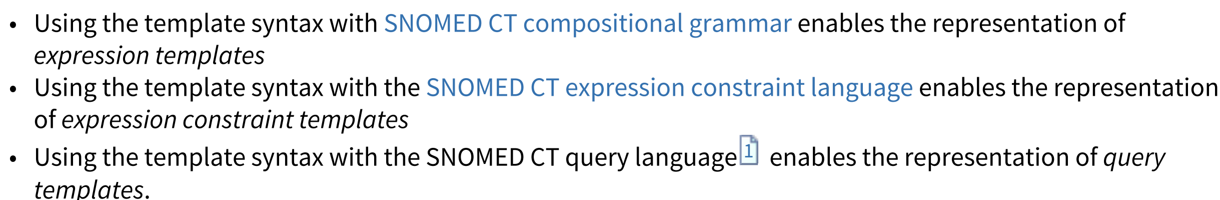
Document Overview

This document defines the SNOMED CT template syntax and describes how and where it may be implemented. Chapter 2 begins by describing some key use cases in which SNOMED CT templates can be used. Chapter 3 then describes the requirements used to guide the definition of this syntax. In Chapter 4, the logical model of the template syntax is presented, while Chapter 5 defines the ABNF serialisation of the logical model. Chapter 6 explains how to apply the SNOMED CT template syntax to other computable languages, and provides the syntax for the expression template language (ETL). In chapter 7, we explain the steps involved in processing an expression template to create a set of populated expressions. And finally, in chapter 8, we present some examples of SNOMED CT expression templates that conform to the syntaxes defined in Chapters 5 and 6.

© Copyright 2020 International Health Terminology Standards Development Organisation

2. Use Cases

When values within a SNOMED CT expression, expression constraint or query are unknown at the time of authoring, a SNOMED CT template may be used. The SNOMED CT template syntax can be used in conjunction with the SNOMED CT computable languages to create SNOMED CT template languages for this purpose. For example:



- Using the template syntax with SNOMED CT compositional grammar enables the representation of *expression templates*
- Using the template syntax with the SNOMED CT expression constraint language enables the representation of *expression constraint templates*
- Using the template syntax with the SNOMED CT query language[1] enables the representation of *query templates*.

This version of the SNOMED CT template syntax specifically focuses on supporting two important use cases in which the SNOMED CT template syntax is needed. In particular to support:

- Authoring of precoordinated concepts
- Defining consistent postcoordinated expressions

These use cases specifically require the use of slots within SNOMED CT Compositional Grammar. Therefore, this version of the specification focuses primarily on supporting the use of the template syntax within the context of the Expression Template Language. Future versions of this specification will explore other use cases, and additional template languages (e.g. the Expression Constraint Template Language).

---

[1] Please note that the SNOMED CT query language is not yet available.

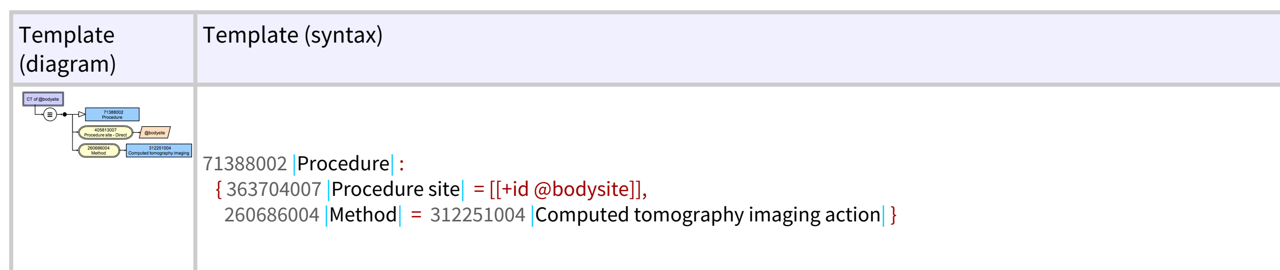
2.1. Authoring of Pre-coordinated Concepts

Expression templates can be used to author precoordinated clinical meanings by providing reusable patterns for the definition of similar concepts. Expression templates can define the set of attributes and attribute values that may be used to define concepts in a specific hierarchy or subhierarchy, thereby assisting SNOMED CT authors to create concepts in a consistent way. They can also be used together with an input file to either create or update batches of concepts very efficiently. Using expression templates to author SNOMED CT concepts can help to improve the quality of the terminology, by controlling the consistency of the definitions, and by validating concept definition against appropriate templates. Similarly, they can also be used as a metric to measure the quality of given subhierarchy.

In summary, expression templates can be used by terminology authors to:

- Create a single concept, or a batch of concepts;
- Update the definition of a concept (or batch of concepts) to conform to a particular pattern;
- Validate the definition of concept or a set of concepts;
- Define quality metrics for a particular subhierarchy.

The table below shows an expression template that may be used to define computed tomography (CT) concepts of a particular body site (e.g. CT of right arm). The 'bodysite' slot represents the placeholder, which is subsequently replaced with a specific body site concept to form a complete concept definition.



Template (diagram)	Template (syntax)
	71388002 |Procedure| : { 363704007 |Procedure site| = [[+id @bodysite]], 260686004 |Method| = 312251004 |Computed tomography imaging action| }

© Copyright 2020 International Health Terminology Standards Development Organisation

Table 2.1-1: An example expression template

The following table shows the expressions that result when processing the template using a specific slot value:

bodysite	Resulting Expression
48979004 |Structure of left lower leg (body structure)|	71388002 |Procedure| : { 363704007 |Procedure site| = 48979004 |Structure of left lower leg| , 260686004 |Method| = 312251004 |Computed tomography imaging action| }
368209003 |Right upper arm structure (body structure)|	71388002 |Procedure| : { 363704007 |Procedure site| = 368209003 |Right upper arm structure| , 260686004 |Method| = 312251004 |Computed tomography imaging action| }

Table 2.1-2: Expressions that result from populating the example expression template

2.2. Defining Post-coordinated Clinical Meanings

Expression templates can be used to enable the easy and consistent authoring of postcoordinated expressions. They are particularly useful in situations where a specific pattern is needed to support the entry, storage or retrieval of SNOMED CT expressions. Therefore, expression templates are useful for:

- Creating batches of postcoordinated expressions with a consistent structure; and
- Composing postcoordinated expressions from data entered into a user interface.

These uses are explained in more detail in the following sections.

Batch Authoring of Expressions

Expression templates may be applied to ensure a consistent structure is used by a set of authored expressions. For example, there may be a preference to always represent allergies using an expression in which the allergen is explicitly defined as the value of the |Causative agent| attribute. Given an appropriate expression template and a predefined list of allergen substances, a set of postcoordinated expressions can automatically be batch authored (as shown below).

Expression Template

```
419199007 |allergy to substance| : 246075003 |Causative agent| = [[+id @Substance]]
```

List of Substances

```
256259004 |Pollen|
89811004 |Gluten|
47703008 |Lactose|
13577000 |Nut|
33396006 |Nickel|
```

Resulting Expressions

```
419199007 |Allergy to substance| : 246075003 |Causative agent| = 256259004 |Pollen|
419199007 |Allergy to substance| : 246075003 |Causative agent| = 89811004 |Gluten|
419199007 |Allergy to substance| : 246075003 |Causative agent| = 47703008 |Lactose|
419199007 |Allergy to substance| : 246075003 |Causative agent| = 13577000 |Nut|
419199007 |Allergy to substance| : 246075003 |Causative agent| = 33396006 |Nickel|
```

Composing Expressions from a User Interface

Expression Templates are also useful for creating postcoordinated expressions from data entered in a user interface. For example, a radiology user interface may use two separate fields to capture the imaging procedure and the body site to which the procedure was applied. An expression template can then be applied to combine the data entered into these two fields into a single postcoordinated expression (see diagram below).

© Copyright 2020 International Health Terminology Standards Development Organisation

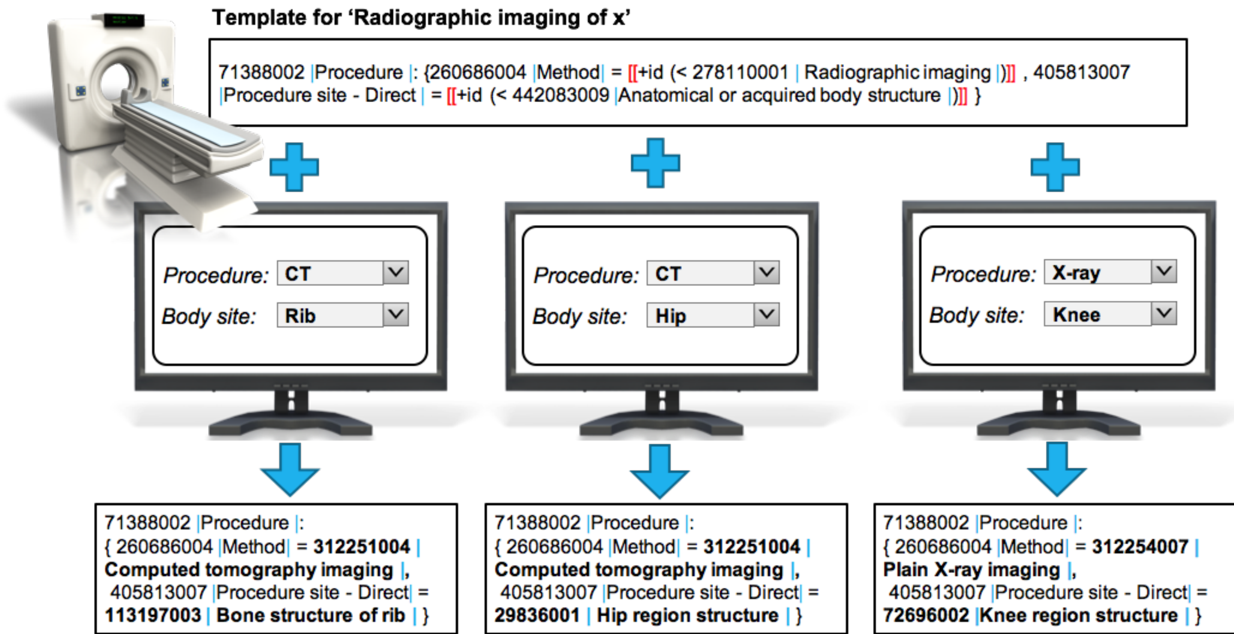


Figure 2.2-1: Using an expression template with data entered into a user interface

© Copyright 2020 International Health Terminology Standards Development Organisation

3. Requirements

In this chapter, we state the requirements of the SNOMED CT Template Syntax. These requirements are grouped into general SNOMED CT language requirements (which are shared by all SNOMED CT computable languages), template design requirements and template processing requirements.

3.1. General SNOMED CT Language Requirements

The general SNOMED CT language requirements include:

Requirement G.1: Backward compatibility

The language must be backwardly compatible with any version of the language that has previously been adopted as an IHTSDO standard. Please note that this requirement is not applicable to this version of the Template Syntax, as no previous version has been published as an IHTSDO standard.

Requirement G.2: Consistency

Each logical feature of the language should have a single, consistent meaning across all the languages in the SNOMED CT family of languages. Each logical feature should also have a consistent set of syntax representations.

Requirement G.3: Sufficient and necessary

Each language must be sufficiently expressive to meet the requirements of the use cases for which it was designed. However, functionality without a corresponding use case will not be included, as this increases the complexity of implementation unnecessarily.

Requirement G.4: Machine processability

In order to facilitate the easy adoption by technical audiences, instances of each language must be able to be parsed into a logical representation using a machine processable syntax specification. This requirement will be met by defining the language syntax in ABNF.

Requirement G.5: Human readability

Non-technical stakeholders require that the language is as human readable as possible, while still meeting the other requirements. This is essential for both the clinical validation of language instances, as well as for education and training.

3.2. Template Design Requirements

The template design requirements include:

Requirement D.1: Placeholders for values

Templates should allow for placeholders (i.e. slots) to be specified, wherever a concept, expression, token or attribute value can be used.

Requirement D.2 Slot names

Templates should allow each slot to be assigned a name, enabling the slot to be referenced outside the slot (e.g. for specifying input data).

Requirement D.3 Slot types

Template slots should be able to be constrained to only permit a specific type of value (e.g. only allowing precoordinated concepts to replace a slot).

Requirement D.4: Template value constraints

Templates slots should be able to be constrained to only permit values from a specific value set (e.g. only allowing expressions that satisfy a given expression constraint).

© Copyright 2020 International Health Terminology Standards Development Organisation

Requirement D.5: Repeatability of template components

Templates should support a way of indicating how many times each focus concept, relationship group and attribute value pair can be repeated when the template is populated.

3.3. Template Processing Requirements

The template processing requirements include:

Requirement P.1: Processable input data

Input data must be available, which can be automatically processed with the template to populate the slots with values. Input data must be clear (either explicitly or implicitly) as to which slot it is intended to populate, and whether multiple values are intended to be used within the same relationship group, the same expression (but different relationship groups), or different expressions.

Requirement P.2: Post-processing validity of constraints

After template processing, the resulting language instance (e.g. expression) must be valid against the cardinality, slot type and value constraints that were defined in the template.

Requirement P.3: Post-processing syntactic validity

After template processing, the resulting language instance (e.g. expression) must conform to the ABNF syntax of the associated base language (e.g. SNOMED CT compositional grammar).

Requirement P.4: Post-processing concept model validity

After template processing, the resulting language instance (e.g. expression) should be valid according to the concept model rules defined by the relevant MRCM.

Requirement P.5: Post-processing use case validity

After template processing, the resulting language instance (e.g. expression) should conform to all additional rules that are imposed by the relevant use case. For example, if an expression template is being used to author precoordinated concepts, then the resulting expression must not use nested values (to enable representation in RF2).

© Copyright 2020 International Health Terminology Standards Development Organisation

4. Logical Model

A SNOMED CT template is a SNOMED CT expression, expression constraint, or query that contains one or more *slots.* Each template slot either provides information as to how the template should be processed (i.e. an information slot), or serves as a placeholder for a specific value that may be completed at a subsequent time. Each replacement slot may have a replacement type (e.g. concept, expression, token, string, integer, decimal or boolean), a slot name, and a replacement constraint. Depending on the replacement type, the replacement constraint may either be an expression constraint, a value list constraint or a range constraint. Each information slot may have a cardinality and a slot name.

The SNOMED CT template syntax defines the syntax used for these slots, irrespective of the computable language in which they are embedded. Figure 4-1 below illustrates the abstract model of the SNOMED CT template syntax. Please note that no specific semantics should be attributed to each arrow in this diagram.

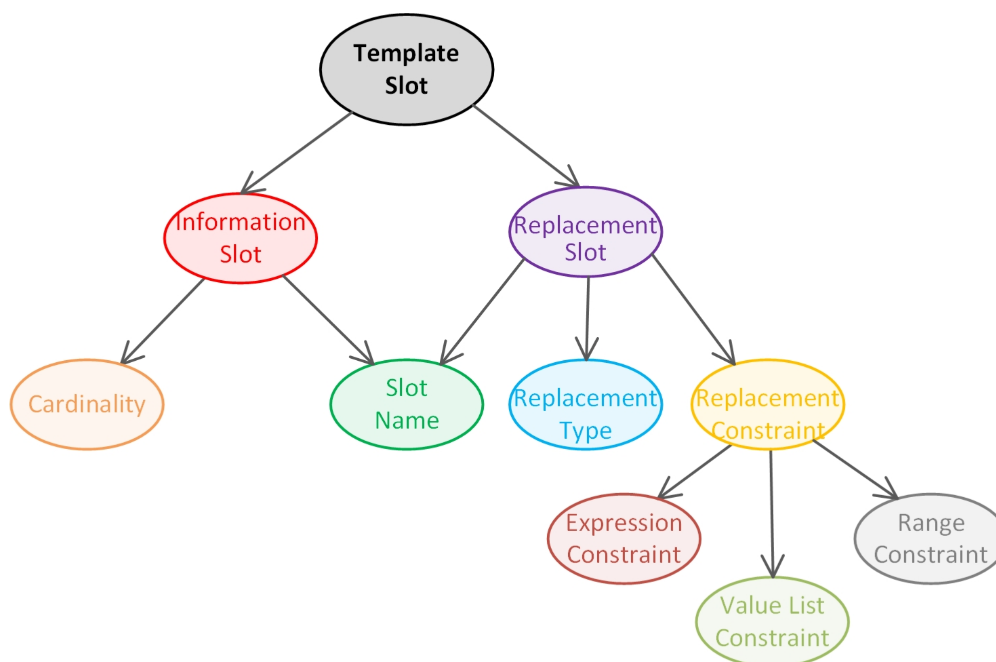


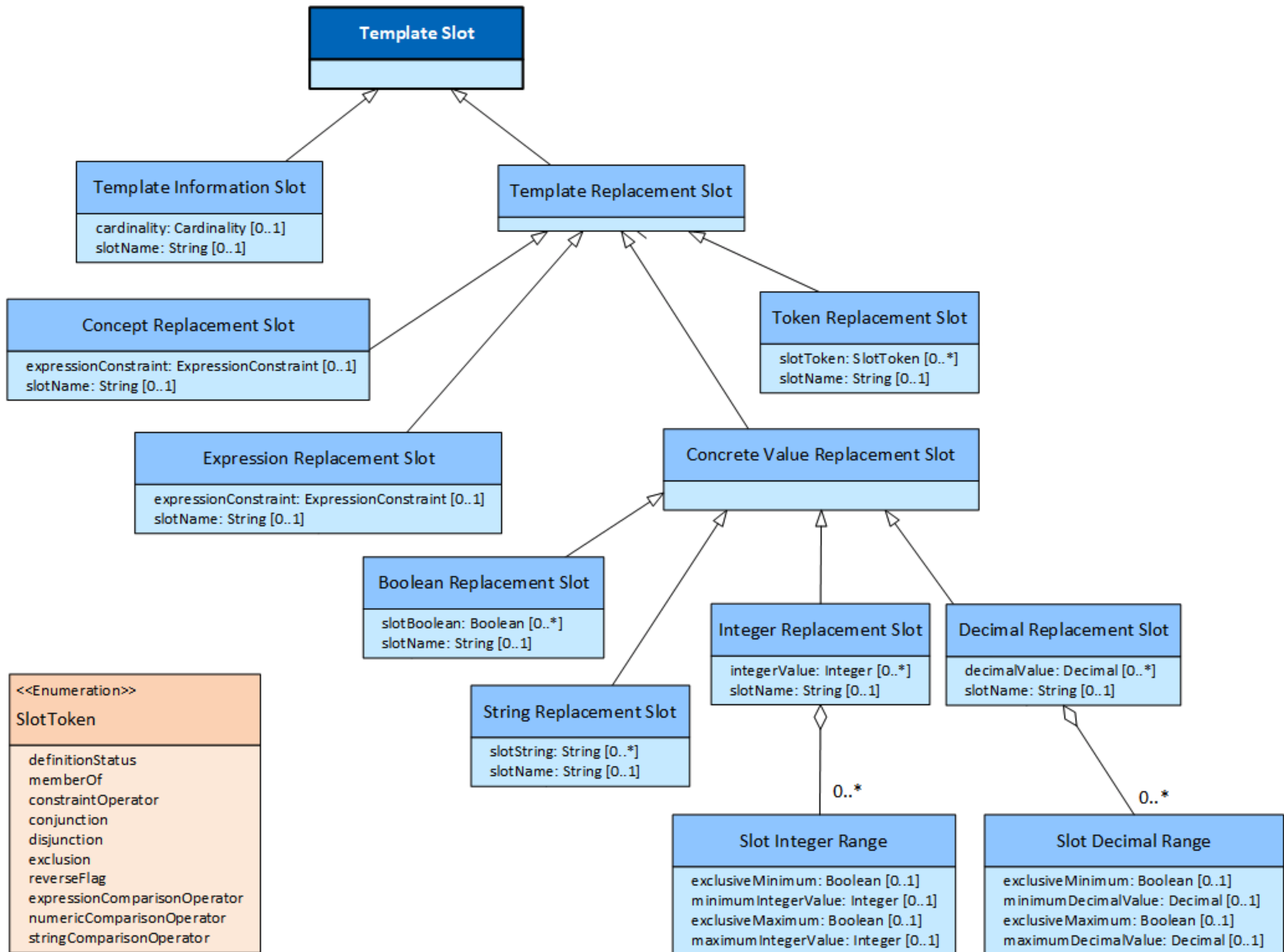
Figure 4-1: Abstract model of SNOMED CT Template Syntax

© Copyright 2020 International Health Terminology Standards Development Organisation

4.1 UML Class Diagram

The figure below provides a non-normative representation of the logical model of the SNOMED CT Template Syntax using a UML class diagram.

Please note that each of the classes in this diagram corresponds to a rule in the syntax specification defined in 5. Syntax Specification. For a short description of each of these, please refer to 5.2. Informative Comments.



© Copyright 2020 International Health Terminology Standards Development Organisation

Figure 4.1-1: UML class diagram of template syntax

© Copyright 2020 International Health Terminology Standards Development Organisation

5. Syntax Specification

The following sections describe the rules used by the Template Syntax. This syntax is a serialised representation of the logical model presented in the previous chapter, and is considered to be the normative syntax for interoperability purposes.

5.1. Normative Specification

The following ABNF definition specifies the SNOMED CT template syntax. This syntax incorporates the Expression Constraint Language version 1.3 (ECL v1.3), with some adaptations to support slot references.

Please note that some of the rules referenced below are defined elsewhere. In particular, **definitionStatus** is define d in SNOMED CT compositional grammar. This rule definition must therefore be combined with the syntax below before it can be used.

```
; Template Syntax v1.1
templateSlot = templateReplacementSlot / templateInformationSlot
templateReplacementSlot = conceptReplacementSlot / expressionReplacementSlot / tokenReplacementSlot /
concreteValueReplacementSlot
conceptReplacementSlot = "[[" ws "+" ws conceptReplacement [slotName ws] "]]"
expressionReplacementSlot = "[[" ws "+" ws expressionReplacement [slotName ws] "]]"
tokenReplacementSlot = "[[" ws "+" ws tokenReplacement [slotName ws] "]]"
concreteValueReplacementSlot = "[[" ws "+" ws concreteValueReplacement [slotName ws] "]]"
conceptReplacement = "id" ws [ "(" ws slotExpressionConstraint ws ")" ws]
expressionReplacement = ["scg" ws] ["(" ws slotExpressionConstraint ws ")" ws]
tokenReplacement = "tok" ws [ "(" ws slotTokenSet ws ")" ws]
concreteValueReplacement = stringReplacement / integerReplacement / decimalReplacement /
booleanReplacement
stringReplacement = "str" ws [ "(" ws slotStringSet ws ")" ws]
integerReplacement  = "int" ws [ "(" ws slotIntegerSet ws ")" ws]
decimalReplacement = "dec" ws [ "(" ws slotDecimalSet ws ")" ws]
booleanReplacement = "bool" ws [ "(" ws slotBooleanSet ws ")" ws]
slotTokenSet = slotToken *(mws slotToken)
slotStringSet = slotString *(mws slotString)
slotIntegerSet = ( slotIntegerValue / slotIntegerRange) *(mws (slotIntegerValue / slotIntegerRange))
slotDecimalSet = ( slotDecimalValue / slotDecimalRange) *(mws (slotDecimalValue / slotDecimalRange))
slotBooleanSet = slotBooleanValue *(mws slotBooleanValue)
slotIntegerRange = ( slotIntegerMinimum to [ slotIntegerMaximum ] ) / ( to slotIntegerMaximum )
slotIntegerMinimum = [ exclusiveMinimum ] slotIntegerValue
slotIntegerMaximum = [ exclusiveMaximum ] slotIntegerValue
slotIntegerValue = "#" digitNonZero *digit / "#" zero
slotDecimalRange = ( slotDecimalMinimum to [ slotDecimalMaximum ] ) / ( to slotDecimalMaximum )
slotDecimalMinimum = [ exclusiveMinimum ] slotDecimalValue
slotDecimalMaximum = [ exclusiveMaximum ] slotDecimalValue
slotDecimalValue = "#" integerValue "." 1*digit
exclusiveMinimum = ">"
exclusiveMaximum = "<"
slotBooleanValue = true / false
true = ("t"/"T") ("r"/"R") ("u"/"U") ("e"/"E")
false = ("f"/"F") ("a"/"A") ("l"/"L") ("s"/"S") ("e"/"E")
slotName = "@" (nonQuoteStringValue / slotString)
slotToken = definitionStatus / memberOf / constraintOperator / conjunction / disjunction / exclusion / reverseFlag /
expressionComparisonOperator / numericComparisonOperator / stringComparisonOperator /
booleanComparisonOperator
slotString = QM slotStringValue QM
```

© Copyright 2020 International Health Terminology Standards Development Organisation

nonQuoteStringValue = *(%x21 / %x23-26 / %x2A-3F / %x41-5A / %x5C / %x5E-7E) ; string with no ws, quotes, at, square brackets or round brackets

templateInformationSlot = "[[" ws slotInformation ws "]]"

slotInformation = [cardinality ws] [slotName ws]

; Expression Constraint Language v1.4

slotExpressionConstraint = ws (slotRefinedExpressionConstraint / slotCompoundExpressionConstraint / slotDottedExpressionConstraint / slotSubExpressionConstraint) ws

slotRefinedExpressionConstraint = slotSubExpressionConstraint ws ":" ws slotEclRefinement

slotCompoundExpressionConstraint = slotConjunctionExpressionConstraint / slotDisjunctionExpressionConstraint / slotExclusionExpressionConstraint

slotConjunctionExpressionConstraint = slotSubExpressionConstraint 1*(ws conjunction ws slotSubExpressionConstraint)

slotDisjunctionExpressionConstraint = slotSubExpressionConstraint 1*(ws disjunction ws slotSubExpressionConstraint)

slotExclusionExpressionConstraint = slotSubExpressionConstraint ws exclusion ws slotSubExpressionConstraint

slotDottedExpressionConstraint = slotSubExpressionConstraint 1*(ws slotDottedExpressionAttribute)

slotDottedExpressionAttribute = dot ws slotEclAttributeName

slotSubExpressionConstraint = [constraintOperator ws] [memberOf ws] (slotEclFocusConcept / "(" ws slotExpressionConstraint ws ")")

slotEclFocusConcept = slotEclConceptReference / wildCard

dot = "."

memberOf = "^"

slotEclConceptReference = conceptId [ws "|" ws term ws "|"]

conceptId = sctId

term = 1*nonwsNonPipe *(1*SP 1*nonwsNonPipe)

wildcard = "*"

constraintOperator = childOf / descendantOrSelfOf / descendantOf / parentOf / ancestorOrSelfOf / ancestorOf

descendantOf = "<"

descendantOrSelfOf = "<<"

childOf = "<!"

ancestorOf = ">"

ancestorOrSelfOf = ">>"

parentOf = ">!"

conjunction = (("a"/"A") ("n"/"N") ("d"/"D") mws) / ","

disjunction = ("o"/"O") ("r"/"R") mws

exclusion = ("m"/"M") ("i"/"I") ("n"/"N") ("u"/"U") ("s"/"S") mws

slotEclRefinement = slotSubRefinement ws [slotConjunctionRefinementSet / slotDisjunctionRefinementSet]

slotConjunctionRefinementSet = 1*(ws conjunction ws slotSubRefinement)

slotDisjunctionRefinementSet = 1*(ws disjunction ws slotSubRefinement)

slotSubRefinement = slotEclAttributeSet / slotEclAttributeGroup / "(" ws slotEclRefinement ws ")"

slotEclAttributeSet = slotSubAttributeSet ws [slotConjunctionAttributeSet / slotDisjunctionAttributeSet]

slotConjunctionAttributeSet = 1*(ws conjunction ws slotSubAttributeSet)

slotDisjunctionAttributeSet = 1*(ws disjunction ws slotSubAttributeSet)

slotSubAttributeSet = slotEclAttribute / "(" ws slotEclAttributeSet ws ")"

slotEclAttributeGroup = ["[" cardinality "]" ws] "{" ws slotEclAttributeSet ws "}"

slotEclAttribute = ["[" cardinality "]" ws] [reverseFlag ws] slotEclAttributeName ws (expressionComparisonOperator ws slotSubExpressionConstraint / numericComparisonOperator ws slotNumericValue / stringComparisonOperator ws slotStringValue / booleanComparisonOperator ws booleanValue)

cardinality = minValue to maxValue

minValue = nonNegativeIntegerValue

to = ".."

maxValue = nonNegativeIntegerValue / many

many = "*"

reverseFlag = "R"

slotEclAttributeName = slotSubExpressionConstraint

```
expressionComparisonOperator = "=" / "!="
numericComparisonOperator = "=" / "!=" / "<=" / "<" / ">=" / ">"
stringComparisonOperator = "=" / "!="
booleanComparisonOperator = "=" / "!="
slotNumericValue = "#" ["-"/"+"] (slotDecimalValue / slotIntegerValue)
slotStringValue = QM 1*(anyNonEscapedChar / escapedChar) QM
integerValue = digitNonZero *digit / zero
decimalValue = integerValue "." 1*digit
booleanValue = true / false
; true = ("t"/"T") ("r"/"R") ("u"/"U") ("e"/"E")
; false = ("f"/"F") ("a"/"A") ("l"/"L") ("s"/"S") ("e"/"E")
nonNegativeIntegerValue = (digitNonZero *digit ) / zero
sctId = digitNonZero 5*17( digit )
ws = *( SP / HTAB / CR / LF / comment ) ; optional white space
mws = 1*( SP / HTAB / CR / LF / comment ) ; mandatory white space
comment = "/*" *(nonStarChar / starWithNonFSlash) "*/"
nonStarChar = SP / HTAB / CR / LF / %x21-29 / %x2B-7E /UTF8-2 / UTF8-3 / UTF8-4
starWithNonFSlash = %x2A nonFSlash
nonFSlash = SP / HTAB / CR / LF / %x21-2E / %x30-7E /UTF8-2 / UTF8-3 / UTF8-4
SP = %x20 ; space
HTAB = %x09 ; tab
CR = %x0D ; carriage return
LF = %x0A ; line feed
QM = %x22 ; quotation mark
BS = %x5C ; back slash
digit = %x30-39
zero = %x30
digitNonZero = %x31-39
nonwsNonPipe = %x21-7B / %x7D-7E / UTF8-2 / UTF8-3 / UTF8-4
anyNonEscapedChar = SP / HTAB / CR / LF / %x20-21 / %x23-5B / %x5D-7E / UTF8-2 / UTF8-3 / UTF8-4
escapedChar = BS QM / BS BS
UTF8-2 = %xC2-DF UTF8-tail
UTF8-3 = %xE0 %xA0-BF UTF8-tail / %xE1-EC 2( UTF8-tail ) / %xED %x80-9F UTF8-tail / %xEE-EF 2( UTF8-tail )
UTF8-4 = %xF0 %x90-BF 2( UTF8-tail ) / %xF1-F3 3( UTF8-tail ) / %xF4 %x80-8F 2( UTF8-tail )
UTF8-tail = %x80-BF
; Additional rules from Compositional Grammar
definitionStatus = equivalentTo / subtypeOf
equivalentTo = "==="
subtypeOf = "<<<"
```

5.2. Informative Comments

This section provides a brief description of each rule listed above in the normative specification.

templateSlot = templateReplacementSlot / templateInformationSlot	
	A template slot is part of a template that is either replaced with a value (i.e. a templateRelacementSlot), or removed after the information it contains is interpreted (i.e. a templateInformationSlot).
templateReplacementSlot = conceptReplacementSlot / expressionReplacementSlot / tokenReplacementSlot / concreteValueReplacementSlot	
	A template replacement slot is a placeholder whose value can be completed at a subsequent time. A template replacement slot may be either a concept replacement slot, an expression replacement slot, a token replacement slot or a concrete value replacement slot.
conceptReplacementSlot = "[[" ws "+" ws conceptReplacement [slotName ws] "]]"	

	A concept replacement slot starts and ends with double square brackets (i.e. "[[...]]"). The first non-whitespace character inside the brackets must always be a plus sign (i.e. "+"), followed by the concept replacement rule and an optional slot name.
expressionReplacementSlot = "[[" ws "+" ws expressionReplacement [slotName ws] "]]"	
	An expression replacement slot starts and ends with double square brackets (i.e. "[[...]]"). The first non-whitespace character inside the brackets must always be a plus sign (i.e. "+"), followed by the expression replacement rule and an optional slot name.
tokenReplacementSlot = "[[" ws "+" ws tokenReplacement [slotName ws] "]]"	
	A token replacement slot starts and ends with double square brackets (i.e. "[[...]]"). The first non-whitespace character inside the brackets must always be a plus sign (i.e. "+"), followed by the token replacement rule and an optional slot name.
concreteValueReplacementSlot = "[[" ws "+" ws concreteValueReplacement [slotName ws] "]]"	
	A concrete value replacement slot starts and ends with double square brackets (i.e. "[[...]]"). The first non-whitespace character inside the brackets must always be a plus sign (i.e. "+"), followed by the concept replacement rule and an optional slot name.
conceptReplacement = "id" ws ["(" ws slotExpressionConstraint ws ")" ws]	
	A concept replacement starts with the text "id", and then optionally includes an expression constraint (in round brackets) that restricts the valid values that may replace the slot.
expressionReplacement = ["scg" ws] ["(" ws slotExpressionConstraint ws ")" ws]	
	An expression replacement starts (optionally) with the text "scg", and then optionally includes an expression constraint (in round brackets) to restrict the valid values that may replace the slot. Note that "scg" is the default type of replacement slot.
tokenReplacement = "tok" ws ["(" ws slotTokenSet ws ")" ws]	
	A token replacement starts with the text "tok", and then optionally includes a token set (in round brackets) to restrict the valid values that may replace the slot.
concreteValueReplacement = stringReplacement / integerReplacement / decimalReplacement / booleanReplacement	
	A concrete value replacement may either be a string replacement, an integer replacement, a decimal replacement, or a boolean replacement.
stringReplacement = "str" ws ["(" ws slotStringSet ws ")" ws]	
	A string replacement starts with the text "str", and then optionally includes a string set (in round brackets) to restrict the valid values that may replace the slot.
integerReplacement = "int" ws ["(" ws slotIntegerSet ws ")" ws]	
	An integer replacement starts with the text "int", and then optionally includes a integer set (in round brackets) to restrict the valid values that may replace the slot.
decimalReplacement = "dec" ws ["(" ws slotDecimalSet ws ")" ws]	
	A decimal replacement starts with the text "dec", and then optionally includes a decimal set (in round brackets) to restrict the valid values that may replace the slot.
booleanReplacement = "bool" ws ["(" ws slotBooleanSet ws ")" ws]	
	A boolean replacement starts with the text "bool", and then optionally includes a boolean set (in round brackets) to list the valid values that may replace the slot.
slotTokenSet = slotToken *(mws slotToken)	
	A slotTokenSet consists of one or more slotTokens separated by mandatory white space (i.e. mws).
slotStringSet = slotString *(mws slotString)	
	a slotStringSet consists of one or more slotStrings separated by mandatory white space (i.e. mws).
slotIntegerSet = ("#" integerValue / slotIntegerRange) *(mws ("#" integerValue / slotIntegerRange))	
	A slotIntegerSet consists of one or more slotIntegers separated by mandatory white space (i.e. mws).
slotDecimalSet = ("#" decimalValue / slotDecimalRange) *(mws ("#" decimalValue / slotDecimalRange))	
	A slotDecimalSet consists of one or more slotDecimals separated by mandatory white space (i.e. mws).
slotBooleanSet = slotBooleanValue *(mws slotBooleanValue)	
	A slotBooleanSet consists of one or more slotBooleanValues separated by mandatory white space (i.e. mws).
slotIntegerRange = (slotIntegerMinimum to [slotIntegerMaximum]) / (to slotIntegerMaximum)	
	A slotIntegerRange includes either a slotIntegerMinimum, a slotIntegerMaximum, or both. The slotIntegerMinimum and slotIntegerMaximum are separated by a 'to' token.

© Copyright 2020 International Health Terminology Standards Development Organisation

slotIntegerMinimum = [exclusiveMinimum] "#" integerValue	
	A slotIntegerMinimum includes a minimum integerValue preceded by a "#". By default the minimum is inclusive (i.e. the minimum value is a valid value in the range). However, the minimum value may be declared to be exclusive (i.e. not a valid value in the range) by preceding it with the exclusiveMinimum symbol (i.e. ">").
slotIntegerMaximum = [exclusiveMaximum] "#" integerValue	
	A slotIntegerMaximum includes a maximum integerValue preceded by a "#". By default the maximum is inclusive (i.e. the maximum value is a valid value in the range). However, the maximum value may be declared to be exclusive (i.e. not a valid value in the range) by preceding it with the exclusiveMaximum symbol (i.e. "<").
slotDecimalRange = (slotDecimalMinimum to [slotDecimalMaximum]) / (to slotDecimalMaximum)	
	A slotDecimalRange includes either a slotDecimalMinimum, a slotDecimalMaximum, or both. The slotDecimalMinimum and slotDecimalMaximum are separated by a 'to' token.
slotDecimalMinimum = [exclusiveMinimum] "#" DecimalValue	
	A slotDecimalMinimum includes a minimum decimalValue preceded by a "#". By default the minimum is inclusive (i.e. the minimum value is a valid value in the range). However, the minimum value may be declared to be exclusive (i.e. not a valid value in the range) by preceding it with the exclusiveMinimum symbol (i.e. ">").
slotDecimalMaximum = [exclusiveMaximum] "#" DecimalValue	
	A slotDecimalMaximum includes a maximum decimalValue preceded by a "#". By default the maximum is inclusive (i.e. the maximum value is a valid value in the range). However, the maximum value may be declared to be exclusive (i.e. not a valid value in the range) by preceding it with the exclusiveMaximum symbol (i.e. "<").
exclusiveMinimum = ">"	
	The exclusiveMinimum symbol is ">". When used before the minimum value, it declares that this value is not a valid part of the range.
exclusiveMaximum = "<"	
	The exclusiveMaximum symbol is "<". When used before the maximum value, it declares that this value is not a valid part of the range.
slotBooleanValue = true / false	
	A boolean value is either true or false.
true = ("t"/"T") ("r"/"R") ("u"/"U") ("e"/"E")	
	A boolean value of true is represented by the word "true" (case insensitive).
false = ("f"/"F") ("a"/"A") ("l"/"L") ("s"/"S") ("e"/"E")	
	A boolean value of false is represented by the word "false" (case insensitive).
slotName = "@" (nonQuoteStringValue / slotString)	
	A slotName starts with an at symbol (i.e. "@"), followed by either a quoted or unquoted string.
slotToken = definitionStatus / memberOf / constraintOperator / conjunction / disjunction / exclusion / reverseFlag / expressionComparisonOperator / numericComparisonOperator / stringComparisonOperator / booleanComparisonOperator	
	A slotToken is any token from SNOMED CT Compositional Grammar or the SNOMED CT Expression Constraint Language. This includes the definitionStatus, memberOf, constraintOperator, conjunction, disjunction, exclusion, reverseFlag, expressionComparisonOperator, numericComparisonOperator, stringComparisonOperator and booleanComparisonOperator tokens.
slotString = QM slotStringValue QM	
	A slotString is a string value enclosed in quotation marks.
nonQuoteStringValue = *(%x21 / %x23-26 / %x28-3F / %x41-5A / %x5C / %x5E-7E) ; string with no ws, quotes, at or square brackets	
	A nonQuoteStringValue includes any character that is not whitespace, quotes, or the @ symbol.
templateInformationSlot = "[[" ws slotInformation ws "]]"	
	A templateInformationSlot starts and ends with double square brackets (i.e. "[[...]]"), and contains slotInformation.
slotInformation = [cardinality ws] [slotName ws]	
	A slotInformation consists of either a cardinality, a slotName or both.

6. SNOMED CT Language Templates

In this section, we explain how the SNOMED CT template syntax can be applied to the computable languages to create template languages. In particular, we combine the Template Syntax with Compositional Grammar to create the Expression Template Language. A similar process can be used to combine the Template Syntax with the Expression Constraint Language to create an Expression Constraint Template Language.

SNOMED CT template languages are created by:

1. Combining the base language (to which the slots are added) with the SNOMED CT template syntax;
2. Adding any additional rules referenced by the SNOMED CT template syntax (e.g. the Expression Constraint Language to represent slot value constraints);
3. Removing any duplicate rules (e.g. rules that are repeated in both the base language and the expression constraint language;
4. Adding references to the template syntax in the appropriate rules of the base language to support the inclusion of slots. This involves:
    a. Renaming the first rule in the base language to add the word "Template" (e.g. from "expression" to "expressionTemplate");
    b. Adding the rule **tokenReplacementSlot** as an alternative wherever a token is referenced (e.g. "definitionStatus / tokenReplacementSlot");
    c. Adding the rules **conceptReplacementSlot** and **expressionReplacementSlot** as alternatives within the **conceptReference** rule;
    d. Adding the rule **concreteValueReplacementSlot** as an alternative attributeValue; and
    e. Adding the rule **templateInformationSlot** before each focus concept, each attribute group and each attribute name value pair.

For an example of how this process is applied to SNOMED CT compositional grammar to create expression templates, please refer to the Expression Template Language syntax in the next section.

6.1. Expression Template Language

The formal syntax for SNOMED CT expression templates (v1) is shown below. This syntax is derived by combining:

- Compositional Grammar v2.4,
- Template Syntax v1.1, and
- Expression Constraint Language v1.4.

As explained in 6. SNOMED CT Language Templates, references to the Template Syntax are also added into the Compositional Grammar rules to enable slots to be included within an expression template. Please note that rules that appear in both Compositional Grammar and the Expression Constraint Language (e.g. conceptId and term) are removed from the Expression Constraint Language (by commenting out) to avoid duplication.

```
; Compositional Grammar v2.4 with slot references (in blue)
expressionTemplate =  ws [ (definitionStatus / tokenReplacementSlot) ws] subExpression ws
subExpression = focusConcept [ws ":" ws refinement]
definitionStatus = equivalentTo / subtypeOf
equivalentTo = "==="
subtypeOf = "<<<"
focusConcept = [templateInformationSlot ws] conceptReference *(ws "+" ws [templateInformationSlot
ws] conceptReference)
conceptReference = conceptReplacementSlot / expressionReplacementSlot / ( conceptId [ws "|" ws  term ws "|"] )
conceptId = sctId
term = nonwsNonPipe *( *SP nonwsNonPipe )
refinement = (attributeSet / attributeGroup) *( ws ["," ws] attributeGroup )
attributeGroup =  [ templateInformationSlot ws ] "{" ws attributeSet ws "}"
attributeSet = attribute *(ws "," ws attribute)
attribute = [ templateInformationSlot ws ] attributeName ws "=" ws attributeValue
attributeName = conceptReference
```

attributeValue *= expressionValue / QM stringValue QM / "#" numericValue / booleanValue /* ***concreteValueReplacementSlot***
expressionValue *= conceptReference / "(" ws subExpression ws ")"*
stringValue *= 1*(anyNonEscapedChar / escapedChar)*
numericValue *= ["-"/"+"] (decimalValue / integerValue)*
integerValue *= digitNonZero *digit / zero*
decimalValue *= integerValue "." 1*digit*
booleanValue *= true / false*
true *= ("t"/"T") ("r"/"R") ("u"/"U") ("e"/"E")*
false *= ("f"/"F") ("a"/"A") ("l"/"L") ("s"/"S") ("e"/"E")*
sctId *= digitNonZero 5*17(digit)*
ws *= *(SP / HTAB / CR / LF) ; optional white space*
SP *= %x20 ; space*
HTAB *= %x09 ; tab*
CR *= %x0D ; carriage return*
LF *= %x0A ; line feed*
QM *= %x22 ; quotation mark*
BS *= %x5C ; back slash*
digit *= %x30-39*
zero *= %x30*
digitNonZero *= %x31-39*
nonwsNonPipe *= %x21-7B / %x7D-7E / UTF8-2 / UTF8-3 / UTF8-4*
anyNonEscapedChar *= HTAB / CR / LF / %x20-21 / %x23-5B / %x5D-7E / UTF8-2 / UTF8-3 / UTF8-4*
escapedChar *= BS QM / BS BS*
UTF8-2 *= %xC2-DF UTF8-tail*
UTF8-3 *= %xE0 %xA0-BF UTF8-tail / %xE1-EC 2(UTF8-tail) / %xED %x80-9F UTF8-tail / %xEE-EF 2(UTF8-tail)*
UTF8-4 *= %xF0 %x90-BF 2(UTF8-tail) / %xF1-F3 3(UTF8-tail) / %xF4 %x80-8F 2(UTF8-tail)*
UTF8-tail *= %x80-BF*
; Template Syntax v1.1
templateSlot *= templateReplacementSlot / templateInformationSlot*
templateReplacementSlot *= conceptReplacementSlot / expressionReplacementSlot / tokenReplacementSlot / concreteValueReplacementSlot*
conceptReplacementSlot *= "[[" ws "+" ws conceptReplacement [slotName ws] "]]"*
expressionReplacementSlot *= "[[" ws "+" ws expressionReplacement [slotName ws] "]]"*
tokenReplacementSlot *= "[[" ws "+" ws tokenReplacement [slotName ws] "]]"*
concreteValueReplacementSlot *= "[[" ws "+" ws concreteValueReplacement [slotName ws] "]]"*
conceptReplacement *= "id" ws ["(" ws slotExpressionConstraint ws ")" ws]*
expressionReplacement *= ["scg" ws] ["(" ws slotExpressionConstraint ws ")" ws]*
tokenReplacement *= "tok" ws ["(" ws slotTokenSet ws ")" ws]*
concreteValueReplacement *= stringReplacement / integerReplacement / decimalReplacement / booleanReplacement*
stringReplacement *= "str" ws ["(" ws slotStringSet ws ")" ws]*
integerReplacement *= "int" ws ["(" ws slotIntegerSet ws ")" ws]*
decimalReplacement *= "dec" ws ["(" ws slotDecimalSet ws ")" ws]*
booleanReplacement *= "bool" ws ["(" ws slotBooleanSet ws ")" ws]*
slotTokenSet *= slotToken *(mws slotToken)*
slotStringSet *= slotString *(mws slotString)*
slotIntegerSet *= (slotIntegerValue / slotIntegerRange) *(mws (slotIntegerValue / slotIntegerRange))*
slotDecimalSet *= (slotDecimalValue / slotDecimalRange) *(mws (slotDecimalValue / slotDecimalRange))*
slotBooleanSet *= slotBooleanValue *(mws slotBooleanValue)*
slotIntegerRange *= (slotIntegerMinimum to [slotIntegerMaximum]) / (to slotIntegerMaximum)*
slotIntegerMinimum *= [exclusiveMinimum] slotIntegerValue*
slotIntegerMaximum *= [exclusiveMaximum] slotIntegerValue*
slotIntegerValue *= "#" digitNonZero *digit / "#" zero*
slotDecimalRange *= (slotDecimalMinimum to [slotDecimalMaximum]) / (to slotDecimalMaximum)*

© Copyright 2020 International Health Terminology Standards Development Organisation

```
slotDecimalMinimum = [ exclusiveMinimum ] slotDecimalValue
slotDecimalMaximum = [ exclusiveMaximum ] slotDecimalValue
slotDecimalValue = "#" integerValue "." 1*digit
exclusiveMinimum = ">"
exclusiveMaximum = "<"
slotBooleanValue = true / false
; true = ("t"/"T") ("r"/"R") ("u"/"U") ("e"/"E")
; false = ("f"/"F") ("a"/"A") ("l"/"L") ("s"/"S") ("e"/"E")
slotName = "@" (nonQuoteStringValue / slotString)
slotToken = definitionStatus / memberOf / constraintOperator / conjunction / disjunction / exclusion / reverseFlag / expressionComparisonOperator / numericComparisonOperator / stringComparisonOperator / booleanComparisonOperator
slotString = QM slotStringValue QM
nonQuoteStringValue  = *(%x21 / %x23-26 / %x2A-3F / %x41-5A / %x5C / %x5E-7E)   ; string with no ws, quotes, at, square brackets or round brackets
templateInformationSlot  = "[[" ws slotInformation ws "]]"
slotInformation = [cardinality ws] [slotName ws]
; Expression Constraint Language v1.4 - Note that some rules are commented out because they are repeated in the Compositional Grammar rules above.
slotExpressionConstraint = ws ( slotRefinedExpressionConstraint / slotCompoundExpressionConstraint / slotDottedExpressionConstraint / slotSubExpressionConstraint ) ws
slotRefinedExpressionConstraint = slotSubExpressionConstraint ws ":" ws slotEclRefinement
slotCompoundExpressionConstraint = slotConjunctionExpressionConstraint / slotDisjunctionExpressionConstraint / slotExclusionExpressionConstraint
slotConjunctionExpressionConstraint = slotSubExpressionConstraint 1*(ws conjunction ws slotSubExpressionConstraint)
slotDisjunctionExpressionConstraint = slotSubExpressionConstraint 1*(ws disjunction ws slotSubExpressionConstraint)
slotExclusionExpressionConstraint = slotSubExpressionConstraint ws exclusion ws slotSubExpressionConstraint
slotDottedExpressionConstraint = slotSubExpressionConstraint 1*(ws slotDottedExpressionAttribute)
slotDottedExpressionAttribute = dot ws slotEclAttributeName
slotSubExpressionConstraint = [constraintOperator ws] [memberOf ws] (slotEclFocusConcept / "(" ws slotExpressionConstraint ws ")")
slotEclFocusConcept = slotEclConceptReference / wildCard
dot = "."
memberOf = "^"
slotEclConceptReference = conceptId [ws "|" ws term ws "|"]
; conceptId = sctId
; term = 1*nonwsNonPipe *( 1*SP 1*nonwsNonPipe )
wildcard = "*"
constraintOperator = childOf / descendantOrSelfOf / descendantOf / parentOf / ancestorOrSelfOf / ancestorOf
descendantOf = "<"
descendantOrSelfOf = "<<"
childOf = "<!"
ancestorOf = ">"
ancestorOrSelfOf = ">>"
parentOf = ">!"
conjunction = (("a"/"A") ("n"/"N") ("d"/"D") mws) / ","
disjunction = ("o"/"O") ("r"/"R") mws
exclusion = ("m"/"M") ("i"/"I") ("n"/"N") ("u"/"U") ("s"/"S") mws
slotEclRefinement = slotSubRefinement ws [slotConjunctionRefinementSet / slotDisjunctionRefinementSet]
slotConjunctionRefinementSet = 1*(ws conjunction ws slotSubRefinement)
slotDisjunctionRefinementSet = 1*(ws disjunction ws slotSubRefinement)
slotSubRefinement = slotEclAttributeSet / slotEclAttributeGroup / "(" ws slotEclRefinement ws ")"
slotEclAttributeSet = slotSubAttributeSet ws [slotConjunctionAttributeSet / slotDisjunctionAttributeSet]
```

© Copyright 2020 International Health Terminology Standards Development Organisation

slotConjunctionAttributeSet = 1*(ws conjunction ws slotSubAttributeSet)
slotDisjunctionAttributeSet = 1*(ws disjunction ws slotSubAttributeSet)
slotSubAttributeSet = slotEclAttribute / "(" ws slotEclAttributeSet ws ")"
slotEclAttributeGroup = ["[" cardinality "]" ws] "{" ws slotEclAttributeSet ws "}"
slotEclAttribute = ["[" cardinality "]" ws] [reverseFlag ws] slotEclAttributeName ws (expressionComparisonOperator ws slotSubExpressionConstraint / numericComparisonOperator ws slotNumericValue / stringComparisonOperator ws slotStringValue / booleanComparisonOperator ws booleanValue)
cardinality = minValue to maxValue
minValue = nonNegativeIntegerValue
to = ".."
maxValue = nonNegativeIntegerValue / many
many = "*"
reverseFlag = "R"
slotEclAttributeName = slotSubExpressionConstraint
expressionComparisonOperator = "=" / "!="
numericComparisonOperator = "=" / "!=" / "<=" / "<" / ">=" / ">"
stringComparisonOperator = "=" / "!="
booleanComparisonOperator = "=" / "!="
slotNumericValue = "#" ["-"/"+"] (slotDecimalValue / slotIntegerValue)
slotStringValue = QM 1*(anyNonEscapedChar / escapedChar) QM
; integerValue = digitNonZero *digit / zero
; decimalValue = integerValue "." 1*digit
; booleanValue = true / false
; true = ("t"/"T") ("r"/"R") ("u"/"U") ("e"/"E")
; false = ("f"/"F") ("a"/"A") ("l"/"L") ("s"/"S") ("e"/"E")
nonNegativeIntegerValue = (digitNonZero *digit) / zero
; sctId = digitNonZero 5*17(digit)
; ws = *(SP / HTAB / CR / LF / comment) ; optional white space
mws = 1*(SP / HTAB / CR / LF / comment) ; mandatory white space
comment = "/*" *(nonStarChar / starWithNonFSlash) "*/"
nonStarChar = SP / HTAB / CR / LF / %x21-29 / %x2B-7E /UTF8-2 / UTF8-3 / UTF8-4
starWithNonFSlash = %x2A nonFSlash
nonFSlash = SP / HTAB / CR / LF / %x21-2E / %x30-7E /UTF8-2 / UTF8-3 / UTF8-4
; SP = %x20 ; space
; HTAB = %x09 ; tab
; CR = %x0D ; carriage return
; LF = %x0A ; line feed
; QM = %x22 ; quotation mark
; BS = %x5C ; back slash
; digit = %x30-39
; zero = %x30
; digitNonZero = %x31-39
; nonwsNonPipe = %x21-7B / %x7D-7E / UTF8-2 / UTF8-3 / UTF8-4
; anyNonEscapedChar = SP / HTAB / CR / LF / %x20-21 / %x23-5B / %x5D-7E / UTF8-2 / UTF8-3 / UTF8-4
; escapedChar = BS QM / BS BS
; UTF8-2 = %xC2-DF UTF8-tail
; UTF8-3 = %xE0 %xA0-BF UTF8-tail / %xE1-EC 2(UTF8-tail) / %xED %x80-9F UTF8-tail / %xEE-EF 2(UTF8-tail)
; UTF8-4 = %xF0 %x90-BF 2(UTF8-tail) / %xF1-F3 3(UTF8-tail) / %xF4 %x80-8F 2(UTF8-tail)
; UTF8-tail = %x80-BF

© Copyright 2020 International Health Terminology Standards Development Organisation

7. Processing Expression Templates

When an expression template is used, a number of steps must be performed to create a valid SNOMED CT expression that conforms to the given template.

In this section, we explain the steps involved in *processing an expression template*. We also explain the pre-processing and post-processing tasks involved.

The figure below illustrates this process. Please note that these steps do not necessarily need to be performed in this order.

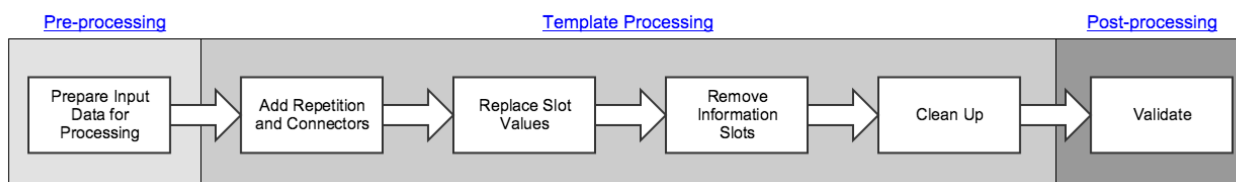


Figure 7-1: Template processing steps

7.1. Preparing Input Data

Before a template can be processed, it is important that the input data is represented in a clear and unambiguous way. This is required to ensure that the template is processed in the expected manner, and the intended results are produced. In this section, we explain some of the considerations in representing and preparing the input data for processing.

Input Data Representation

Template input data may be represented in a variety of forms, ranging from flat tabular structures to nested serializations. Irrespective of the format, however, it is important that there is no ambiguity as to how each piece of input data should be used to create the resulting expressions. This can be particularly challenging where repetition of relationship groups or attribute name-value pairs is required.

The UML diagram below illustrates the logical structure of expression template input data. Each set of *Expression Template Input Data* includes the data intended to be used to create one or more expressions. The data used to populate a single expression is referred to in this model as *Expression Data*. Each *Expression Data* (identified by an expression id), may include at most one *Definition Status Slot* (with a slot name and a definitionStatus value), zero

© Copyright 2020 International Health Terminology Standards Development Organisation

or more *Focus Concept Slots* (each with a slotName and zero or more values), zero or more *Relationship Group Slots* (each with a group name), and zero or more ungrouped *Attribute Name-Value Pair Slots* (each with a name).
Each Relationship Group Slot has zero or more *Relationship Group Data* instances in the input data (each identified by a group id). Each of these *Relationship Group Data* instances has input data for one to many *Attribute Name-Value Pair Slots*. And for each *Attribute Name-Value Pair Slot* within a *Relationship Group Data* instance, there are zero to many *Attribute Name-Value Pair* Data instances (identified by an anvPair id), each with at most one *Attribute Name Slot* (with name and value), and at most one *Attribute Value Slot* (with name and either a simple data value, or an *Expression Data* instance of its own).

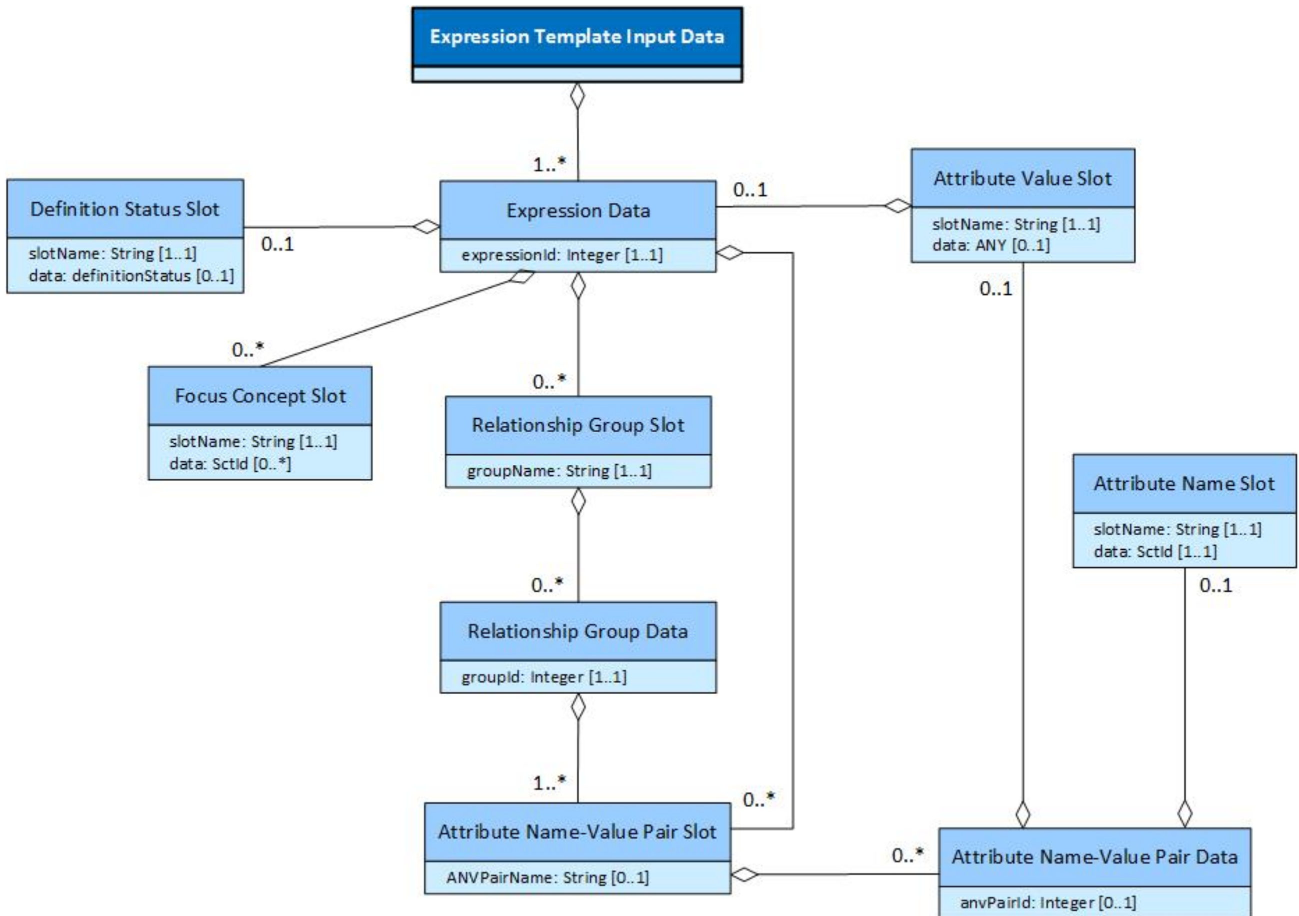


Figure 7.1-1: Logical model of expression template input data

Input Data Examples

In this section, we provide some examples of unambiguous expression template input data, and discuss how this input data can be used to populate each expression.

Example 1

The expression template below is used to create expressions that represent a |Disease| with one or more |Finding site| and |Associated morphology|. When using expression templates, such as this one, in which attribute name-value pairs and relationship groups may be repeated, the input data should be explicit about which data values are used to populate each slot, and how these values are grouped into relationship groups.

```
[[+tok (=== <<<) @DefStatus)]] [[+id (<< 64572001 |Disease| ) @Disease]] :
  [[@Group]] { 363698007 |Finding site| = [[+ (<< 272673000 |Bone structure| ) @Site]],
         116676008 |Associated morphology| = [[+ (<< 72704001 |Fracture| ) @Morphology]] }
```

To support the creation of input data for this expression template, the logical model in Figure 7.1-1 above can be specialized by replacing the 'Slot' classes (e.g. 'Relationship Group Slot' and 'Attribute Value Slot') with the name of the respective slots in the template, and simplifying where possible. The resulting logical model of input data for the above expression template is shown below in Figure 7.1-2. Please note that this model has been simplified by removing unnamed logical classes, which have a cardinality of 1..1 and no data attribute. For example, attribute name-value slots are not required in this example. In general, attribute name-value slots are only required where both the attribute name and the attribute value are represented using a slot.

© Copyright 2020 International Health Terminology Standards Development Organisation

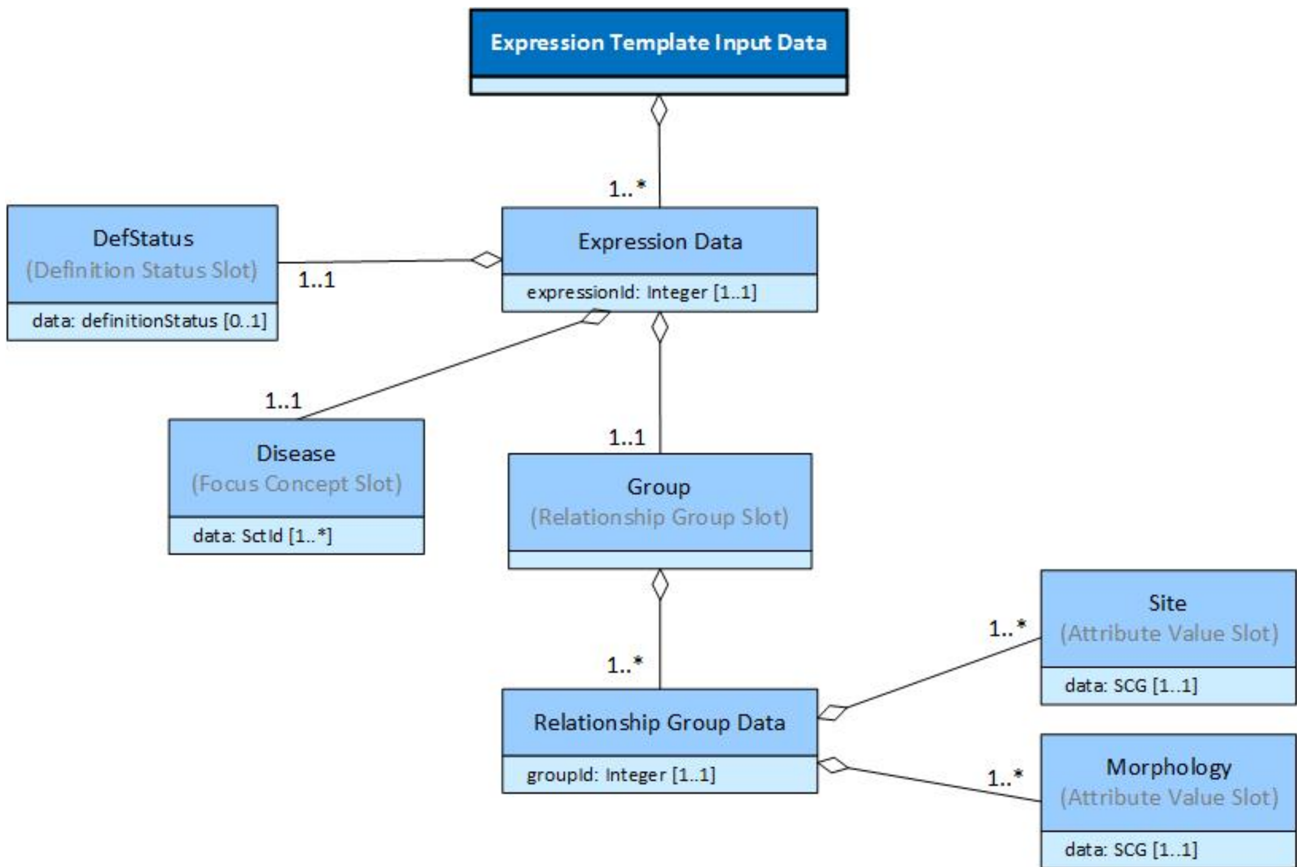


Figure 7.1-2: Logical model of example 1 input data

By populating this logical model with input data, as shown below in Table 7.1-1, the expression template can be processed to generate completed expressions. Please note that the first column in the table below is used to group together the input data intended to populate each expression. Subsequent columns are named according to the associated slot in the expression template. Relationship group slots are used to group the data that is intended to populate a single relationship group. Attribute name-value slots are not required in this example. They are are only required where both the attribute name and attribute value use a slot.

© Copyright 2020 International Health Terminology Standards Development Organisation

Table 7.1-1: Input Data for Example 1

<table>
<tr><th>Expression Data</th><th>DefStatus</th><th>Disease</th><th>Group</th><th>Site</th><th>Morphology</th></tr>
<tr><td>1</td><td>===</td><td>46866001 |Fracture of lower limb|</td><td>1</td><td>12611008 |Bone structure of tibia|</td><td>72704001 |Fracture|</td></tr>
<tr><td rowspan="2">2</td><td rowspan="2"><<<</td><td>92196005 |Benign neoplasm of lung|</td><td>1</td><td>39607008 |Lung structure|</td><td>3898006 |Neoplasm, benign|</td></tr>
<tr><td>92038006 |Benign neoplasm of bronchus|</td><td>2</td><td>955009 |Bronchial structure|</td><td>3898006 |Neoplasm, benign|</td></tr>
<tr><td rowspan="2">3</td><td rowspan="2"><<<</td><td>60667009 |Closed fracture of rib|</td><td rowspan="2">1</td><td>113197003 |Bone structure of rib|</td><td rowspan="2">34305007 |Fracture, multiple, closed|</td></tr>
<tr><td>36991002 |Closed fracture of upper limb|</td><td>371195002 |Bone structure of upper limb|</td></tr>
<tr><td rowspan="3">4</td><td rowspan="3">===</td><td>16119006 |Abscess of jaw|</td><td rowspan="3">1</td><td rowspan="3">70925003 |Bone structure of maxilla|</td><td rowspan="3">44132006 |Abscess|</td></tr>
<tr><td>109327001 |Abscess of facial bone|</td></tr>
<tr><td>128234004 |Disorder of maxilla|</td></tr>
</table>

Using input data shown in Table 7.1-1 to populate the given expression template will result in following four expressions.

Expression	
1	=== 46866001 |Fracture of lower limb| : { 363698007 |Finding site| = 12611008 |Bone structure of tibia| , 116676008 |Associated morphology| = 72704001 |Fracture| }
2	<<< 92196005 |Benign neoplasm of lung| + 92038006 |Benign neoplasm of bronchus| : { 363698007 |Finding site| = 39607008 |Lung structure| , 116676008 |Associated morphology| = 3898006 |Neoplasm, benign| }, { 363698007 |Finding site| = 955009 |Bronchial structure| , 116676008 |Associated morphology| = 3898006 |Neoplasm, benign| }
3	<<< 60667009 |Closed fracture of rib| + 36991002 |Closed fracture of upper limb| : { 363698007 |Finding site| = 113197003 |Bone structure of rib| , 363698007 |Finding site| = 371195002 |Bone structure of upper limb| , 116676008 |Associated morphology| = 34305007 |Fracture, multiple, closed| }
4	=== 16119006 |Abscess of jaw| + 109327001 |Abscess of facial bone| + 128234004 |Disorder of maxilla| : { 363698007 |Finding site| = 70925003 |Bone structure of maxilla| , 116676008 |Associated morphology| = 44132006 |Abscess| }

Example 2

The expression template below is used as a pattern for family history expressions. It contains a nested relationship group (i.e. SSgroup) inside the outer relationship group (i.e. AFgroup). To populate this expression template, the input data must be clear as to where each value should be used, and how these values should be grouped into relationship groups and expressions.

```
[[+id (<< 413350009 |Finding with explicit context| ) @Condition]]:
  [[ 1..2 @AFgroup ]] { [[1..1]] 246090004 |Associated finding| = ([[+id (<< 404684003 |Clinical finding|
) @Finding]]:
    [[0..1 @SSgroup]] { [[0..1]] 246112005 |Severity| = [[+id (< 272141005 |Severities| ) @Severity]],
             [[0..1]] 363698007 |Finding site| = [[+id (< 91723000 |Anatomical structure| ) @Site]] }),
         [[1..1]] 408732007 |Subject relationship context| = [[+id (< 444148008 |Person in family of
subject| ) @Relationship]],
         [[1..1]] 408731000 |Temporal context| = [[+id (< 410510008 |Temporal context value|
) @Time]],
         [[1..1]] 408729009 |Finding context| = [[+id (< 410514004 |Finding context value| ) @Context]] }
```

To support the creation of input data for this expression template, the logical model in Figure 7.1-1 above can be specialized as shown below in Figure 7.1-3. Please note that this model has been simplified by removing unnamed logical classes, which have a cardinality of 1..1 and no data attribute.

© Copyright 2020 International Health Terminology Standards Development Organisation

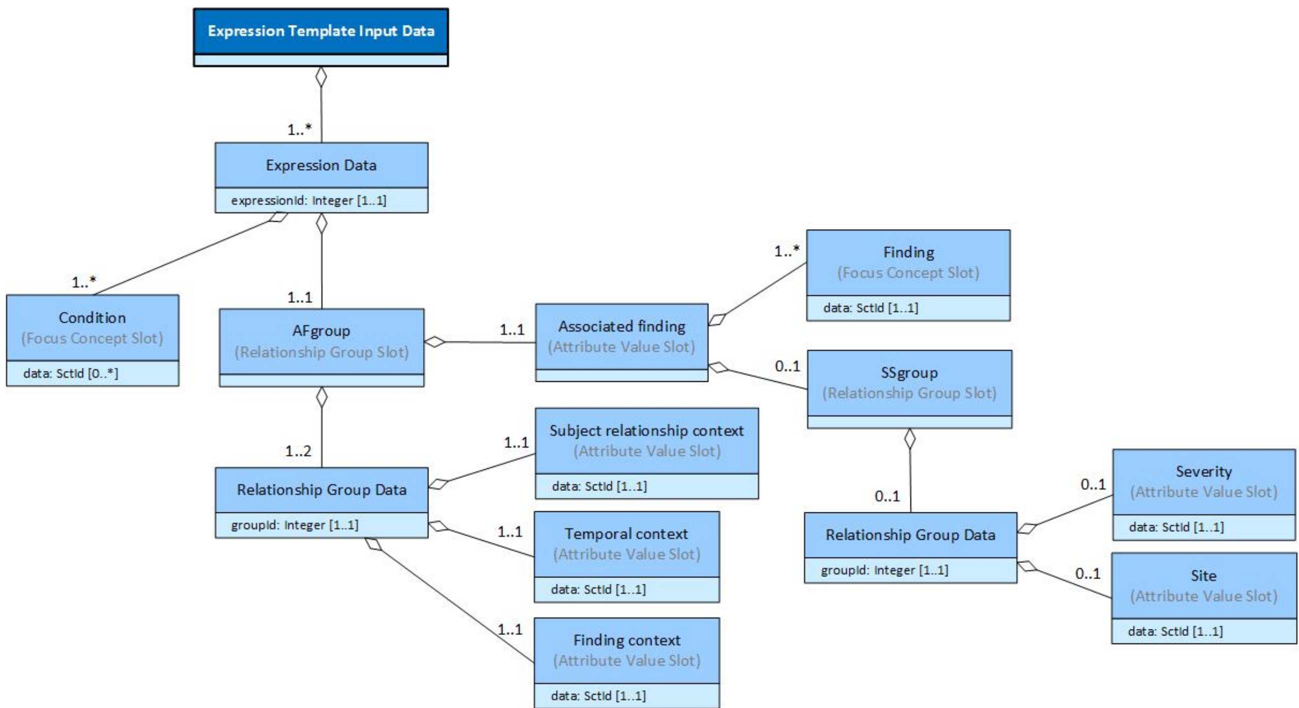


Figure 7.1-3: Logical model of example 2 input data

This logical model can be populated with input data, as shown below in Table 7.1-2.

Table 7.1-2: Input Data for Example 2

<table>
<tr><th>Expression Data</th><th>Condition</th><th>AFgroup</th><th>Finding</th><th>SSgroup</th><th>Severity</th><th>Site</th><th>Relationship</th><th>Time</th><th>Context</th></tr>
<tr><td>1</td><td>266898002 |Family history: Respiratory disease|</td><td>1</td><td>195967001 |Asthma|</td><td>1</td><td>24484000 |Severe|</td><td></td><td>444301002 |Mother of subject|</td><td>410589000 |All times past|</td><td>410515003 |Known present|</td></tr>
<tr><td rowspan="2">2</td><td>161077003 |Father smokes|</td><td>1</td><td>77176002 |Smoker|</td><td>1</td><td>255604002 |Mild|</td><td></td><td>444295003 |Father of subject|</td><td>15240007 |Current|</td><td>410515003 |Known present|</td></tr>
<tr><td>161078008 |Mother smokes|</td><td>2</td><td>77176002 |Smoker|</td><td>1</td><td>24484000 |Severe|</td><td></td><td>444301002 |Mother of subject|</td><td>15240007 |Current|</td><td>410515003 |Known present|</td></tr>
<tr><td rowspan="2">3</td><td>160288009 |Family history: neoplasm of skin|</td><td rowspan="2">1</td><td rowspan="2">372130007 |Malignant neoplasm of skin|</td><td>1</td><td>6736007 |Moderate|</td><td>113179006 |Skin structure of nose|</td><td rowspan="2">444304005 |Sister of subject|</td><td rowspan="2">410511007 |Current or past (actual)|</td><td rowspan="2">410515003 |Known present|</td></tr>
<tr><td>275937001 |Family history of cancer|</td><td>2</td><td>255604002 |Mild|</td><td>88089004 |Skin structure of lip|</td></tr>
</table>

Using the input data shown above to process the given expression template will result in following three expressions.

Expression	
1	266898002 |Family history: Respiratory disease| : { 246090004 |Associated finding| = (195967001 |Asthma| : { 246112005 |Severity| = 24484000 |Severe| }), 408732007 |Subject relationship context| = 444301002 |Mother of subject| , 408731000 |Temporal context| = 410511007 |Current or past (actual)| , 408729009 |Finding context| = 410515003 |Known present| }
2	161077003 |Father smokes| + 161078008 |Mother smokes| : { 246090004 |Associated finding| = (77176002 |Smoker| : { 246112005 |Severity| = 24484000 |Severe| }), 408732007 |Subject relationship context| = 444295003 |Father of subject| , 408731000 |Temporal context| = 15240007 |Current| , 408729009 |Finding context| = 410515003 |Known present| }, { 246090004 |Associated finding| = (77176002 |Smoker| : { 246112005 |Severity| = 255604002 |Mild| }), 408732007 |Subject relationship context| = 444301002 |Mother of subject| , 408731000 |Temporal context| = 15240007 |Current| , 408729009 |Finding context| = 410515003 |Known present| }
3	160288009 |Family history: neoplasm of skin| + 275937001 |Family history of cancer| : { 246090004 |Associated finding| = (372130007 |Malignant neoplasm of skin| : { 246112005 |Severity| = 6736007 |Moderate| , 363698007 |Finding site| = 113179006 |Skin structure of nose| }, { 246112005 |Severity| = 255604002 |Mild| , 363698007 |Finding site| = 88089004 |Skin structure of lip| }), 408732007 |Subject relationship context| = 444304005 |Sister of subject| , 408731000 |Temporal context| = 410511007 |Current or past (actual)| , 408729009 |Finding context| = 410515003 |Known present| }

Example 3

The expression template below represents a procedure with a single method and one or more procedure devices. Please note that in the first attribute name-value pair, both the attribute name and the attribute value use a slot. Because this name-value pair is repeatable, the input data needs to include an attribute name-value pair slot to ensure that the corresponding attribute name and attribute value stays connected.

```
[[+id (<< 71388002 |Procedure| ) @Procedure]]:
  [[1..1 @Group]]
    { [[1..* @PD_ANVpair]] [[+id (< 405815000 |Procedure device| ) @DeviceType]] = [[+ (< 260787004 |
Physical object| ) @Device]],
     [[1..1]] 260686004 |Method| = [[+ (< 129264002 |Action (qualifier value)| ) @Method]] }
```

To support the creation of input data for this expression template, the logical model in Figure 7.1-1 above can be specialized as shown below in Figure 7.1-4. Please note that this model has been simplified by removing unnamed logical classes, which have a cardinality of 1..1 and no data attribute.

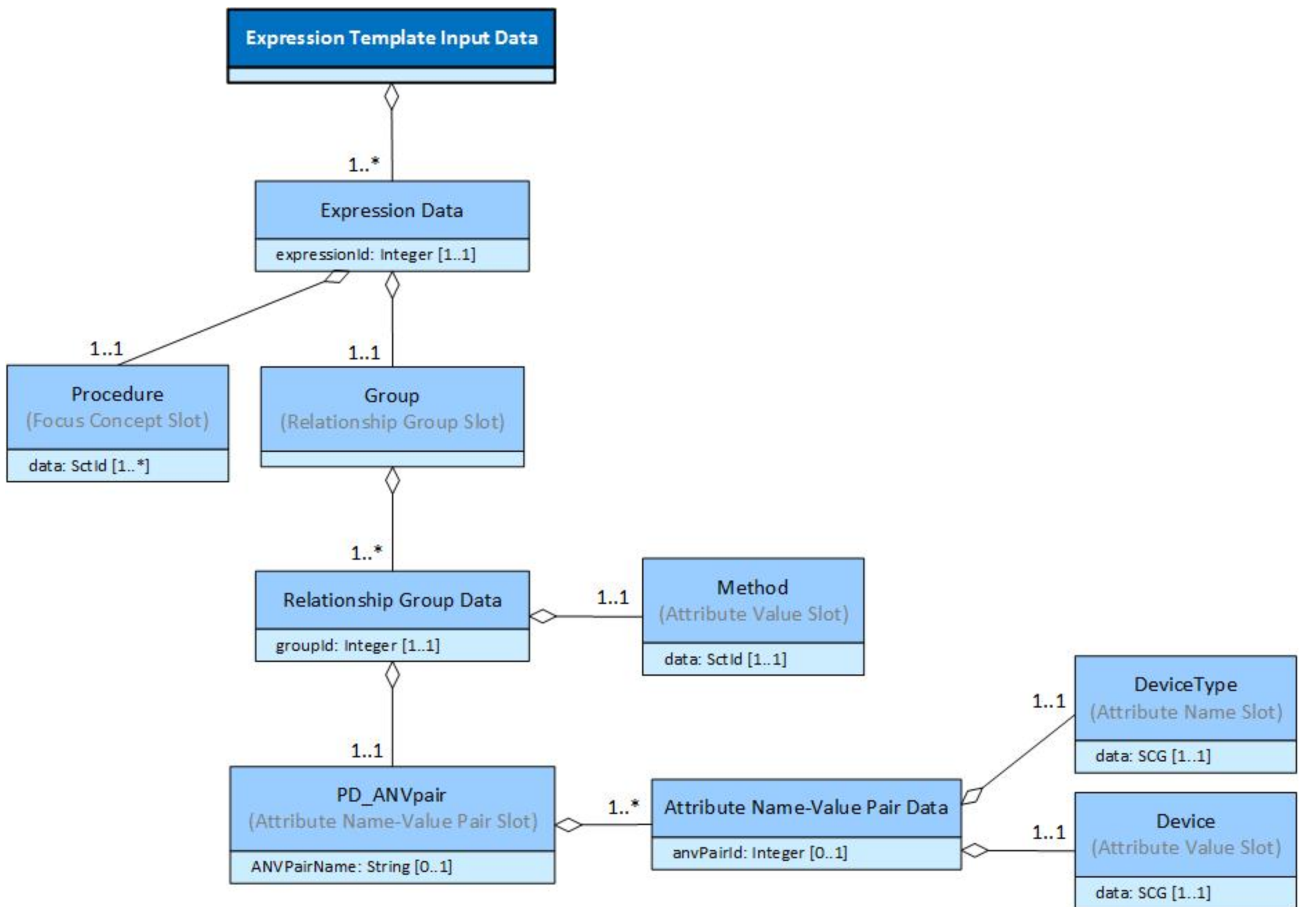


Figure 7.1-4: Logical model of example 3 input data

This logical model can be populated with input data, as shown below in Table 7.1-3. Please note that because the first attribute name-value pair is repeatable and uses a replacement slot for both the attribute name and attribute value, the input data needs to include the attribute name-value pair slot to ensure that the corresponding attribute name and attribute value stays connected.

© Copyright 2020 International Health Terminology Standards Development Organisation

Table 7.1-3: Input Data for Example 3

<table>
<tr><th>Expression Data</th><th>Procedure</th><th>Group</th><th>PD_ANVpair</th><th>DeviceType</th><th>Device</th><th>Method</th></tr>
<tr><td>1</td><td>387713003 |Surgical procedure|</td><td>1</td><td>1</td><td>363699004 |Direct device|</td><td>2282003 |Breast prosthesis, device|</td><td>257867005 |Insertion - action|</td></tr>
<tr><td rowspan="2">2</td><td rowspan="2">71388002 |Procedure|</td><td rowspan="2">1</td><td>1</td><td>363699004 |Direct device|</td><td>313025003 |Hearing aid battery|</td><td rowspan="2">282089006 |Replacement - action|</td></tr>
<tr><td>2</td><td>363710007 |Indirect device|</td><td>6012004 |Hearing aid, device|</td></tr>
</table>

Using the input data shown in Table 7.1-3 to process the given expression template will result in the following two expressions.

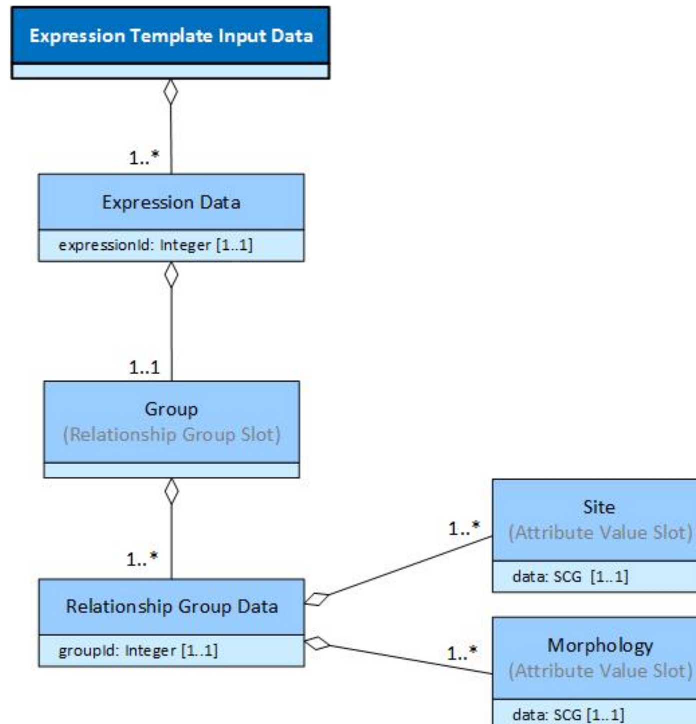
Expression	
1	387713003 |Surgical procedure| : { 363699004 |Direct device| = 2282003 |Breast prosthesis, device| , 260686004 |Method| = 257867005 |Insertion - action| }
2	384728007 |Replacement of device| : { 363699004 |Direct device| = 313025003 |Hearing aid battery| , 363710007 |Indirect device| = 6012004 |Hearing aid, device| , 260686004 |Method| = 282089006 |Replacement - action| }

Example 4

The expression template below represents a |Disease| with one or more values for |Finding site| and |Associated morphology|, grouped into one or more relationship groups.

```
64572001 |Disease| :
  [[@Group]] { 363698007 |Finding site| = [[+ (<< 272673000 |Bone structure| ) @Site]],
          116676008 |Associated morphology| = [[+ (<< 72704001 |Fracture| ) @Morphology]] }
```

To support the creation of input data for this expression template, the logical model in Figure 7.1-1 above can be specialized as shown below in Figure 7.1-5. Please note that this model has been simplified by removing unnamed logical classes, which have a cardinality of 1..1 and no data attribute.



© Copyright 2020 International Health Terminology Standards Development Organisation

Figure 7.1-5: Logical model of example 4 input data

Table 7.1-4 below shows some example input data for the above template represented using the tabular format used in the previous examples.

Table 7.1-4: Input Data for Example 4

Expression Data	Group	Site	Morphology
1	1	312763008 |Bone structure of trunk|	72704001 |Fracture|
	2	84667006 |Bone structure of cervical vertebra|	72704001 |Fracture|
2	1	71341001 |Bone structure of femur|	72704001 |Fracture|
3	1	12611008 |Bone structure of tibia|	72704001 |Fracture|

In addition to this tabular representation, there are a wide variety of other possible formats for representing template input data, including json, xml, tsv, csv etc. The exact format used will depend on the format required by the template processor. For example, the above input data can be represented in JSON as shown below.

© Copyright 2020 International Health Terminology Standards Development Organisation

Example JSON Representation of Input Data


```
{"Expression Data": [
        { "Group": [
                {  "Site":"312763008 |Bone structure of trunk|",
                    "Morphology":"72704001 |Fracture|" },
                {  "Site": "84667006 |Bone structure of cervical vertebra|",
                    "Morphology": "72704001 |Fracture|" } ] },
        { "Group": [
                {  "Site":"71341001 |Bone structure of femur|",
                    "Morphology": "72704001 |Fracture|" } ] },
        { "Group":[
                {  "Site":"12611008 | Bone structure of tibia|",
                    "Morphology": "72704001 |Fracture|" } ] } ] }
```


Simplification of Data Representation

While it is important that there is no ambiguity as to how each piece of input data should be used in processing the associated expression template, there are often opportunities to make the input data much simpler than is represented in the full logical model above. In particular:

- When the maximum cardinality of a relationship group is 1, there is no need to include the *relationship group* slot in the input data to group the attributes it contains;
- When the maximum cardinality of an attribute name-value pair is 1, there is no need to include the *attribute name-value pair slot* in the input data to group the corresponding name and value pairs;
- When there is either an attribute name slot with a fixed attribute value, or a fixed attribute name with an attribute value slot, there is no need to include the *attribute name-value pair slot* in the input data to group the corresponding name and value pairs.

With this in mind, the examples in 8. Expression Template Examples simplify the input data, where appropriate, using these assumptions and an implicit association with the logical model described above.

7.2. Template Processing

In this section, we explain the steps involved in generating a set of SNOMED CT expressions from a SNOMED CT expression template and a set of input data. As illustrated in 7. Processing Expression Templates, this includes:

- Adding repetition and connectors;
- Replacing slot values;
- Removing information slots; and
- Cleaning up.

Please note that these steps do not necessarily need to be performed in this order.

Add Repetition and Connectors

Expression templates may use explicit or default cardinalities to indicate parts of the expression that may be repeated (i.e. where the maximum cardinality is great than 1). In particular, a focus concept may be repeated, a relationship group may be repeated, or an attribute-value pair may be repeated. Repetition, however, is only required when multiple values are provided in the input data for a particular part of the template (see 7.1. Preparing Input Data). When multiple values are found in the input data, and this corresponds to a repeatable part of the expression template, the associated part of the expression must be duplicated to enable each separate value to be included.

For example, let's consider the following expression template and input data.

```
[[1..*]] [[+id (<< 64572001 |Disease| ) @Disease]]:
   [[1..* @Group]] { [[1..* @CausedBy]] 246075003 |Causative agent| = [[+id (< 410607006 |Organism|
) @Organism]],
                [[0..1 @After]] 255234002 |After| = [[+id (< 404684003 |Clinical finding| ) @Finding]] }
```

<table>
<tr><th>Expression Data</th><th>Disease</th><th>Group</th><th>CausedBy</th><th>Organism</th><th>After</th><th>Finding</th></tr>
<tr><td rowspan="3">1</td><td>40733004 |Disorder due to infection|</td><td rowspan="2">1</td><td>1</td><td>80166006 |Streptococcus pyogenes|</td><td rowspan="2">1</td><td rowspan="2">58718002 |Rheumatic fever|</td></tr>
<tr><td rowspan="2">19342008 |Subacute disease|</td><td>2</td><td>113985000 |Streptococcus gallolyticus|</td></tr>
<tr><td>2</td><td>1</td><td>49872002 |Virus|</td><td>1</td><td></td></tr>
</table>

The input data above includes 2 focus concepts, 2 relationship groups, and 2 CausedBy attribute-name value pairs in the first relationship group. To support this input data, the expression template must duplicate the focus concept, relationship group, and the attribute name-value pair (in the first relationship group).

When parts of an expression template are repeated, it is important to ensure that the correct connector is added between repeated parts. For example, a "+" is added between repeated focus concepts, a "," is added between repeated relationship groups (although this is optional), and a "," is added between repeated name-value pairs.

The result of adding repetition and connectors to the above expression template (for the given input data) is shown below:

```
[[1..*]] [[+id (<< 64572001 |Disease| ) @Disease]] + [[+id (<< 64572001 |Disease| ) @Disease]]:
   [[1..* @Group]] { [[1..* @CausedBy]] 246075003 |Causative agent| = [[+id (< 410607006 |Organism|
) @Organism]],
                     246075003 |Causative agent| = [[+id (< 410607006 |Organism| ) @Organism]],
                [[0..1 @After]] 255234002 |After| = [[+id (< 404684003 |Clinical finding| ) @Finding]] },
              { [[1..* @CausedBy]] 246075003 |Causative agent| = [[+id (< 410607006 |Organism|
) @Organism]],
                [[0..1 @After]] 255234002 |After| = [[+id (< 404684003 |Clinical finding| ) @Finding]] }
```

Please note that when an expression is used to populate an attribute value, it may be required to add brackets around the attribute value to ensure syntactic correctness of the resulting expression

Replace Slot Values

With the repetition and connectors in place, the replacement slots must now be replaced with values. Each definition status, focus concept, attribute name and attribute value replacement slot must be removed, and the corresponding value from the input data inserted into the expression. After this step, the expression template above would look like the following:

© Copyright 2020 International Health Terminology Standards Development Organisation

```
[[1..*]] 40733004 |Disorder due to infection| + 19342008 |Subacute disease| :
  [[1..* @Group]] { [[1..* @CausedBy]] 246075003 |Causative agent| = 80166006 |Streptococcus pyogenes| ,
                  246075003 |Causative agent| = 113985000 |Streptococcus gallolyticus| ,
            [[0..1 @After]] 255234002 |After| = 58718002 |Rheumatic fever| },
          { [[1..* @CausedBy]] 246075003 |Causative agent| = 49872002 |Virus| ,
            [[0..1 @After]] 255234002 |After| = }
```

Remove Information Slots

Once the information contained in the information slots has been interpreted and processed, the information slots can be removed from the template. After this step, the expression template above would look like the following:

```
40733004 |Disorder due to infection| + 19342008 |Subacute disease| :
 { 246075003 |Causative agent| = 80166006 |Streptococcus pyogenes| ,
   246075003 |Causative agent| = 113985000 |Streptococcus gallolyticus| ,
   255234002 |After| = 58718002 |Rheumatic fever| },
 { 246075003 |Causative agent| = 49872002 |Virus| ,
   255234002 |After| = }
```

Clean Up

Clean up is an activity that can occur at any or all stages of template processing. However, any clean up that has not occurred during the previous stages, should be performed before completion.

In particular, this step involves removing any extra brackets (i.e. "(...)"), braces (i.e. "{ ... }"), colons (i.e. ":"), equals (i.e. "="), attribute names or connectors (i.e. "+" or ",") that would cause the resulting expression to be syntactically invalid. Extraneous characters are most likely to occur when optional parts of an expression template have no corresponding value in the populated expression. When this occurs, clean up may be required to remove these characters.

In our example expression above, this step would involve removing the second instance of the attribute name 255234002 |After|, the comma before it, and the equals signs after. The resulting expression would look like:

```
40733004 |Disorder due to infection| + 19342008 |Subacute disease| :
 { 246075003 |Causative agent| = 80166006 |Streptococcus pyogenes| ,
   246075003 |Causative agent| = 113985000 |Streptococcus gallolyticus| ,
   255234002 |After| = 58718002 |Rheumatic fever| },
 { 246075003 |Causative agent| = 49872002 |Virus| }
```

7.3. Post-processing Validation

After an expression template has been processed, it is important to validate the expressions that are generated. This validation process may include checking that:

- Each expression is syntactically valid;
- Each expression conforms to the structure and constraints defined in the template; and
- Each expression is valid according to the SNOMED CT concept model.

Please note that concept model validation may not be required in all use cases.

Syntactic Validation

The SNOMED CT expressions generated when processing an expression template must be syntactically valid according the SNOMED CT compositional grammar syntax. To test for syntactic correctness, an expression parser, based on the ABNF rules defined in the SNOMED CT compositional grammar specification is used. For more information, please refer to 7.2 Parsing.

Template Validation

The SNOMED CT expressions generated when processing an expression template must conform to the structure and constraints defined in the template. This includes:

- Each part of the resulting expression should conform structurally to a correspond part of the template, in the same order that it appears in the template;
- All mandatory parts of the expression template must appear at least once in the resulting expression (i.e. where the minimum cardinality is > 0);
- All non-repeatable parts of the expression template must appear at most once in the resulting expression (ie. where the maximum cardinality is 1); and
- Each token, concept, expression or value in the resulting expression, that was added as a result of replacing a slot, must conform to the type (e.g. id, scg) and value constraint (e.g. expression constraint, value list or range) defined in the corresponding slot;

Concept Model Validation

In most cases, the SNOMED CT expressions generated when processing an expression template should conform to the SNOMED CT concept model. This can be tested automatically by checking for conformance with the rules defined in the SNOMED CT machine readable concept model (MRCM) in conjunction with a specific SNOMED CT substrate (e.g. the active concepts in the current SNOMED CT international edition). This includes determining the domains that the focus concepts belong to, checking that the attributes are valid for the given domains, checking that the attributes appear an appropriate number of times, and ensuring that the value of each attribute is in the correct range. For more information on using the MRCM to validate postcoordinated expressions please refer to 6. Considerations.

SNOMED International

8. Expression Template Examples

In this chapter, we present a variety of examples to illustrate how the template syntax defined in 5. Syntax Specification can be used to represent *expression templates*. A *SNOMED CT expression template* is a SNOMED CT expression, which contains one or more template slots. As defined in 5. Syntax Specification, all template slots are represented using a pair of double square brackets - i.e. [[ ]].

There are two main types of template slots:

1. *Replacement Slots*, which are replaced by a concept, expression or string during template processing, and
2. *Information Slots*, which are purely there to provide metadata about how the template is to be processed.

The following pages present a range of examples of how these two types of slots are used within expression templates:

8.1. Simple Replacement Slots

Replacement slots serve as a placeholder for a value that is not known at the time of authoring, but which can be completed at a subsequent time using data recorded elsewhere (such as in an information model or entered into a data entry form). A slot indicates that it may be replaced by a value by including a plus sign (+) as the first symbol within the slot. Replacement slots may be used in an expression template wherever a concept, expression or symbol may appear within an expression.

Focus concept

A replacement slot may be used to delay the selection of a focus concept in an expression. For example, the expression template below uses a slot to indicate that the focus concept in the expression can be replaced with any appropriate SNOMED CT expression. This expression template represents any expression with a single refinement in which the |Laterality| equals |Right|.

```
[[+]]: 272741003 |Laterality| = 24028007 |Right|
```

If the following values are provided to complete the slot:

```
182245002 |Entire upper limb|
182281004 |Entire lower limb|
244486005 |Entire eye|
1910005 |Entire ear|
```

then the expressions that result from processing the template would be:

```
182245002 |Entire upper limb| : 272741003 |Laterality| = 24028007 |Right|
182281004 |Entire lower limb| : 272741003 |Laterality| = 24028007 |Right|
244486005 |Entire eye| : 272741003 |Laterality| = 24028007 |Right|
1910005 |Entire ear| : 272741003 |Laterality| = 24028007 |Right|
```

Attribute Value

A replacement slot may also be used to replace an attribute value in an expression. For example, the expression template below represents any expression with a focus concept of |Clinical finding| and a refinement which uses

© Copyright 2020 International Health Terminology Standards Development Organisation

the attribute |Finding site| .The value of the |Finding site| attribute may be replaced by any valid value at a future time.

```
404684003 |Clinical finding| : 363698007 |Finding site| = [[+]]
```

If the following value is provided to complete the slot:

```
53120007 |Upper limb structure|
```

then the expression that results from processing the template would be:

```
404684003 |Clinical finding| : 363698007 |Finding site| = 53120007 |Upper limb structure|
```

If, however, the value provided to complete the slot was the expression:

```
53120007 |Upper limb structure| : 272741003 |Laterality| = 7771000 |Left|
```

then the result of processing the template would be:

```
404684003 |Clinical finding| : 363698007 |Finding site| = ( 53120007 |Upper limb structure| : 272741003 |
Laterality| = 7771000 |Left| )
```

Please note that the template processor must add round brackets around the subexpression to ensure that the resulting expression is syntactically valid.

Attribute Name

A replacement slot may also be used to replace an attribute name in an expression. For example, the expression template below represents any expression with a focus concept of |Clinical finding| and a refinement whose value equals |Rheumatic fever|The attribute name in the expression may be replaced by any valid attribute concept at a future time.

```
404684003 |Clinical finding| : [[+]] = 80166006 |Streptococcus pyogenes|
```

If the following values are provided to complete the slot:

```
42752001 |Due to|
255234002 |After|
```

then the expressions that result from processing the template would be:

```
404684003 |Clinical finding| : 42752001 |Due to| = 80166006 |Streptococcus pyogenes|
404684003 |Clinical finding| : 255234002 |After| = 80166006 |Streptococcus pyogenes|
```

8.2. Typed Replacement Slots

Overview

Replacement slots may be given a *replacement type* to indicate what type of value may be used to replace the slot. The *replacement type* directly follows the '+' symbol inside the slot (with optional whitespace between). Permitted *replacement types* include:

- **id**: The slot may be replaced by a single SNOMED CT concept reference.
- **scg**: The slot may be replaced by a SNOMED CT compositional grammar expression (either a precoordinated or postcoordinated expression).
- **tok**: The slot may be replaced by a token (or symbol) that is defined in the base syntax.
- **str**: The slot may be replaced by a quoted string of characters. This is used to represent a concrete attribute value that is typed as a string.
- **int**: The slot may be replaced by a integer (preceded by a '#' symbol). This is used to represent a concrete attribute value that is typed as an integer.
- **dec**: The slot may be replaced by a decimal (preceded by a '#' symbol). This is used to represent a concrete attribute value that is typed as a decimal.
- **bool**: The slot may be replaced by a boolean. This is used to represent a concrete attribute value that is typed as a boolean.

Please note, that if no *replacement type* is specified after the '+' symbol, then a *replacement type* of '**scg**' is assumed.

Concept Replacement Slots

Slots with a replacement type of '**id**' may only be replaced by a single concept reference. For example, the replacement slot in the following expression template

```
404684003 |Clinical finding| : 255234002 |After| = [[+id]]
```

may be replaced by the concept 82271004 |Injury of head| to form the expression

```
404684003 |Clinical finding| : 255234002 |After| = 82271004 |Injury of head|
```

However, it is not possible to generate a nested expression using this expression template, as the slot may not be replaced by a postcoordinated expression.

Expression Replacement Slots

Slots with a replacement type of '**scg**' may be replaced by any (precoordinated or postcoordinated) expression. For example, the slot in the following expression template

```
404684003 |Clinical finding| : 255234002 |After| = [[+scg]]
```

may be replaced by any of the following values

```
82271004 |Injury of head|
417163006 |Injury| : 363698007 |Finding site| = 69536005 |Head structure|
417163006 |Injury| + 118934005 |Disorder of head|
```

to generate the following expressions.

```
404684003 |Clinical finding| : 255234002 |After| = 82271004 |Injury of head|
404684003 |Clinical finding| : 255234002 |After| = ( 417163006 |Injury| : 363698007 |Finding site| =
69536005 |Head structure| )
404684003 |Clinical finding| : 255234002 |After| = ( 417163006 |Injury| + 118934005 |Disorder of head| )
```

Please note that for the second and third replacements, the template processor must add round brackets around the subexpression to ensure that the resulting expression is syntactically valid.

Token Replacement Slots

Slots with a replacement type of '**tok**' may be replaced by any token (or symbol) that is defined in the base language. For example, the following expression has a definition status that has not yet been defined.

```
[[+tok]] 73211009 |Diabetes mellitus| : 363698007 |Finding site| = 113331007 |Endocrine system|
```

If the definitionStatus |'<<<'|is used to complete the slot then the resulting expression would be:

```
<<< 73211009 |Diabetes mellitus| : 363698007 |Finding site| = 113331007 |Endocrine system|
```

Concrete Value Replacement Slots

Slots with a replacement type of '**str**', '**int**', '**dec**' or '**bool**' may be replaced by any string, integer, decimal or boolean value respectively. For example, the slot in the expression template below:[1]

```
322236009 |Paracetamol 500mg tablet| : 209999999104 |Has trade name| = [[+str]]
```

may be replaced by the string PANADOL to form the following expression

```
322236009 |Paracetamol 500mg tablet| : 209999999104 |Has trade name| = "PANADOL"
```

Please note that the template processor must add quotation marks around the string to ensure that the resulting expression is syntactically valid.

The slot in the following expression template:

```
323510009 |Amoxycillin 500mg capsule| :
  { 749999999108 |Has pack size magnitude|  = [[+int]],
     759999999106 |Has pack size units|  =  428641000 |Capsule|  }
```

may be replaced by the integer value 30 to form the expression

```
323510009 |Amoxycillin 500mg capsule| :
  { 749999999108 |Has pack size magnitude|  = #30,
     759999999106 |Has pack size units|  =  428641000 |Capsule|  }
```

Please note that the template processor must add a hash symbol ("#") before the integer value to ensure that the resulting expression is syntactically valid.

The slot in the following expression template:

```
326645001 |Chlorhexidine gluconate 0.02% irrigation solution| :
  { 749999999108 |Has pack size magnitude|  = [[+dec]],
     759999999106 |Has pack size units|  =  258770004 |Liter|  }
```

may be replaced by the decimal value 1.5 to form the expression

```
326645001 |Chlorhexidine gluconate 0.02% irrigation solution| :
  { 749999999108 |Has pack size magnitude|  = #1.5,
     759999999106 |Has pack size units|  =  258770004 |Liter|  }
```

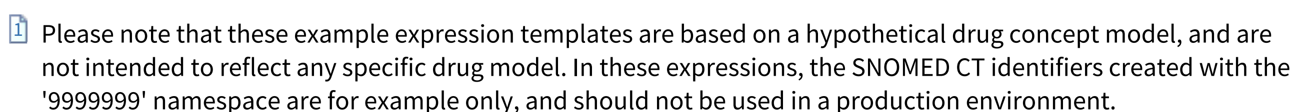
Please note that the template processor must add a hash symbol ("#") before the decimal value to ensure that the resulting expression is syntactically valid.

And lastly, the slot in the following expression template:

```
318969005 |Irbesartan 150 mg oral tablet| :  859999999102 |Is in national benefit scheme|  = [[+bool]]
```

may be replaced by the boolean value TRUE to form the expression

```
318969005 |Irbesartan 150 mg oral tablet| :  859999999102 |Is in national benefit scheme|  = TRUE
```



Please note that these example expression templates are based on a hypothetical drug concept model, and are not intended to reflect any specific drug model. In these expressions, the SNOMED CT identifiers created with the '9999999' namespace are for example only, and should not be used in a production environment.

© Copyright 2020 International Health Terminology Standards Development Organisation

8.3. Constrained Replacement Slots

Overview

The value that can be used to replace a slot in an expression template may be constrained by an expression constraint, value list constraint or range constraint (depending on the type of replacement slot). The following types of slot constraint may be used:

- Expression constraints may be used to constrain replacement slots of type **id** and **scg**,
- Value list constraints may be used to constrain replacement slots of type **tok**, **str**, **int, dec** and **bool.**
- Range constraints may be used to constrain replacement slots of type **int** and **dec**.

The following sections provide examples of each of these types of replacement constraints.

Expression Constraints

Expression constraints may be used to constrain the values permitted to replace a slot of type **id** or **scg** in an expression template.

For example, the slot in the following expression template can only be replaced by a concept reference that is a descendant or self of |Anatomical or acquired body structure|.

```
71388002 |Procedure| :
 { 260686004 |Method| = 312251004 |Computed tomography imaging action| ,
   405813007 |Procedure site - Direct| =
     [[+id (<< 442083009 |Anatomical or acquired body structure| )]] }
```

Because the concept |Shoulder region structure| is a descendant or self of |Anatomical or acquired body structure|, it may be used to replace the slot in the above expression template, resulting in the following expression.

```
71388002 |Procedure| :
 { 260686004 |Method| = 312251004 |Computed tomography imaging action| ,
   405813007 |Procedure site - Direct| = 16982005 |Shoulder region structure| }
```

However, the concept |Nonspecific site| may not be used as a replacement (because it is not a descendant or self of |Anatomical or acquired body structure|).

Similarly, in the following expression template, the slot can be replaced by any expression that is a descendant of |Anatomical or acquired body structure|.

```
71388002 |Procedure| :
 { 260686004 |Method| = 312251004 |Computed tomography imaging action| ,
   405813007 |Procedure site - Direct| =
     [[+scg (<< 442083009 |Anatomical or acquired body structure| )]] }
```

Value List Constraints

Value list constraints can be used to constrain the possible values that may replace slots of type **tok**, **str**, **int**, **dec** and **bool**.

For example, the following expression template uses a value list constraint to specify the possible definitionStatus tokens that may be used. The slot in this expression template may be replaced with either the token "<<<" or the token "===".

```
[[+tok (<<< ===)]]  281647001 |Adverse reaction (disorder)| :
    246075003 |Causative agent (attribute)|  = [[+id]]
```

If the causative agent is assigned the value |Amoxicillin| then the following two expressions can be generated (depending on the token selected to replace the first slot).

```
<<< 281647001 |Adverse reaction (disorder)| : 246075003 |Causative agent (attribute)|  = 372687004 |
Amoxicillin|
```

```
=== 281647001 |Adverse reaction (disorder)| : 246075003 |Causative agent (attribute)|  = 372687004 |
Amoxicillin|
```

The expression template below uses a value list constraint to constrain the value of the slot to the string "PANADOL", the string "TYLENOL", or the string “HERRON”.[1]

```
322236009 |Paracetamol 500mg tablet| : 209999999104 |Has trade name|
= [[+str ("PANADOL" "TYLENOL" "HERRON")]]
```

Depending on the value used to replace the slot, any of the following three expressions may be generated.

```
322236009 |Paracetamol 500mg tablet| : 209999999104 |Has trade name|  = "PANADOL"
```

```
322236009 |Paracetamol 500mg tablet|  : 209999999104 |Has trade name|  = "TYLENOL"
```

```
322236009 |Paracetamol 500mg tablet| : 209999999104 |Has trade name|  = "HERRON"
```

In the next example, a value constraint is used to constrain a replacement slot of type **int**. In this case, the slot may be replaced by the integer values 10, 20 or 30.

```
323510009 |Amoxycillin 500mg capsule| :
{ 749999999108 |Has pack size magnitude|  = [[+int (#10 #20 #30)]],
    759999999106 |Has pack size units|  =  428641000 |Capsule|  }
```

The possible expressions that may be generated by filling the slot are:

© Copyright 2020 International Health Terminology Standards Development Organisation

```
323510009 |Amoxycillin 500mg capsule| :
 { 749999999108 |Has pack size magnitude| = #10,
   759999999106 |Has pack size units| = 428641000 |Capsule| }
```

```
323510009 |Amoxycillin 500mg capsule| :
 { 749999999108 |Has pack size magnitude| = #20,
   759999999106 |Has pack size units| = 428641000 |Capsule| }
```

```
323510009 |Amoxycillin 500mg capsule| :
 { 749999999108 |Has pack size magnitude| = #30,
   759999999106 |Has pack size units| = 428641000 |Capsule| }
```

Range Constraints

Range constraints can be used by slots of type **int** or **dec** to constrain the permitted replacement values. Range constraints may specify the minimum permitted value, the maximum permitted value or both. By default, ranges are inclusive (that is, the range includes the stated minimum and maximum values). However exclusive ranges (in which the stated minimum or maximum is not itself allows as a value) may be specified.

For example, the following expression template allows any pack size to be used that is between 20 and 30 capsules (inclusive).[1]

```
323510009 |Amoxycillin 500mg capsule| :
 { 749999999108 |Has pack size magnitude| = [[+int (#20..#30)]],
   759999999106 |Has pack size units| = 428641000 |Capsule| }
```

To specify exclusive minimum values, a greater than sign ('>') is placed before the minimum value, and to specify exclusive maximum values, a less than sign ('<') is placed before the maximum value. For example, the following expression template allows any pack size that is between 20 and 30 capsules (exclusive). This range constraint is equivalent to an inclusive range of 21..29.

```
323510009 |Amoxycillin 500mg capsule| :
 { 749999999108 |Has pack size magnitude| = [[+int (>#20..<#30)]],
   759999999106 |Has pack size units| = 428641000 |Capsule| }
```

Multiple ranges may be specified in a single range constraint. When multiple ranges are specified, these should be interpreted as alternative ranges. For example, the expression template below permits any pack size between 10 and 20 capsules (inclusive), or between 30 and 40 capsules (inclusive). This template does not, for example, permit a pack size between 21 and 29 capsules.

© Copyright 2020 International Health Terminology Standards Development Organisation

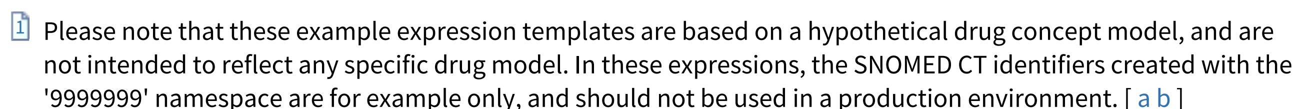
```
323510009 |Amoxycillin 500mg capsule| :
 { 749999999108 |Has pack size magnitude| = [[+int (#10..#20 #30..#40)]],
   759999999106 |Has pack size units| = 428641000 |Capsule| }
```

When a minimum value is required with no maximum, it is possible to omit the maximum value in a range constraint. For example, the expression template below permits any pack size greater than or equal to 20 capsules.

```
323510009 |Amoxycillin 500mg capsule| :
 { 749999999108 |Has pack size magnitude| = [[+int (#20..)]],
   759999999106 |Has pack size units| = 428641000 |Capsule| }
```

Similarly, it is also possible to state a maximum without a minimum constraint. For example, the expression template below permits any pack size less than or equal to 20 capsules. It should be noted that without a concept model rule to ensure that all pack sizes are greater than zero, this template will, by default, allow a pack size of zero or a negative pack size to be defined.

```
323510009 |Amoxycillin 500mg capsule| :
 { 749999999108 |Has pack size magnitude| = [[+int (..#20)]],
   759999999106 |Has pack size units| = 428641000 |Capsule| }
```


Please note that these example expression templates are based on a hypothetical drug concept model, and are not intended to reflect any specific drug model. In these expressions, the SNOMED CT identifiers created with the '9999999' namespace are for example only, and should not be used in a production environment. [a b]

8.4. Named Replacement Slots

In addition to a type and a constraint, replacement slots may also be given a name. Slot names are explained in more detail below.

Slot Names

Replacement slots may be given a slot name, to allow them to be referenced from outside the slot. There are a variety of reasons to do this, including assigning a value to each slot and creating a co-dependency constraint between slots. The most common use of slot names is to refer to the slot to which a value is assigned, during the process of populating the template.

Slot names are defined within the slot using an '@' prefix. For example, the following expression template includes one slot that represents the value of |Associated with|. This slot has been given the slot name "finding".

```
243796009 |Situation with explicit context| :
 { 246090004 |Associated finding| = [[+id (< 404684003 |Clinical finding| ) @finding]],
   40873100 |Temporal context| = 410511007 |Current or past (actual)| ,
   408729009 |Finding context| = 410515003 |Known present| ,
   408732007 |Subject relationship context| = 444148008 |Person in family of subject| }
```

If a segment of code, a data file or a document assigns the value 56265001 |Heart disease| to the finding slot (e.g. finding = 56265001 |Heart disease|), then the following expression can be created.

© Copyright 2020 International Health Terminology Standards Development Organisation

```
243796009 |Situation with explicit context| :
    { 246090004 |Associated finding| = 56265001 |Heart disease| ,
      40873100 |Temporal context| = 410511007 |Current or past (actual)| ,
      408729009 |Finding context| = 410515003 |Known present| ,
      408732007 |Subject relationship context| = 444148008 |Person in family of subject| }
```

Repeated Slot Names

Slot names may be repeated within a template. When the same slot name is associated with more than one slot in the same template, it indicates that these slots must be populated with the same value.

For example, the expression template below has two slots, which have each been given the name "site".

```
  404684003 |Finding| :
    { 363698007 |Finding site| = [[+ @site]],
      363714003 |Interprets| = ( 363787002 |Observable entity| : 704319004 |Inheres in| = [[+ @site]]) }
```

If we assign the value |Liver structure| to the slots named "site" (i.e. site = |Liver structure|), the below expression would be created.

```
  404684003 |Finding| :
    { 363698007 |Finding site| = 10200004 |Liver structure| ,
      363714003 |Interprets| = ( 363787002 |Observable entity| : 704319004 |Inheres in| = 10200004 |Liver
structure| ) }
```

For additional uses of slot names, please refer to 8.6. Advanced Expression Templates.

8.5. Information Slots

Overview

Unlike replacement slots, information slots are added to an expression template purely to provide metadata about how the template should be processed. When an expression template is processed, the information contained in the information slots is used and then the information slot itself is removed.

Information slots may be added to an expression template in one of three places:

1. **Before a focus concept slot** to indicate information about the focus concept that follows,
2. **Before a relationship group** to indicate information about the relationship group that follows, and
3. **Before an attribute** to indicate information about the attribute name-value pair that follows.

Information slots may include either a cardinality constraint or a slot name (or both). In the following sections, we describe how each of these may be used in an information slot.

Cardinality

One of the important roles of an information slot is to specify the cardinality of the expression part that follows. A cardinality constraint states the minimum and maximum number of times that the focus concept, relationship group or attribute name-value pair that follows may be repeated in an expression that is generated from the given template.

© Copyright 2020 International Health Terminology Standards Development Organisation

For example, the expression template below allows between one and three concepts to be used to populate the *finding* slot, and exactly one |Finding site| name-value pair.

```
[[1..3]] [[+id (< 404684003 |Clinical finding| : [0..0] 363698007 |Finding site| = * ) @finding]]:
   [[1..1]] 363698007 |Finding site| =
     [[+id (<< 442083009 |Anatomical or acquired body structure| ) @site ]]
```

Based on the cardinality specified in the information slots, the *finding* slot could be replaced by the two concepts |Infectious disease| and |Congenital disease| and the *site* slot could be replaced by the concept |Lung structure| to generate the following expression:

```
40733004 |Infectious disease| + 66091009 |Congenital disease| : 363698007 |Finding site| = 39607008 |
Lung structure|
```

Please note that when two or more concepts are used to replace a focus concept slot in an expression template, a plus sign (+) must be added between them to ensure the resulting expression is syntactically valid.

Default Cardinality

When a cardinality constraint is not provided in an information slot preceding a focus concept, relationship group or attribute name-value pair, the default cardinality that is assumed is 1..*. This means that by default focus concepts, relationship groups and attribute name-value pairs must appear at least once in the resulting expression, but may also be repeated many times. The SNOMED CT concept model, if enforced, may also impose some additional cardinality constraints on the number of times an attribute may be repeated. However, these additional concept model rules are not directly implied by the expression template itself.

As an example, the following expression template allows one or more procedures to be collectively refined by one or more relationship groups, each containing one or more values for |Method| and |Procedure site - Direct|.

```
[[+id (<< 71388002 |Procedure| )]]:
  { 260686004 |Method| = [[+id (<< 129264002 |Action (qualifier value)| )]],
    405813007 |Procedure site - Direct| = [[+id (<< 442083009 |Anatomical or acquired body structure
(body structure)| )]] }
```

The above expression template is therefore equivalent to the following template, in which the cardinality of 1..* is explicitly stated on the focus concept, the relationship group, and each attribute name-value pair.

```
[[1..*]] [[+id (<< 71388002 |Procedure| )]]:
  [[1..*]] { [[1..*]] 260686004 |Method| = [[+id (<< 129264002 |Action (qualifier value)| )]],
        [[1..*]] 405813007 |Procedure site - Direct| = [[+id (<< 442083009 |Anatomical or acquired body
structure (body structure)| )]] }
```

The replacement slots in this expression template could, for example, be replaced to generate the following expression:

© Copyright 2020 International Health Terminology Standards Development Organisation

```
76193006 |Routinely scheduled operation| + 387713003 |Surgical procedure| :
  { 260686004 |Method| = 281615006 |Exploration| ,
    260686004 |Method| = 312250003 |Magnetic resonance imaging - action| ,
    405813007 |Procedure site - Direct| = 28273000 |Bile duct structure| },
  { 260686004 |Method| = 129304002 |Excision| ,
    405813007 |Procedure site - Direct| = 28231008 |Gallbladder structure| }
```

Please note that if the international SNOMED CT concept model was enforced, the above expression would not be valid due to the 260686004 |Method| appearing twice in the one relationship group.

Slot Name

Information slots can also be given a slot name, to allow the expression part that follows the slot (i.e. focus concept, relationship group, or attribute name-value pair) to be referenced. The most common use of information slot names is to support the process of populating the template (e.g. as part of a programmatic value replacement process).

For example, the following expression template uses the name *mpGroup* to name the relationship group that contains both a |Method| and a |Procedure site - Direct|

```
71388002 |Procedure| : [[1..1 @mpGroup]]
  { 260686004 |Method| = 312251004 |Computed tomography imaging action| ,
    405813007 |Procedure site - Direct| = [[+id (<< 442083009 |Anatomical or acquired body structure| ) @si
te]] }
```

8.6. Advanced Expression Templates

Overview

In addition to the simple examples shown on the previous pages, more advanced expression templates can also be used. On this page we show some examples of expression constraints with multiple replacement slots, multiple cardinality constraints and slot co-dependency constraints.

Multiple Replacement Slots

In many situations, it is useful for an expression template to contain more than one replacement slot. Below are some examples.

Example 1

The following expression template uses three replacement slots to generate an expression that represents a type of procedure. The first slot (named "Procedure") is a placeholder for the focus concept, while the second slot (named "BodySite") is a placeholder for the |Procedure site - Direct|, and the third slot (named "Method") is a placeholder for the |Method|.

```
[[+ (< 71388002 |Procedure| ) @Procedure]]:
  { 405813007 |Procedure site - direct| = [[+ (< 91723000 |Anatomical structure| ) @BodySite]],
    260686004 |Method| = [[+ (< 129264002 |Action (qualifier value)| ) @Method]] }
```

If the slots are populated with the values:

- Procedure = 387713003 |Surgical procedure|
- BodySite = 66754008 |Appendix structure|
- Method = 129304002 |Excision - action|

then the following expression would be generated.

```
387713003 |Surgical procedure| :
  { 405813007 |Procedure site - direct| = 66754008 |Appendix structure| ,
    260686004 |Method| = 129304002 |Excision - action| }
```

Example 2

Another example of an expression template with multiple slots is shown below. This expression template is used to generate expressions that represent the family history of a patient. The template contains two slots - the first slot (named "Finding") is a placeholder for the |Clinical finding| known to be present in the family member, while the second slot (named "Relationship") is a placeholder for the |Subject relationship context| of this |Clinical finding|.

```
243796009 |Situation with explicit context| :
  { 246090004 |Associated finding| = [[+id (< 404684003 |Clinical finding| ) @Finding]],
    408731000 |Temporal context| = 410511007 |Current or past (actual)| ,
    408729009 |Finding context| = 410515003 |Known present| ,
    408732007 |Subject relationship context| = [[+id (<< 444148008 |Person in family of subject| ) @Relati
onship]] }
```

If the following input data is provided (in which each row represents a separate expression):

Finding	Relationship
93870000 |Liver cancer|	444244000 |Maternal grandmother of subject|
57809008 |Myocardial disease|	444292000 |Paternal grandfather of subject|
46635009 |Diabetes mellitus type 1|	444301002 |Mother of subject|

Then the following family history expressions would be generated.

```
243796009 |Situation with explicit context| :
  { 246090004 |Associated finding| = 93870000 |Liver cancer| ,
    408731000 |Temporal context| = 410511007 |Current or past (actual)| ,
    408729009 |Finding context| = 410515003 |Known present| ,
    408732007 |Subject relationship context| = 444244000 |Maternal grandmother of subject| }
```

```
243796009 |Situation with explicit context| :
  { 246090004 |Associated finding| = 57809008 |Myocardial disease| ,
    408731000 |Temporal context| = 410511007 |Current or past (actual)| ,
```

© Copyright 2020 International Health Terminology Standards Development Organisation

```
    408729009 |Finding context| = 410515003 |Known present| ,
    408732007 |Subject relationship context| = 444292000 |Paternal grandfather of subject| }
```

```
243796009 |Situation with explicit context| :
 { 246090004 |Associated finding| = 46635009 |Diabetes mellitus type 1| ,
   408731000 |Temporal context| = 410511007 |Current or past (actual)| ,
   408729009 |Finding context| = 410515003 |Known present| ,
   408732007 |Subject relationship context| = 444301002 |Mother of subject| }
```

Multiple Cardinality Constraints

Expression templates may also use more than one cardinality constraint to indicate the repeatability of different parts of the expression. Below are some examples.

Example 1

The following expression template uses three replacement slots to generate an expression that represents a type of procedure. The first slot (named "Procedure") is a placeholder for the focus concept, while the second slot (named "BodySite") is a placeholder for the |Procedure site - Direct|, and the third slot (named "Method") is a placeholder for the |Method|.

```
[[1..1]] [[+ (< 71388002 |Procedure| ) @Procedure]] :
   [[ 1..2 @SMgroup]] { [[1..1]] 405813007 |Procedure site - direct| = [[+ (< 91723000 |Anatomical structure|
) @BodySite]],
                 [[1..1]] 260686004 |Method| = [[+ (< 129264002 |Action (qualifier value)| ) @Method]] }
```

The first cardinality constraint (i.e. 1..1) indicates that exactly one value should be populated in the *Procedure* slot. The second cardinality constraint (i.e. 1..2) indicates that it is valid to include either 1 or 2 relationship groups in the resulting expression. The last two cardinality constraints, that appear inside the relationship group, (i.e. 1..1) indicate that in each instance of a relationship group, exactly one |Procedure site - direct| value and exactly one |Method| value should be used.

The following input data satisfies these cardinality constraints.

Procedure	SMgroup	BodySite	Method
387713003 |Surgical procedure|	1	28273000 |Bile duct structure|	281615006 |Exploration - action|
	2	28231008 |Gallbladder structure|	129304002 |Excision - action|
387713003 |Surgical procedure|	1	66754008 |Appendix structure|	129304002 |Excision - action|

If the slots are populated with the values above, then the following expressions would be generated.

```
387713003 |Surgical procedure| :
  { 405813007 |Procedure site - direct| = 28273000 |Bile duct structure| , 260686004 |Method| = 281615006
|Exploration - action| },
```

© Copyright 2020 International Health Terminology Standards Development Organisation

```
    { 405813007 |Procedure site - direct| = 28231008 |Gallbladder structure| , 260686004 |Method| =
129304002 |Excision - action| }
```

```
387713003 |Surgical procedure| :
    { 405813007 |Procedure site - direct| = 66754008 |Appendix structure| , 260686004 |Method| =
129304002 |Excision - action| }
```

© Copyright 2020 International Health Terminology Standards Development Organisation

Example 2

Another example of an expression template with multiple cardinality constraints is shown below. This expression template is used to generate expressions that represent a clinical finding with explicit context.

```
[[1..1]] [[+id (<< 413350009 |Finding with explicit context| ) @Condition]]:
   [[ 1..2 @AFgroup ]] { [[1..1]] 246090004 |Associated finding| = ( [[+id (<< 404684003 |Clinical finding|
) @Finding]]:
          [[0..1 @SSgroup]] { [[0..1]] 246112005 |Severity| = [[+id (< 272141005 |Severities| ) @Severity]],
                    [[0..1]] 363698007 |Finding site| = [[+id (< 91723000 |Anatomical structure| ) @Site]] } ),
              [[1..1]] 408732007 |Subject relationship context| = [[+id (< 444148008 |Person in family of
subject| ) @Relationship]],
              [[1..1]] 408731000 |Temporal context| = [[+id (< 410510008 |Temporal context value|
) @Time]],
              [[1..1]] 408729009 |Finding context| = [[+id (< 410514004 |Finding context value| ) @Context]] }
```

The first cardinality constraint (i.e. 1..1) indicates that exactly one value should be populated in the *Condition* slot. The second cardinality constraint (i.e. 1..2) indicates that it is valid to include either 1 or 2 relationship groups (named "AFgroup") in the resulting expression. Each *AFgroup* relationship group must have exactly one |Associated finding|, exactly one |Subject relationship context|, exactly one |Temporal context| and exactly one |Finding context| . The value of the |Associated finding| in each *AFgroup* is an expression, may optionally be refined using a single relationship group named "SSgroup". Each *SSgroup* optionally has one |Severity| and optionally has one |Finding site|. Based on these cardinality constraints, the input data shown in Table 8.6-1 would be valid.

Table 8.6-1: Valid Input Data for Example 2

<table>
<tr><th>Condition</th><th>AF group</th><th>Finding</th><th>Severity</th><th>Site</th><th>Relationship</th><th>Time</th><th>Context</th></tr>
<tr><td rowspan="2">243796009
|Situation with explicit context|</td><td>1</td><td>56265001
|Heart disease|</td><td>24484000
|Severe|</td><td></td><td>444292000
|Paternal grandfather of subject|</td><td>410512000
|Current or specified time|</td><td>410515003
|Known present|</td></tr>
<tr><td>2</td><td>22298006
|Myocardial infarction|</td><td></td><td></td><td>444292000
|Paternal grandfather of subject|</td><td>410589000
|All times past|</td><td>410516002
|Known absent|</td></tr>
<tr><td>57177007
|Family history with explicit context|</td><td>1</td><td>363346000
|Cancer|</td><td>6736007
|Moderate|</td><td>76752008
|Breast structure|</td><td>444244000
|Maternal grandmother of subject|</td><td>410512000
|Current or specified time|</td><td>410515003
|Known present|</td></tr>
<tr><td>160303001
|FH: Diabetes mellitus|</td><td>1</td><td>46635009
|Diabetes mellitus type 1|</td><td></td><td></td><td>444301002
|Mother of subject|</td><td>410512000
|Current or specified time|</td><td>410515003
|Known present|</td></tr>
</table>

If the slots are populated with the values above, then the following expressions would be generated.

```
243796009 |Situation with explicit context| :
  { 246090004 |Associated finding| = ( 56265001 |Heart disease| : { 246112005 |Severity| = 24484000 |Severe
| }),
     408732007 |Subject relationship context| = 444292000 |Paternal grandfather of subject| ,
     408731000 |Temporal context| = 410511007 |Current or past (actual)| ,
     408729009 |Finding context| = 410515003 |Known present| },
  { 246090004 |Associated finding| = 22298006 |Myocardial infarction| ,
     408732007 |Subject relationship context| = 444292000 |Paternal grandfather of subject| ,
     408731000 |Temporal context| = 410589000 |All times past| ,
     408729009 |Finding context| = 410516002 |Known absent| }
```

```
57177007 |Family history with explicit context| :
  { 246090004 |Associated finding| = ( 363346000 |Cancer| :
       { 246112005 |Severity| = 6736007 |Moderate| , 363698007 |Finding site| = 76752008 |Breast structure
| } ),
     408732007 |Subject relationship context| = 444244000 |Maternal grandmother of subject| ,
     40873100 |Temporal context| = 410511007 |Current or past (actual)| ,
     408729009 |Finding context| = 410515003 |Known present| }
```

```
160303001 |FH: Diabetes mellitus| :
  { 246090004 |Associated finding| = 46635009 |Diabetes mellitus type 1| ,
     40873100 |Temporal context| = 410511007 |Current or past (actual)| ,
     408729009 |Finding context| = 410515003 |Known present| ,
     408732007 |Subject relationship context| = 444301002 |Mother of subject| }
```

Please note that when part of an expression is repeated, connectors (e.g. a comma) must be added between the parts during processing. Similarly, when part of an expression is absent then clean up is required (e.g. to remove commas and brackets). For more information on these processing steps, please refer to 7.2. Template Processing.

© Copyright 2020 International Health Terminology Standards Development Organisation