Leading healthcare terminology, worldwide

SNOMED CT Expression Constraint Language Specification and Guide

Version 2.0

Publication date: 2022-04-03

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

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.

Leading healthcare
terminology, worldwide

Table of Contents

1. Introduction 2
2. Use Cases 5
2.1 Terminology Binding 5
2.2 Intensional Reference Set Definitions 5
2.3 SNOMED CT Content Queries 5
2.4 SNOMED CT Concept Model 6
3. Requirements 7
3.1 General SNOMED CT Language Requirements 7
3.2 Expression Constraint and Query Requirements 7
3.3 Concept Model Requirements 9
4. Logical Model 10
4.1 Details 11
5. Syntax Specification 13
5.1 Brief Syntax (Normative) 13
5.2 Long Syntax (Informative) 16
5.3 Informative Comments 20
5.4 Order of Operation 34
5.5 Character Collation for Term Filters 37
6. Examples 40
6.1 Simple Expression Constraints 40
6.2 Refinements 46
6.3 Cardinality 53
6.4 Conjunction and Disjunction 58
6.5 Exclusion and Not Equals 63
6.6 Constraint Comments 64
6.7 Nested Expression Constraints 65
6.8 Description Filters 69
6.9 Concept Filters 76
6.10 Member Filters 83
6.11 History Supplements 84
7. Implementation Considerations 89
7.1 Authoring 89
7.2 Parsing 91
7.3 Validating 91

7.4 Executing 91
7.5 Storing 92
7.6 Displaying 92
7.7 Exchanging 93
Appendix A – Examples Of Valid Expressions 94
A.1 Simple Expression Constraints - Valid Expressions 94
A.2 Refinements - Valid Expressions 95
A.3 Cardinality - Valid Expressions 98
A.4 Conjunction and Disjunction - Valid Expressions 100
A.5 Exclusion and Not Equals - Valid Expressions 102
A.6 Nested Expression Constraints - Valid Expressions 103
Appendix B – Examples Of Invalid Expressions 105
B.1 Simple Expression Constraints - Invalid Expressions 105
B.2 Refinements - Invalid Expressions 106
B.3 Cardinality - Invalid Expressions 109
B.4 Conjunction and Disjunction - Invalid Expressions 111
B.5 Exclusion and Not Equals - Invalid Expressions 112
B.6 Nested Expression Constraints - Invalid Expressions 114
Appendix C - Dialect Aliases 116
Appendix D - ECL Quick Reference 118
Syntax Overview 118
Examples 119
Appendix E - Reference Set Fields 123
References 126

The *Expression Constraint Language* is a formal syntax for representing SNOMED CT expression constraints. Expression constraints are computable rules used to define a bounded sets of clinical meanings represented by either precoordinated or postcoordinated expressions. Expression constraints can be used to restrict the valid values for a data element in an EHR, as the intensional definition of a concept-based reference set, as a machine processable query that identifies a set of matching expressions, or as a constraint that restricts the range of an attribute defined in the SNOMED CT concept model.

This document defines and describes the current version of the Expression Constraint Language - ECL 2.0.

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

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

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 expressions are a structured combination of one or more concept identifiers used to represent a clinical idea in a logical manner. The SNOMED CT Compositional Grammar provides a lightweight syntax for the representation of SNOMED CT expressions.

In contrast, a *SNOMED CT Expression Constraint* is a computable rule that can be used to define *a bounded set of* clinical meanings represented by either precoordinated or postcoordinated expressions. Expression constraints can be used as formal constraints on the content of a particular data element in an EHR, as the intensional definition of a concept-based reference set, as a machine processable query that identifies a set of matching precoordinated or postcoordinated expressions, or as a constraint that restricts the range of an attribute defined in the SNOMED CT concept model.

Purpose

The purpose of this document is to define and describe a formal language for representing SNOMED CT Expression Constraints. A SNOMED CT Expression Constraint is a computable rule that defines a bounded set of clinical meanings represented by either precoordinated or postcoordinated expressions. Two equivalent syntaxes are presented – a brief syntax, which is designed to be as compact as possible for interoperable communication between systems, and a long syntax, which introduces textual alternatives to the symbols from the brief syntax. This document also provides examples and guidance to assist in the implementation of this language.

Scope

This document presents the specification of an Expression Constraint Language, which can be used to represent SNOMED CT Expression Constraints. It includes a logical model of the language, two syntaxes, a set of example expression constraints and a summary of implementation considerations.

The Expression Constraint Language specified in this document is part of a consistent set of computer processable languages designed to support a variety of use cases involving the use of SNOMED CT. Other SNOMED CT computable languages include:

- Compositional Grammar: designed to represent SNOMED CT expressions; and
- Template Syntax: which allow slots to be added to expressions, expression constraints or queries that can be filled with specific values at a later time.

The compositional grammar is designed to provide a common foundation for the additional functionality added by the other languages.

This document does not include a full description of how to implement an expression constraint parser, classifier or interpreter. It does not describe how to transform an expression constraint into other languages, such as OWL, SPARQL or SQL; or how to determine whether two expression constraints are equivalent. It also does not describe how to implement an EHR which uses expression constraints to constrain or query its content, or a terminology server which uses expression constraints to query its content. Instead, it provides a specification, examples and general guidance to assist in the implementation of expression constraints in any of these applications.

This document defines and describes the current version of the Expression Constraint Language - ECL 2.0.

History

Expression constraints have been used in projects and programs around the world for a number of years – for example HL7 TermInfo, and the NHS Logical Record Architecture.

In 2013, a draft document on "SNOMED CT Expression Constraint Syntax Specification for Terminology Binding" was developed as an assignment during the SNOMED CT Implementation Advisor (SIA) scheme.

In 2014, this work was revised and extended to support a wider range of relevant use cases to produce version 1.0 of the Expression Constraint Language specification (2015). These updates included:

- Concrete values (e.g. integers, decimals and strings) are now permitted as attribute values. This is to provide alignment with the recent extensions to SNOMED CT Compositional Grammar;
- Cardinality constraints have been introduced, and as a result the optional operator (i.e. ~) is no longer provided;
- Attributes may now be preceded by a 'descendantOf' or 'descendantOrSelfOf' operator to indicate whether attribute descendants and/or the attribute itself should be used in the matching process;
- A reverse flag has been introduced, which allows relationships to be traversed in the reverse direction;
- Exclusion has been changed from a unary operator ('negation') to a binary operator ('minus');
- A wildcard character ('*') has been introduced to represent any concept in the substrate;
- A number of clarifications have been made, including the 'memberOf' operator and the default substrate upon which the expression constraints are executed.

An update to the Expression Constraint Language was then published in 2016 (version 1.1) to incorporate some additional features requested by implementers of the language. These updates include:

- Two new operators 'childOf' and 'parentOf' were added to support querying immediate children and immediate parents of a concept during user interface design;
- A new 'dot notation' was introduced (as an alternative to the Reverse flag) to refer to an attribute value for a concept or expression;
- The ability for a constraint operator (e.g. 'descendantOf') to be applied to a nested expression constraint was added;
- The ability to add comments within the text of an expression constraint was added;
- Additional optional brackets were allowed around subexpressions; and
- The non-normative syntax (previously named the 'Full Syntax') was renamed to the 'Long Syntax'.

Early in 2017 version 1.2 was published, to include a new feature requested by implementers: namely, the ability for the 'memberOf' function to be applied to a set of reference set concepts defined using an expression constraint. In this version, the explanation of *Operator Precedence* was also moved from section 6.7 to section 5.4. Version 1.3 was then published in mid 2017 to support a range of additional features - including allowing the refinement of subexpression constraints, permitting the use of subexpression constraints to represent a set of valid attribute names and simplifying the parsing of dotted expression constraints.

In mid 2020, version 1.4 was published to support boolean attribute values and to introduce the 'childOrSelfOf' and 'parentOrSelfOf' operators. Later that year, version 1.5 was published to support description filter constraints. These constraints filter the result set, by matching only on concepts which have a description that satisfies the filter criteria. Section 5.5 (Character Collation for Term Filters) and section 6.8 (Filter Constraints) were added in ECL version 1.5.

In 2021, version 1.6 added concept filters, which allow the result set to be filtered based on the definition status, module, effectiveTime and active status of each concept.

And then in early 2022, version 2.0 was published. Version 2.0 includes a number of significant features, including:

- History supplements, to supplement the results with relevant inactive concepts,
- Reference set member filters, to filter the rows of a reference set, based on the value of specified fields,
- Support for returning multiple fields of a reference set, including fields other than the referencedComponentId,
- Support for module, effectiveTime and active filters on descriptions, and
- Support for word-prefix-any-order and wildcard searches for string-based concrete attribute values (for consistency with term searches in a Description filter).

Most significantly, version 2.0 is the first version of ECL that is specifically designed to support querying over historical patient records, which may contain inactive codes.

For a full list of previous versions and a summary of updates, please refer to Previous Versions.

Audience

The target audiences of this document include:

- SNOMED 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 (including the use cases and requirements) and the extensive set of examples that is presented. It should also be noted that even though complex expression constraints are possible, most expression constraints are likely to be very simple, such as those described in Simple Expression Constraints.

Document Overview

This document defines the SNOMED CT Expression Constraint Language and describes how and where it may be implemented. Chapter 2 begins by describing the use cases in which it is anticipated that SNOMED CT Expression Constraint Language will be used. Chapter 3 then describes the requirements used to guide the definition of this language. In Chapter 4, the logical model of the Expression Constraint Language is presented, while in Chapter 5 two syntaxes are defined using an ABNF serialisation of the logical model. Chapter 6 then presents some examples of expression constraints that conform to the SNOMED CT Expression Constraint syntaxes, and Chapter 7 discusses some implementation considerations. Appendix A – Examples Of Valid Expressions provides some examples of precoordinated and postcoordinated expressions that satisfy each of the expression constraints presented earlier in the document. Appendix B – Examples Of Invalid Expressions then provides some examples that do not satisfy these expression constraints. Appendix C - Dialect Aliases provides a list of example aliases that may be used to specify a particular dialect in an ECL filter constraint. Appendix D - ECL Quick Reference provides a quick reference to the key syntax features of the Expression Constraint Language. And finally, Appendix E - Reference Set Fields explains how reference set field names are used in ECL 2.0+.

2. Use Cases

The SNOMED CT Expression Constraint Language enables the intensional definition of a bounded set of clinical meanings. This is important for a number of use cases, including:

- Terminology Binding;
- Intensional Reference Set Definitions;
- SNOMED CT Content Queries; and
- SNOMED CT Concept Model.

In the following subsections, we describe each of these key use cases.

2.1 Terminology Binding

Most Electronic Health Records (EHRs) are designed and developed using one or more information models, which describe the information that is collected, stored, communicated and displayed. Some information models are designed for a specific proprietary system, while others are based on a common health information standard (e.g. HL7 FHIR resource, HL7 CDA template, ISO 13606 archetype). Information models may also be defined using a wide variety of representations (e.g. UML class diagram, database table design, Archetype Definition Language, or XML Schema). Irrespective of the purpose, design and representation of the information models, however, the use of clinical terminology is an important part of making the models complete and useful.

Terminology binding provides the links between the information model and the terminology. These links may be used to constrain the set of possible values which can populate a given coded data element in the information model, or they may define the meaning of an information model artefact using the terminology. Terminology binding is an important part of supporting the following clinical information system functions:

- Data capture;
- Retrieval and querying;
- Information model library management; and
- Semantic interoperability.

To enable terminology binding to be defined using intensional rules, a formal language must be used. The SNOMED CT Expression Constraint Language can be used in this way to define terminology bindings which constrain the set of possible coded values within an information model.

2.2 Intensional Reference Set Definitions

Reference sets are a flexible, extensible SNOMED CT file structure used to support a variety of requirements for the customization and enhancement of SNOMED CT content. These include the representation of subsets, language preferences, or maps to/from other code systems.

Some reference sets (using the Query Specification type) allow a serialised query to represent the membership of a subset of SNOMED CT components. A query contained in this reference set is executed against the content of SNOMED CT to produce a subset of concepts, descriptions or relationships. This query is referred to as an intensional definition of the subset. It can be run against future releases of SNOMED CT to generate a potentially different set of subset members. The members of the resulting subset may also be represented in an enumerated form as a Simple Reference Set. An enumerated representation of a subset is referred to as an extensional definition.

The SNOMED CT Expression Constraint Language can be used in this way to represent the intensional definition of a subset of SNOMED CT concepts that can be enumerated as a Simple Reference Set.

2.3 SNOMED CT Content Queries

SNOMED CT provides both hierarchies and formal concept definitions to allow a range of advanced query techniques. SNOMED CT queries can be performed over different sets of terminology artefacts (known as the substrate of the query), including:

- The precoordinated components distributed as part of the SNOMED CT international edition;
- The precoordinated components distributed by a local release centre as part of a national or local SNOMED CT edition;
- The postcoordinated expressions stored within an expression repository; or
- The SNOMED CT expressions stored within an Electronic Health Record (EHR).

The SNOMED CT Expression Constraint Language enables queries over SNOMED CT content to be expressed. These queries may be performed for a range of purposes, including the authoring and quality assurance of new SNOMED CT content, the design and development of extensional reference sets, and the design and display of SNOMED CT subsets in clinical user interfaces. While the language itself does not support querying over the full EHR content, the SNOMED CT Expression Constraint Language could be embedded within record-based query languages (such as SQL) to represent the terminological aspects of these queries.

2.4 SNOMED CT Concept Model

The SNOMED CT Concept Model is the set of rules that determines the permitted sets of attributes and values that may be applied to particular types of concepts. There are also additional rules on the cardinality and grouping of each type of attribute. The SNOMED CT Concept Model includes the definition of the domain and range of each attribute. The domain is the set of concepts which are permitted to be used as the source of the attribute, while the range is the set of concepts which are permitted to be used as the target of the attribute. For example, the domain of the attribute 363698007 |Finding site| is the descendants and self of 404684003 |Clinical finding|, while the range is the descendants and self of 442083009 |Anatomical or acquired body structure|The SNOMED CT Concept Model rules are represented in a computable form in the SNOMED CT Machine Readable Concept Model.

3. Requirements

In this chapter, we state the requirements of the SNOMED CT Expression Constraint Language. These requirements are grouped into General SNOMED CT Language Requirements (which are shared by all SNOMED CT computable languages), Expression Constraint and Query Requirements, and Concept Model 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 SNOMED International 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 expressions, as well as for the education and training required to author expressions.

3.2 Expression Constraint and Query Requirements

The general expression constraint language requirements include:

Requirement E.1: Able to be evaluated against SNOMED CT content

> Expression constraints must be able to be evaluated against a specific set of SNOMED CT content (referred to as the substrate). When evaluated against a finite set of precoordinated concepts or postcoordinated SNOMED CT expressions, a finite subset of the substrate can be found which satisfies the expression constraint.
>
> Please note that the substrate over which the expression constraint is evaluated is not explicitly defined within the expression constraint, and must therefore be established by some other means. By default, the assumed substrate is the set of active components from the snapshot release (in distribution normal form) of the SNOMED CT versioned edition currently loaded into the given tool.

Requirement E.2: Expression constraint functional requirements

> The expression constraint language must support the following capabilities:

Function	Details
Concept reference	The ability to reference a precoordinated SNOMED CT concept using its identifier and optional human-readable term.

Concept hierarchy	The ability to refer to a set of concepts which is exactly equal to the descendants, descendants and self, ancestors, or ancestors and self of a given concept.
Immediate children and parents	The ability to refer to a set of concepts which are either immediate children or immediate parents of a given concept (based on non-redundant 116680003 |is a| relationships) (with or without the given concept itself).
Conjunction	The ability to connect two expression constraints, attribute groups or attribute sets via a logical AND operator.
Disjunction	The ability to connect two expression constraints, attribute groups or attribute sets via a logical OR operator.
Refinement	The ability to refine (or specialize) the meaning of an expression constraint using one or more attributes values.
Reverse	The ability to constrain the source concepts of a set of relationships, and refer to the destination concepts of these relationships.
Dotted attribute	The ability to refer to the value (or set of values) of an attribute that is included in the definition of a set of concepts.
Attribute group	The ability to group a collection of attributes which operate together as part of a refinement.
Attribute	The ability to specify an attribute name-value pair which further refines the meaning of the matching expressions.
Attribute descendants	The ability to define an attribute which may apply to either the descendants of the given attribute name, or the descendants and self of the given attribute name.
Nesting	The ability to use an expression constraint to represent the valid set of attribute names and/or attribute values.
Concrete values	The ability to use integers, decimals, strings and booleans as attribute values.
Concrete value comparison	The ability to compare the attribute value of the matching expressions with the attribute value in the expression constraint using mathematical comparison operators (e.g. =, <, >, <=, >=, !=).
Member of	The ability to refer to a set of concepts that are referenced by members of a reference set (or set of reference sets).
Reference set field value selection	The ability to return the value of any non-metadata field of a reference set.
Exclusion	The ability to filter out a set of expressions from the result, by either removing expressions whose focus concept is in a specific set, or removing expressions whose attribute value matches a given value.
Any	The ability to refer to any concept in the substrate, without relying on the availability of a single root concept.
Description filter	The ability to filter the result set, based on the properties of each concept's descriptions. Expression constraints should be able to filter the concepts based on whether or not it has a description with a matching term, type, language, membership of a language reference set, and acceptability within that language reference set. Term matching approaches should include wildcard and word-prefix-any-order. Expression constraints should also be able to filter concepts based on the module, effectiveTime and active status of their descriptions.

Concept filter	The ability to filter the result set, based on the properties of each concept. Expression constraints should be able to restrict the definition status, module, effectiveTime and active status of matching concepts.
Member filter	The ability to filter rows of a reference set member, based on the value of specified fields.
History supplements	The ability to include inactive concepts that are associated with any active concept in a given result set, via an historical association reference set.

3.3 Concept Model Requirements

The SNOMED CT concept model requirements include:

Requirement C.1: The ability to express SNOMED CT concept model constraints

The language must support the ability to express SNOMED CT concept model constraints, such that the resulting expression constraint can be used to validate SNOMED CT concept definitions and postcoordinated expressions.

In particular, the language must support the ability to define the domain and cardinality of each attribute in the SNOMED CT concept model, and the range of all concept model **object** attributes (whose range is a set of SNOMED CT concepts). The domain of an attribute is the set of valid source concepts of relationships of that type. In most cases, this will be defined as the descendants and self of a given concept. The range of a concept model object attribute is the set of valid destination concepts of relationships of that type. This will be defined as the set of concepts that match a given expression constraint. The cardinality of an attribute constrains the number of times an active relationship of this type can be added to a concept in the SNOMED CT snapshot release (in necessary normal form). For more information about the SNOMED CT necessary normal form, please refer to 2.5. Generating Necessary Normal Form in the SNOMED CT OWL Guide (http://snomed.org/owl).

Please note that the range of a concept model **data** attribute (whose value is concrete) will be specified using a value list constraint from the SNOMED CT Template Syntax (http://snomed.org/sts).

4. Logical Model

A SNOMED CT Expression Constraint contains either a single focus concept, or a series of focus concepts joined by either conjunction, disjunction or exclusion. Each focus concept in an Expression Constraint is either a concept reference or a wildcard, and is normally preceded by either a constraint operator or a memberOf function. An Expression Constraint may also contain a refinement, which consists of grouped or ungrouped attributes (or both). Each attribute consists of the attribute name (optionally preceded by a cardinality, reverse flag and/or attribute operator) together with the value of the attribute. The attribute name is either a concept reference or a wild card. The attribute value is either an expression constraint or a concrete value (i.e. string, integer, decimal or boolean). Conjunction or disjunction can be applied at a variety of levels, including between expression constraints, refinements, attribute groups, and attributes. An expression constraint can also be followed by a dot and attribute name pair. One or more description filters may be applied to an expression constraint, which can include module, effective time, active status, term, language, type, dialect and acceptability criteria. Similarly, one or more concept filters may be applied to an expression constraint, which can include definition status, module, effective time and active status criteria. Member filters may be applied to results of the memberOf function, and may include module, effective time, active status and specific refset field criteria. Finally, history supplements may be applied, which include an ECL query to specify the set of historical association reference sets to be used.

Figure 1 below illustrates the overall structure of an expression constraint using an abstract representation. Those parts of an expression constraint, which are in common with SNOMED CT Compositional Grammar expressions, are shown with dotted lines to emphasise the new features (using solid lines) in the Expression Constraint Language. Please note that no specific semantics should be attributed to each arrow in this abstract diagram.

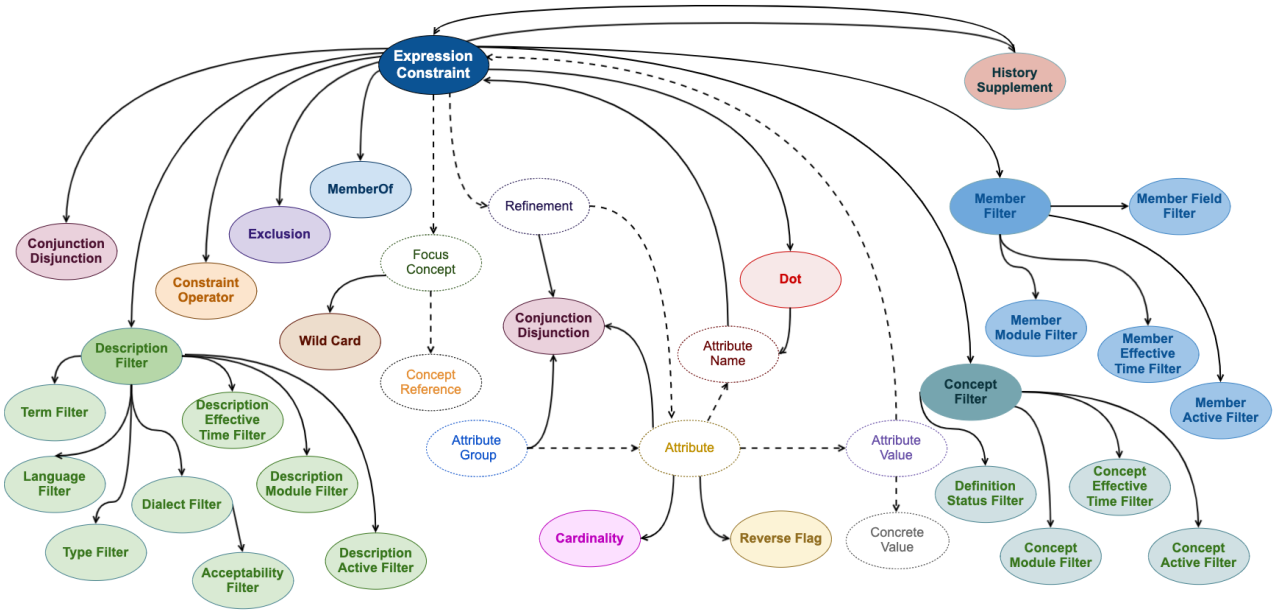

Figure 1: Abstract Model of a SNOMED CT Expression Constraint

Figure 2 below shows an example of an expression constraint[1] with the main components marked. These components will be explained further in the subsequent sections of this document.

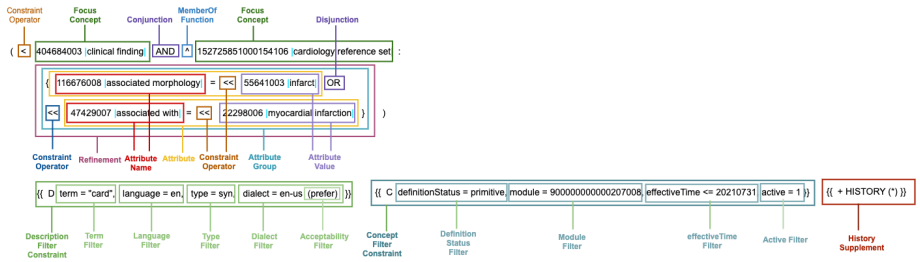

Figure 2: The main components of an example expression constraint

The expression constraint in Figure 2 is satisfied by concepts which are clinical findings **and** members of the cardiology reference set **and** have an attribute group that either has an associated morphology of infarct (or descendant) **or** are associated with myocardial infarction (or descendant). In addition, all matching concepts must also have a description that matches the term "card", has a language of English, has a type of |Synonym| and are preferred in the en-us language reference set. And matching concepts must be primitive, belong to the international core module, be published on or before 31st July 2021, and be active. The results of this expression constraint are then supplemented by any inactive concept that is associated with the active results via an historical association reference set.

4.1 Details

Figure 3 below provides a non-normative representation of the logical model of the SNOMED CT Expression Constraint Language 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 Chapter 5. For a short description of each of these, please refer to Section 5.4.

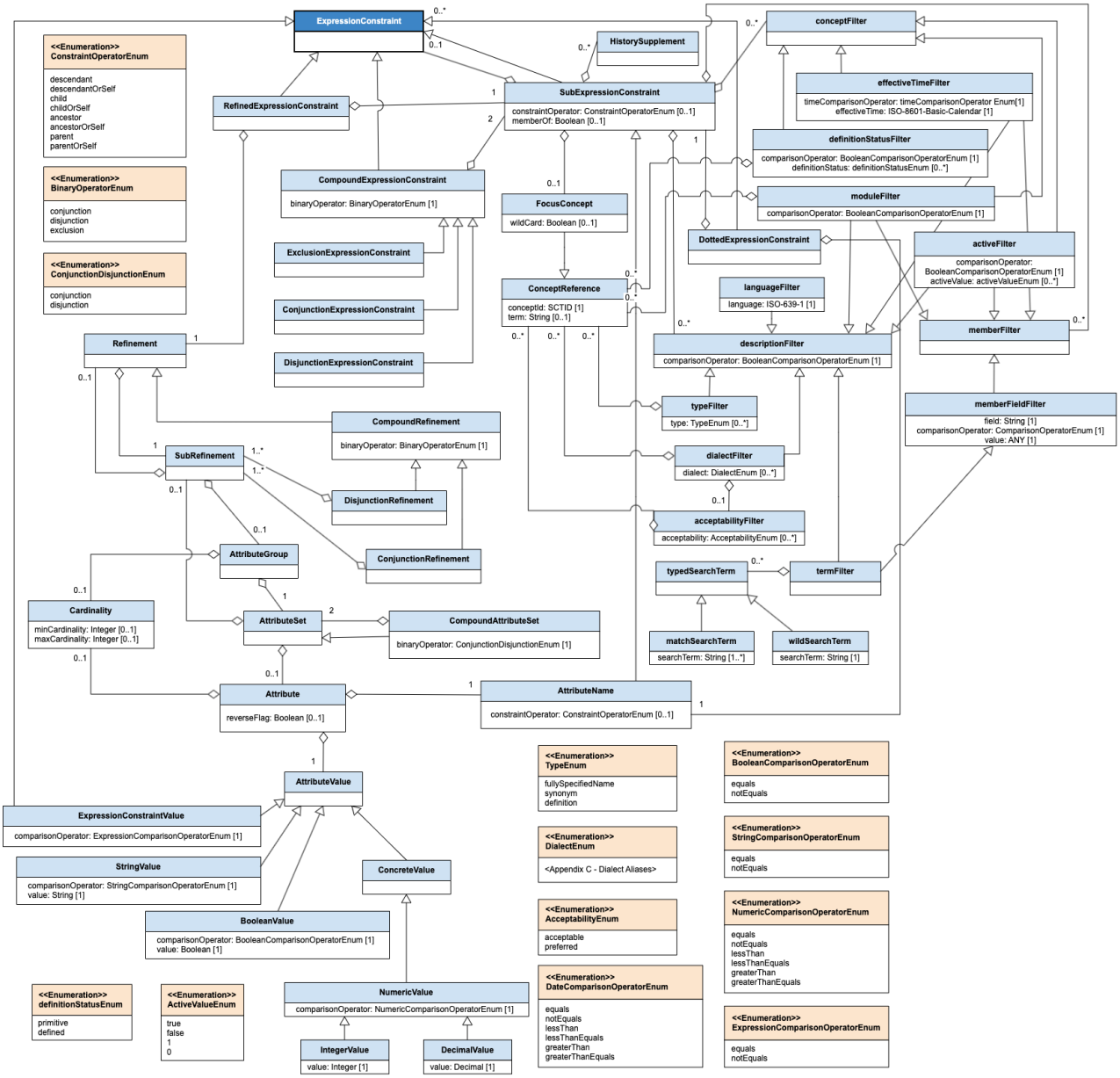

Figure 3: Logical Model of Expression Constraint Language

5. Syntax Specification

The following sections describe two syntaxes for use with the SNOMED CT Expression Constraint Language. These syntaxes are serialised representations of the logical model presented in the previous chapter, and are therefore logically equivalent.

The first of these syntaxes is referred to as the 'brief syntax' as it primarily uses a symbolic representation aimed to be as compact as possible. This syntax is considered to be the normative syntax, and is recommended for use in interoperable communications between systems.

The second syntax is referred to as the 'long syntax'. The long syntax introduces English-based textual alternatives to the symbols defined in the 'brief syntax', with the aim of increasing the human readability of the language. The textual alternatives provided in the 'long syntax' may (in theory) be translated into other languages to provide equivalent expression constraint representations that are human-readable by non-English speakers. Please note that the 'long syntax' (and any translations) is non-normative, and should only be used when a reliable mapping to the normative brief syntax is possible.

Please note that by default each expression constraint is evaluated against only the active components (and active members of each reference set) from the snapshot release (in distribution normal form) of a specified SNOMED CT versioned edition.

5.1 Brief Syntax (Normative)

The following ABNF definition specifies the Brief Syntax of the SNOMED CT Expression Constraint Language.

expressionConstraint = *ws (refinedExpressionConstraint / compoundExpressionConstraint / dottedExpressionConstraint / subExpressionConstraint) ws*
refinedExpressionConstraint = *subExpressionConstraint ws ":" ws eclRefinement*
compoundExpressionConstraint = *conjunctionExpressionConstraint / disjunctionExpressionConstraint / exclusionExpressionConstraint*
conjunctionExpressionConstraint = *subExpressionConstraint 1*(ws conjunction ws subExpressionConstraint)*
disjunctionExpressionConstraint = *subExpressionConstraint 1*(ws disjunction ws subExpressionConstraint)*
exclusionExpressionConstraint = *subExpressionConstraint ws exclusion ws subExpressionConstraint*
dottedExpressionConstraint = *subExpressionConstraint 1*(ws dottedExpressionAttribute)*
dottedExpressionAttribute = *dot ws eclAttributeName*
subExpressionConstraint= *[constraintOperator ws] (([memberOf ws] (eclFocusConcept / "(" ws expressionConstraint ws ")") *(ws memberFilterConstraint)) / (eclFocusConcept / "(" ws expressionConstraint ws ")")) *(ws (descriptionFilterConstraint / conceptFilterConstraint)) [ws historySupplement]*
eclFocusConcept = *eclConceptReference / wildCard*
dot = "."
memberOf = *"^" [ws "[" ws (refsetFieldNameSet / wildCard) ws "]"]*
refsetFieldNameSet = *refsetFieldName *(ws "," ws refsetFieldName)*
refsetFieldName = *1*alpha*
eclConceptReference = *conceptId [ws "|" ws term ws "|"]*
eclConceptReferenceSet = *"(" ws eclConceptReference 1*(mws eclConceptReference) ws ")"*
conceptId = *sctId*
term = *1*nonwsNonPipe *(1*SP 1*nonwsNonPipe)*
wildCard = *"*"*
constraintOperator = *childOf / childOrSelfOf / descendantOrSelfOf / descendantOf / parentOf / parentOrSelfOf / ancestorOrSelfOf / ancestorOf*
descendantOf = *"<"*
descendantOrSelfOf = *"<<"*
childOf = *"<!"*
childOrSelfOf = *"<<!"*
ancestorOf = *">"*
ancestorOrSelfOf = *">>"*
parentOf = *">!"*

parentOrSelfOf *= ">>!"*
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*
eclRefinement *= subRefinement ws [conjunctionRefinementSet / disjunctionRefinementSet]*
conjunctionRefinementSet *= 1*(ws conjunction ws subRefinement)*
disjunctionRefinementSet *= 1*(ws disjunction ws subRefinement)*
subRefinement *= eclAttributeSet / eclAttributeGroup / "(" ws eclRefinement ws ")"*
eclAttributeSet *= subAttributeSet ws [conjunctionAttributeSet / disjunctionAttributeSet]*
conjunctionAttributeSet *= 1*(ws conjunction ws subAttributeSet)*
disjunctionAttributeSet *= 1*(ws disjunction ws subAttributeSet)*
subAttributeSet *= eclAttribute / "(" ws eclAttributeSet ws ")"*
eclAttributeGroup *= ["[" cardinality "]" ws] "{" ws eclAttributeSet ws "}"*
eclAttribute *= ["[" cardinality "]" ws] [reverseFlag ws] eclAttributeName ws (expressionComparisonOperator ws subExpressionConstraint / numericComparisonOperator ws "#" numericValue / stringComparisonOperator ws (typedSearchTerm / typedSearchTermSet) / booleanComparisonOperator ws booleanValue)*
cardinality *= minValue to maxValue*
minValue *= nonNegativeIntegerValue*
to *= ".."*
maxValue *= nonNegativeIntegerValue / many*
many *= "*"*
reverseFlag *= "R"*
eclAttributeName *= subExpressionConstraint*
expressionComparisonOperator *= "=" / "!="*
numericComparisonOperator *= "=" / "!=" / "<=" / "<" / ">=" / ">"*
timeComparisonOperator *= "=" / "!=" / "<=" / "<" / ">=" / ">"*
stringComparisonOperator *= "=" / "!="*
booleanComparisonOperator *= "=" / "!="*
descriptionFilterConstraint *= "{{" ws ["d" / "D"] ws descriptionFilter *(ws "," ws descriptionFilter) ws "}}"*
descriptionFilter *= termFilter / languageFilter / typeFilter / dialectFilter / moduleFilter / effectiveTimeFilter / activeFilter*
termFilter *= termKeyword ws stringComparisonOperator ws (typedSearchTerm / typedSearchTermSet)*
termKeyword *= ("t"/"T") ("e"/"E") ("r"/"R") ("m"/"M")*
typedSearchTerm *= ([match ws ":" ws] matchSearchTermSet) / (wild ws ":" ws wildSearchTermSet)*
typedSearchTermSet *= "(" ws typedSearchTerm *(mws typedSearchTerm) ws ")"*
wild *= ("w"/"W") ("i"/"I") ("l"/"L") ("d"/"D")*
match *= ("m"/"M") ("a"/"A") ("t"/"T") ("c"/"C") ("h"/"H")*
matchSearchTerm = *1*(nonwsNonEscapedChar / escapedChar)*
matchSearchTermSet *= QM ws matchSearchTerm *(mws matchSearchTerm) ws QM*
wildSearchTerm *= 1*(anyNonEscapedChar / escapedWildChar)*
wildSearchTermSet *= QM wildSearchTerm QM*
languageFilter *= language ws booleanComparisonOperator ws (languageCode / languageCodeSet)*
language *= ("l"/"L") ("a"/"A") ("n"/"N") ("g"/"G") ("u"/"U") ("a"/"A") ("g"/"G") ("e"/"E")*
languageCode *= 2alpha*
languageCodeSet *= "(" ws languageCode *(mws languageCode) ws ")"*
typeFilter *= typeIdFilter / typeTokenFilter*
typeIdFilter *= typeId ws booleanComparisonOperator ws (subExpressionConstraint / eclConceptReferenceSet)*
typeId *= ("t"/"T") ("y"/"Y") ("p"/"P") ("e"/"E") ("i"/"I") ("d"/"D")*
typeTokenFilter *= type ws booleanComparisonOperator ws (typeToken / typeTokenSet)*
type *= ("t"/"T") ("y"/"Y") ("p"/"P") ("e"/"E")*
typeToken *= synonym / fullySpecifiedName / definition*
typeTokenSet *= "(" ws typeToken *(mws typeToken) ws ")"*
synonym *= ("s"/"S") ("y"/"Y") ("n"/"N")*
fullySpecifiedName *= ("f"/"F") ("s"/"S") ("n"/"N")*
definition *= ("d"/"D") ("e"/"E") ("f"/"F")*

dialectFilter = (dialectIdFilter / dialectAliasFilter) [ws acceptabilitySet]
dialectIdFilter = dialectId ws booleanComparisonOperator ws (subExpressionConstraint / dialectIdSet)
dialectId = ("d"/"D") ("i"/"I") ("a"/"A") ("l"/"L") ("e"/"E") ("c"/"C") ("t"/"T") ("i"/"I") ("d"/"D")
dialectAliasFilter = dialect ws booleanComparisonOperator ws (dialectAlias / dialectAliasSet)
dialect = ("d"/"D") ("i"/"I") ("a"/"A") ("l"/"L") ("e"/"E") ("c"/"C") ("t"/"T")
dialectAlias = alpha *(dash / alpha / integerValue)
dialectAliasSet = "(" ws dialectAlias [ws acceptabilitySet] *(mws dialectAlias [ws acceptabilitySet]) ws ")"
dialectIdSet = "(" ws eclConceptReference [ws acceptabilitySet] *(mws eclConceptReference [ws acceptabilitySet]) ws ")"
acceptabilitySet = acceptabilityConceptReferenceSet / acceptabilityTokenSet
acceptabilityConceptReferenceSet = "(" ws eclConceptReference *(mws eclConceptReference) ws ")"
acceptabilityTokenSet = "(" ws acceptabilityToken *(mws acceptabilityToken) ws ")"
acceptabilityToken = acceptable / preferred
acceptable = ("a"/"A") ("c"/"C") ("c"/"C") ("e"/"E") ("p"/"P") ("t"/"T")
preferred = ("p"/"P") ("r"/"R") ("e"/"E") ("f"/"F") ("e"/"E") ("r"/"R")
conceptFilterConstraint = "{{" ws ("c" / "C") ws conceptFilter *(ws "," ws conceptFilter) ws "}}"
conceptFilter = definitionStatusFilter / moduleFilter / effectiveTimeFilter / activeFilter
definitionStatusFilter = definitionStatusIdFilter / definitionStatusTokenFilter
definitionStatusIdFilter = definitionStatusIdKeyword ws booleanComparisonOperator ws (subExpressionConstraint / eclConceptReferenceSet)
definitionStatusIdKeyword = ("d"/"D") ("e"/"E") ("f"/"F") ("i"/"I") ("n"/"N") ("i"/"I") ("t"/"T") ("i"/"I") ("o"/"O") ("n"/"N") ("s"/"S") ("t"/"T") ("a"/"A") ("t"/"T") ("u"/"U") ("s"/"S") ("i"/"I") ("d"/"D")
definitionStatusTokenFilter = definitionStatusKeyword ws booleanComparisonOperator ws (definitionStatusToken / definitionStatusTokenSet)
definitionStatusKeyword = ("d"/"D") ("e"/"E") ("f"/"F") ("i"/"I") ("n"/"N") ("i"/"I") ("t"/"T") ("i"/"I") ("o"/"O") ("n"/"N") ("s"/"S") ("t"/"T") ("a"/"A") ("t"/"T") ("u"/"U") ("s"/"S")
definitionStatusToken = primitiveToken / definedToken
definitionStatusTokenSet = "(" ws definitionStatusToken *(mws definitionStatusToken) ws ")"
primitiveToken = ("p"/"P") ("r"/"R") ("i"/"I") ("m"/"M") ("i"/"I") ("t"/"T") ("i"/"I") ("v"/"V") ("e"/"E")
definedToken = ("d"/"D") ("e"/"E") ("f"/"F") ("i"/"I") ("n"/"N") ("e"/"E") ("d"/"D")
moduleFilter = moduleIdKeyword ws booleanComparisonOperator ws (subExpressionConstraint / eclConceptReferenceSet)
moduleIdKeyword = ("m"/"M") ("o"/"O") ("d"/"D") ("u"/"U") ("l"/"L") ("e"/"E") ("i"/"I") ("d"/"D")
effectiveTimeFilter = effectiveTimeKeyword ws timeComparisonOperator ws (timeValue / timeValueSet)
effectiveTimeKeyword
= ("e"/"E") ("f"/"F") ("f"/"F") ("e"/"E") ("c"/"C") ("t"/"T") ("i"/"I") ("v"/"V") ("e"/"E") ("t"/"T") ("i"/"I") ("m"/"M") ("e"/"E")
timeValue = QM [year month day] QM
timeValueSet = "(" ws timeValue *(mws timeValue) ws ")"
year = digitNonZero digit digit digit
month = "01" / "02" / "03" / "04" / "05" / "06" / "07" / "08" / "09" / "10" / "11" / "12"
day = "01" / "02" / "03" / "04" / "05" / "06" / "07" / "08" / "09" / "10" / "11" / "12" / "13" / "14" / "15" / "16" / "17" / "18" / "19" / "20" / "21" / "22" / "23" / "24" / "25" / "26" / "27" / "28" / "29" / "30" / "31"
activeFilter = activeKeyword ws booleanComparisonOperator ws activeValue
activeKeyword = ("a"/"A") ("c"/"C") ("t"/"T") ("i"/"I") ("v"/"V") ("e"/"E")
activeValue = activeTrueValue / activeFalseValue
activeTrueValue = "1" / "true"
activeFalseValue = "0" / "false"
memberFilterConstraint = "{{" ws ("m" / "M") ws memberFilter *(ws "," ws memberFilter) ws "}}"
memberFilter = memberFieldFilter / moduleFilter / effectiveTimeFilter / activeFilter
memberFieldFilter = refsetFieldName ws (expressionComparisonOperator ws subExpressionConstraint / numericComparisonOperator ws "#" numericValue / stringComparisonOperator ws (typedSearchTerm / typedSearchTermSet) / booleanComparisonOperator ws booleanValue / ws timeComparisonOperator ws (timeValue / timeValueSet))
historySupplement = "{{" ws "+" ws historyKeyword [historyProfileSuffix / ws historySubset] ws "}}"

historyKeyword *= ("h"/"H") ("i"/"I") ("s"/"S") ("t"/"T") ("o"/"O") ("r"/"R") ("y"/"Y")*
historyProfileSuffix *= historyMinimumSuffix / historyModerateSuffix / historyMaximumSuffix*
historyMinimumSuffix *= ("-"/"_") ("m"/"M") ("i"/"I") ("n"/"N")*
historyModerateSuffix *= ("-"/"_") ("m"/"M") ("o"/"O") ("d"/"D")*
historyMaximumSuffix *= ("-"/"_") ("m"/"M") ("a"/"A") ("x"/"X")*
historySubset *= "(" ws expressionConstraint ws ")"*
numericValue *= ["-"/"+"] (decimalValue / integerValue)*
stringValue *= 1*(anyNonEscapedChar / escapedChar)*
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*
star *= %x2A ; asterisk*
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*
escapedWildChar *= BS QM / BS BS / BS star*
nonwsNonEscapedChar *= %x21 / %x23-5B / %x5D-7E / UTF8-2 / UTF8-3 / UTF8-4*
alpha *= %x41-5A / %x61-7A*
dash *= %x2D*
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*

5.2 Long Syntax (Informative)

The following ABNF definition specifies the Long Syntax the SNOMED CT Expression Constraint Language. Please note that all keywords are case insensitive.

expressionConstraint *= ws (refinedExpressionConstraint / compoundExpressionConstraint / dottedExpressionConstraint / subExpressionConstraint) ws*
refinedExpressionConstraint *= subExpressionConstraint ws ":" ws eclRefinement*
compoundExpressionConstraint *= conjunctionExpressionConstraint / disjunctionExpressionConstraint / exclusionExpressionConstraint*
conjunctionExpressionConstraint *= subExpressionConstraint 1*(ws conjunction ws subExpressionConstraint)*
disjunctionExpressionConstraint *= subExpressionConstraint 1*(ws disjunction ws subExpressionConstraint)*
exclusionExpressionConstraint *= subExpressionConstraint ws exclusion ws subExpressionConstraint*

dottedExpressionConstraint = *subExpressionConstraint 1*(ws dottedExpressionAttribute)*
dottedExpressionAttribute = *dot ws eclAttributeName*
subExpressionConstraint= *[constraintOperator ws] (([memberOf ws] (eclFocusConcept / "(" ws expressionConstraint ws ")") *(ws memberFilterConstraint)) / (eclFocusConcept / "(" ws expressionConstraint ws ")")) *(ws (descriptionFilterConstraint / conceptFilterConstraint)) [ws historySupplement]*
eclFocusConcept = *eclConceptReference / wildCard*
dot = *"."*
memberOf = *("^" / ("m"/"M") ("e"/"E") ("m"/"M") ("b"/"B") ("e"/"E") ("r"/"R") ("o"/"O") ("f"/"F")) [ws "[" ws (refsetFieldNameSet / wildCard) ws "]"]*
refsetFieldNameSet = *refsetFieldName *(ws "," ws refsetFieldName)*
refsetFieldName = *1*alpha*
eclConceptReference = *conceptId [ws "|" ws term ws "|"]*
eclConceptReferenceSet = *"(" ws eclConceptReference 1*(mws eclConceptReference) ws ")"*
conceptId = *sctId*
term = *1*nonwsNonPipe *(1*SP 1*nonwsNonPipe)*
wildCard = *"*" / (("a"/"A") ("n"/"N") ("y"/"Y"))*
constraintOperator = *childOf / childOrSelfOf / descendantOrSelfOf / descendantOf / parentOf / parentOrSelfOf / ancestorOrSelfOf / ancestorOf*
descendantOf = *"<" / (("d"/"D") ("e"/"E") ("s"/"S") ("c"/"C") ("e"/"E") ("n"/"N") ("d"/"D") ("a"/"A") ("n"/"N") ("t"/"T") ("o"/"O") ("f"/"F") mws)*
descendantOrSelfOf = *"<<" / (("d"/"D") ("e"/"E") ("s"/"S") ("c"/"C") ("e"/"E") ("n"/"N") ("d"/"D") ("a"/"A") ("n"/"N") ("t"/"T") ("o"/"O") ("r"/"R") ("s"/"S") ("e"/"E") ("l"/"L") ("f"/"F") ("o"/"O") ("f"/"F") mws)*
childOf = *"<!" / (("c"/"C") ("h"/"H") ("i"/"I") ("l"/"L") ("d"/"D") ("o"/"O") ("f"/"F") mws)*
childOrSelfOf = *"<<!" / (("c"/"C") ("h"/"H") ("i"/"I") ("l"/"L") ("d"/"D") ("o"/"O") ("r"/"R") ("s"/"S") ("e"/"E") ("l"/"L") ("f"/"F") ("o"/"O") ("f"/"F") mws)*
ancestorOf = *">" / (("a"/"A") ("n"/"N") ("c"/"C") ("e"/"E") ("s"/"S") ("t"/"T") ("o"/"O") ("r"/"R") ("o"/"O") ("f"/"F") mws)*
ancestorOrSelfOf = *">>" / (("a"/"A") ("n"/"N") ("c"/"C") ("e"/"E") ("s"/"S") ("t"/"T") ("o"/"O") ("r"/"R") ("o"/"O") ("r"/"R") ("s"/"S") ("e"/"E") ("l"/"L") ("f"/"F") ("o"/"O") ("f"/"F") mws)*
parentOf = *">!" / (("p"/"P") ("a"/"A") ("r"/"R") ("e"/"E") ("n"/"N") ("t"/"T") ("o"/"O") ("f"/"F") mws)*
parentOrSelfOf = *">>!" / (("p"/"P") ("a"/"A") ("r"/"R") ("e"/"E") ("n"/"N") ("t"/"T") ("o"/"O") ("r"/"R") ("s"/"S") ("e"/"E") ("l"/"L") ("f"/"F") ("o"/"O") ("f"/"F") mws)*
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*
eclRefinement = *subRefinement ws [conjunctionRefinementSet / disjunctionRefinementSet]*
conjunctionRefinementSet = *1*(ws conjunction ws subRefinement)*
disjunctionRefinementSet = *1*(ws disjunction ws subRefinement)*
subRefinement = *eclAttributeSet / eclAttributeGroup / "(" ws eclRefinement ws ")"*
eclAttributeSet = *subAttributeSet ws [conjunctionAttributeSet / disjunctionAttributeSet]*
conjunctionAttributeSet = *1*(ws conjunction ws subAttributeSet)*
disjunctionAttributeSet = *1*(ws disjunction ws subAttributeSet)*
subAttributeSet = *eclAttribute / "(" ws eclAttributeSet ws ")"*
eclAttributeGroup = *["[" cardinality "]" ws] "{" ws eclAttributeSet ws "}"*
eclAttribute = *["[" cardinality "]" ws] [reverseFlag ws] eclAttributeName ws (expressionComparisonOperator ws subExpressionConstraint / numericComparisonOperator ws "#" numericValue / stringComparisonOperator ws (typedSearchTerm / typedSearchTermSet) / booleanComparisonOperator ws booleanValue)*
cardinality = *minValue to maxValue*
minValue = *nonNegativeIntegerValue*
to = *".." / (mws ("t"/"T") ("o"/"O") mws)*
maxValue = *nonNegativeIntegerValue / many*
many = *"*" / (("m"/"M") ("a"/"A") ("n"/"N") ("y"/"Y"))*
reverseFlag = *(("r"/"R") ("e"/"E") ("v"/"V") ("e"/"E") ("r"/"R") ("s"/"S") ("e"/"E") ("o"/"O") ("f"/"F")) / "R"*
eclAttributeName = *subExpressionConstraint*
expressionComparisonOperator = *"=" / "!=" / ("n"/"N") ("o"/"O") ("t"/"T") ws "=" / "<>"*

numericComparisonOperator *= "=" / "!=" / ("n"/"N") ("o"/"O") ("t"/"T") ws "=" / "<>" / "<=" / "<" / ">=" / ">"*
timeComparisonOperator *= "=" / "!=" / ("n"/"N") ("o"/"O") ("t"/"T") ws "=" / "<>" / "<=" / "<" / ">=" / ">"*
stringComparisonOperator *= "=" / "!=" / ("n"/"N") ("o"/"O") ("t"/"T") ws "=" / "<>"*
booleanComparisonOperator *= "=" / "!=" / ("n"/"N") ("o"/"O") ("t"/"T") ws "=" / "<>"*
filterConstraint *= descriptionFilterConstraint / conceptFilterConstraint / memberFilterConstraint / historySupplement*
descriptionFilterConstraint *= "{{" ws ["d" / "D"] ws descriptionFilter *(ws "," ws descriptionFilter) ws "}}"*
descriptionFilter *= termFilter / languageFilter / typeFilter / dialectFilter / moduleFilter / effectiveTimeFilter / activeFilter*
termFilter *= termKeyword ws stringComparisonOperator ws (typedSearchTerm / typedSearchTermSet)*
termKeyword *= ("t"/"T") ("e"/"E") ("r"/"R") ("m"/"M")*
typedSearchTerm *= ([match ws ":" ws] matchSearchTermSet) / (wild ws ":" ws wildSearchTermSet)*
typedSearchTermSet *= "(" ws typedSearchTerm *(mws typedSearchTerm) ws ")"*
wild *= ("w"/"W") ("i"/"I") ("l"/"L") ("d"/"D")*
match *= ("m"/"M") ("a"/"A") ("t"/"T") ("c"/"C") ("h"/"H")*
matchSearchTerm = *1*(nonwsNonEscapedChar / escapedChar)*
matchSearchTermSet *= QM ws matchSearchTerm *(mws matchSearchTerm) ws QM*
wildSearchTerm *= 1*(anyNonEscapedChar / escapedWildChar)*
wildSearchTermSet *= QM wildSearchTerm QM*
languageFilter *= language ws booleanComparisonOperator ws (languageCode / languageCodeSet)*
language *= ("l"/"L") ("a"/"A") ("n"/"N") ("g"/"G") ("u"/"U") ("a"/"A") ("g"/"G") ("e"/"E")*
languageCode *= 2alpha*
languageCodeSet *= "(" ws languageCode *(mws languageCode) ws ")"*
typeFilter *= typeIdFilter / typeTokenFilter*
typeIdFilter *= typeId ws booleanComparisonOperator ws (subExpressionConstraint / eclConceptReferenceSet)*
typeId *= ("t"/"T") ("y"/"Y") ("p"/"P") ("e"/"E") ("i"/"I") ("d"/"D")*
typeTokenFilter *= type ws booleanComparisonOperator ws (typeToken / typeTokenSet)*
type *= ("t"/"T") ("y"/"Y") ("p"/"P") ("e"/"E")*
typeToken *= synonym / fullySpecifiedName / definition*
typeTokenSet *= "(" ws typeToken *(mws typeToken) ws ")"*
synonym *= ("s"/"S") ("y"/"Y") ("n"/"N") [("o"/"O") ("n"/"N") ("y"/"Y") ("m"/"M")]*
fullySpecifiedName *= (("f"/"F") ("s"/"S") ("n"/"N")) / (("f"/"F") ("u"/"U") ("l"/"L") ("l"/"L") ("y"/"Y") ("s"/"S") ("p"/"P") ("e"/"E") ("c"/"C") ("i"/"I") ("f"/"F") ("i"/"I") ("e"/"E") ("d"/"D") ("n"/"N") ("a"/"A") ("m"/"M") ("e"/"E"))*
definition *= ("d"/"D") ("e"/"E") ("f"/"F") [("i"/"I") ("n"/"N") ("i"/"I") ("t"/"T") ("i"/"I") ("o"/"O") ("n"/"N")]*
dialectFilter *= (dialectIdFilter / dialectAliasFilter) [ws acceptabilitySet]*
dialectIdFilter *= dialectId ws booleanComparisonOperator ws (subExpressionConstraint / dialectIdSet)*
dialectId *= ("d"/"D") ("i"/"I") ("a"/"A") ("l"/"L") ("e"/"E") ("c"/"C") ("t"/"T") ("i"/"I") ("d"/"D")*
dialectAliasFilter *= dialect ws booleanComparisonOperator ws (dialectAlias / dialectAliasSet)*
dialect *= ("d"/"D") ("i"/"I") ("a"/"A") ("l"/"L") ("e"/"E") ("c"/"C") ("t"/"T")*
dialectAlias *= alpha *(dash / alpha / integerValue)*
dialectAliasSet *= "(" ws dialectAlias [ws acceptabilitySet] *(mws dialectAlias [ws acceptabilitySet]) ws ")"*
dialectIdSet *= "(" ws eclConceptReference [ws acceptabilitySet] *(mws eclConceptReference [ws acceptabilitySet]) ws ")"*
acceptabilitySet *= acceptabilityConceptReferenceSet / acceptabilityTokenSet*
acceptabilityConceptReferenceSet *= "(" ws eclConceptReference *(mws eclConceptReference) ws ")"*
acceptabilityTokenSet *= "(" ws acceptabilityToken *(mws acceptabilityToken) ws ")"*
acceptabilityToken *= acceptable / preferred*
acceptable *= ("a"/"A") ("c"/"C") ("c"/"C") ("e"/"E") ("p"/"P") ("t"/"T") [("a"/"A") ("b"/"B") ("l"/"L") ("e"/"E")]*
preferred *= ("p"/"P") ("r"/"R") ("e"/"E") ("f"/"F") ("e"/"E") ("r"/"R") [("r"/"R") ("e"/"E") ("d"/"D")]*
conceptFilterConstraint *= "{{" ws ("c" / "C") ws conceptFilter *(ws "," ws conceptFilter) ws "}}"*
conceptFilter *= definitionStatusFilter / moduleFilter / effectiveTimeFilter / activeFilter*
definitionStatusFilter *= definitionStatusIdFilter / definitionStatusTokenFilter*
definitionStatusIdFilter *= definitionStatusIdKeyword ws booleanComparisonOperator ws (subExpressionConstraint / eclConceptReferenceSet)*

definitionStatusIdKeyword *= ("d"/"D") ("e"/"E") ("f"/"F") ("i"/"I") ("n"/"N") ("i"/"I") ("t"/"T") ("i"/"I") ("o"/"O") ("n"/"N") ("s"/"S") ("t"/"T") ("a"/"A") ("t"/"T") ("u"/"U") ("s"/"S") ("i"/"I") ("d"/"D")*
definitionStatusTokenFilter *= definitionStatusKeyword ws booleanComparisonOperator ws (definitionStatusToken / definitionStatusTokenSet)*
definitionStatusKeyword *= ("d"/"D") ("e"/"E") ("f"/"F") ("i"/"I") ("n"/"N") ("i"/"I") ("t"/"T") ("i"/"I") ("o"/"O") ("n"/"N") ("s"/"S") ("t"/"T") ("a"/"A") ("t"/"T") ("u"/"U") ("s"/"S")*
definitionStatusToken *= primitiveToken / definedToken*
definitionStatusTokenSet *= "(" ws definitionStatusToken *(mws definitionStatusToken) ws ")"*
primitiveToken *= ("p"/"P") ("r"/"R") ("i"/"I") ("m"/"M") ("i"/"I") ("t"/"T") ("i"/"I") ("v"/"V") ("e"/"E")*
definedToken *= ("d"/"D") ("e"/"E") ("f"/"F") ("i"/"I") ("n"/"N") ("e"/"E") ("d"/"D")*
moduleFilter *= moduleIdKeyword ws booleanComparisonOperator ws (subExpressionConstraint / eclConceptReferenceSet)*
moduleIdKeyword *= ("m"/"M") ("o"/"O") ("d"/"D") ("u"/"U") ("l"/"L") ("e"/"E") ("i"/"I") ("d"/"D")*
effectiveTimeFilter *= effectiveTimeKeyword ws timeComparisonOperator ws (timeValue / timeValueSet)*
effectiveTimeKeyword
= ("e"/"E") ("f"/"F") ("f"/"F") ("e"/"E") ("c"/"C") ("t"/"T") ("i"/"I") ("v"/"V") ("e"/"E") ("t"/"T") ("i"/"I") ("m"/"M") ("e"/"E")
timeValue *= QM [year month day] QM*
timeValueSet *= "(" ws timeValue *(mws timeValue) ws ")"*
year *= digitNonZero digit digit digit*
month *= "01" / "02" / "03" / "04" / "05" / "06" / "07" / "08" / "09" / "10" / "11" / "12"*
day *= "01" / "02" / "03" / "04" / "05" / "06" / "07" / "08" / "09" / "10" / "11" / "12" / "13" / "14" / "15" / "16" / "17" / "18" / "19" / "20" / "21" / "22" / "23" / "24" / "25" / "26" / "27" / "28" / "29" / "30" / "31"*
activeFilter *= activeKeyword ws booleanComparisonOperator ws activeValue*
activeKeyword *= ("a"/"A") ("c"/"C") ("t"/"T") ("i"/"I") ("v"/"V") ("e"/"E")*
activeValue *= activeTrueValue / activeFalseValue*
activeTrueValue *= "1" / "true"*
activeFalseValue *= "0" / "false"*
memberFilterConstraint *= "{{" ws ("m" / "M") ws memberFilter *(ws "," ws memberFilter) ws "}}"*
memberFilter *= memberFieldFilter / moduleFilter / effectiveTimeFilter / activeFilter*
memberFieldFilter *= refsetFieldName ws (expressionComparisonOperator ws subExpressionConstraint / numericComparisonOperator ws "#" numericValue / stringComparisonOperator ws (typedSearchTerm / typedSearchTermSet) / booleanComparisonOperator ws booleanValue / ws timeComparisonOperator ws (timeValue / timeValueSet))*
historySupplement *= "{{" ws "+" ws historyKeyword [historyProfileSuffix / ws historySubset] ws "}}"*
historyKeyword *= ("h"/"H") ("i"/"I") ("s"/"S") ("t"/"T") ("o"/"O") ("r"/"R") ("y"/"Y")*
historyProfileSuffix *= historyMinimumSuffix / historyModerateSuffix / historyMaximumSuffix*
historyMinimumSuffix *= ("-"/"_") ("m"/"M") ("i"/"I") ("n"/"N")*
historyModerateSuffix *= ("-"/"_") ("m"/"M") ("o"/"O") ("d"/"D")*
historyMaximumSuffix *= ("-"/"_") ("m"/"M") ("a"/"A") ("x"/"X")*
historySubset *= "(" ws expressionConstraint ws ")"*
numericValue *= ["-"/"+"] (decimalValue / integerValue)*
stringValue *= 1*(anyNonEscapedChar / escapedChar)*
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*
star *= %x2A ; asterisk*
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*
escapedWildChar *= BS QM / BS BS / BS star*
nonwsNonEscapedChar *= %x21 / %x23-5B / %x5D-7E / UTF8-2 / UTF8-3 / UTF8-4*
alpha *= %x41-5A / %x61-7A*
dash *= %x2D*
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*

5.3 Informative Comments

This section provides a short description of each ABNF rule listed above. The related brief and long syntax rules are grouped together with the same description. Where the syntaxes are the same, the rule is listed once and preceded with the text "BS/LS". Where the brief and long syntaxes are different, both rules are listed separately and preceded with "BS" and "LS" respectively.

BS/LS: expressionConstraint = ws (refinedExpressionConstraint / compoundExpressionConstraint / dottedExpressionConstraint / subExpressionConstraint) ws	
	An expression constraint is either a refined expression constraint, a compound expression constraint, a dotted expression constraint, or a sub expression constraint.
BS/LS: refinedExpressionConstraint = subExpressionConstraint ws ":" ws eclRefinement	
	A refined expression constraint includes a subexpression constraint followed by a refinement.
BS/LS: compoundExpressionConstraint = conjunctionExpressionConstraint / disjunctionExpressionConstraint / exclusionExpressionConstraint	
	A compound expression constraint contains two or more expression constraints joined by either a conjunction, disjunction or exclusion. When potential ambiguity in binary operator precedence may occur, round brackets must be used to clearly disambiguate the order in which these operator are applied. Brackets are not required in expression constraints in which all binary operators are conjunctions, or all binary operators are disjunctions. Please note that unary operators (i.e. constraint operators and member of functions) are always applied before binary operators (i.e. conjunction, disjunction and exclusion).
BS/LS: conjunctionExpressionConstraint = subExpressionConstraint 1*(ws conjunction ws subExpressionConstraint)	
	A conjunction expression constraint combines two or more expression constraints with a conjunction ("and") operator. More than one conjunction may be used without brackets. However any compound expression constraint (using a different binary operator) that appears within a conjunction expression constraint must be enclosed by brackets.
BS/LS: disjunctionExpressionConstraint = subExpressionConstraint 1*(ws disjunction ws subExpressionConstraint)	

	A disjunction expression constraint combines two or more expression constraints with a disjunction ("or") operator. More than one disjunction may be used without brackets. However any compound expression constraint (using a different binary operator) that appears within a disjunction expression constraint must be enclosed by brackets.
BS/LS: exclusionExpressionConstraint = subExpressionConstraint ws exclusion ws subExpressionConstraint	
	An exclusion expression constraint combines two expression constrains with an exclusion ("minus") operator. A single exclusion operator may be used without brackets. However when the operands of the exclusion expression constraint are compound, these compound expression constraints must be enclosed by brackets.
BS/LS: dottedExpressionConstraint = subExpressionConstraint 1*(ws dottedExpressionAttribute)	
	A dotted expression constraint contains a sub expression constraint, followed by one or more dotted attributes. When a single dotted attribute is used, the result is the set of attribute values (for the given attribute name) of each concept that results from evaluating the subExpressionConstraint. When more than one dotted attribute is used, each dottedExpressionAttribute is sequentially evaluated (from left to right) against the given result set.
BS/LS: dottedExpressionAttribute = dot ws eclAttributeName	
	A dotted expression attribute consists of a 'dot', followed by an attribute name. Please note that the attribute name may be represented by any sub expression constraint.
BS/LS: subExpressionConstraint = [constraintOperator ws] (([memberOf ws] (eclFocusConcept / "(" ws expressionConstraint ws ")") *(ws memberFilterConstraint)) / (eclFocusConcept / "(" ws expressionConstraint ws ")")) *(ws (descriptionFilterConstraint / conceptFilterConstraint)) [ws historySupplement]	
	A sub expression constraint optionally begins with a constraint operator and/or a memberOf function. It then includes either a single focus concept or an expression constraint (enclosed in brackets). If the memberOf function is applied, a member filter constraint may be used. A sub expression constraint may then optionally include one or more concept or description filter constraints, followed optionally by a history supplement. Notes: A memberOf function should be used only when the eclFocusConcept or expressionConstraint refers to a reference set concept, a set of reference set concepts, or a wild card. When both a constraintOperator and a memberOf function are used, they are applied from the inside to out (i.e. from right to left) - see 5.4 Order of Operation. Therefore, if a constraintOperator is followed by a memberOf function, then the memberOf function is processed prior to the constraintOperator.
BS/LS: eclFocusConcept = eclConceptReference / wildCard	
	A focus concept is a concept reference or a wild card.
BS/LS: dot = "."	
	A dot connects an expression constraint with an attribute whose values are included in the result.
BS: memberOf = "^" [ws "[" ws (refsetFieldNameSet / wildCard) ws "]"] **LS: memberOf** = ("^" / ("m"/"M") ("e"/"E") ("m"/"M") ("b"/"B") ("e"/"E") ("r"/"R") ("o"/"O") ("f"/"F")) [ws "[" ws (refsetFieldNameSet / wildCard) ws "]"]	
	By default, the 'memberOf' function returns the set of referenced components in the set of reference sets which follows. In the brief syntax, the memberOf function is represented using the "^" symbol. In the long syntax, the text "memberOf " (case insensitive and followed by at least one white space) is also allowed. If a set of reference set fields is listed in square brackets after the memberOf function, then the values of these fields are returned.
BS/LS: refsetFieldNameSet = refsetFieldName *(ws "," ws refsetFieldName)	
	A refsetFieldNameSet is a set of one or more reference set fields, separated by a comma and optional whitespace.
BS/LS: refsetFieldName = 1*alpha	
	A refsetFieldName is the set of alphabetic characters used to name a reference set field.
BS/LS: eclConceptReference = conceptId [ws "|" ws term ws "|"]	

	A conceptReference is represented by a ConceptId, optionally followed by a term enclosed by a pair of "|" characters. Whitespace before or after the ConceptId is ignored as is any whitespace between the initial "|" characters and the first non-whitespace character in the term or between the last non-whitespace character and before second "|" character.
BS/LS: **eclConceptReferenceSet** = "(" ws eclConceptReference 1*(mws eclConceptReference) ws ")"	
	A concept reference set includes two or more concept references separated by mandatory white space and enclosed in brackets.
BS/LS: conceptId = sctId	
	The ConceptId must be a valid SNOMED CT identifier for a concept. The initial digit may not be zero. The smallest number of digits is six, and the maximum is 18.
BS/LS: term = 1*nonwsnonpipe *(1*SP 1*nonwsnonpipe)	
	The term must be the term from a SNOMED CT description that is associated with the concept identified by the preceding concept identifier. For example, the term could be the preferred description, or the preferred description associated with a particular translation. The term may include valid UTF-8 characters except for the pipe "
BS: wildCard = "*" **LS: wildCard** = "*" / (("a"/"A") ("n"/"N") ("y"/"Y"))	
	A wild card represents any concept in the given substrate. In the brief syntax, a wildcard is represented using the "*" symbol. In the long syntax, the text "ANY" (case insensitive) is also allowed.
BS/LS: constraintOperator = childOf / childOrSelfOf / descendantOrSelfOf / descendantOf / parentOf / parentOrSelfOf / ancestorOrSelfOf / ancestorOf	
	A constraint operator is either 'childOf', 'childOrSelfOf', 'descendantOrSelfOf', 'descendantOf', 'parentOf', 'parentOrSelfOf', 'ancestorOrSelfOf', or 'ancestorOf'.
BS: descendantOf = "<" **LS: descendantOf** = "<" / (("d"/"D") ("e"/"E") ("s"/"S") ("c"/"C") ("e"/"E") ("n"/"N") ("d"/"D")("a"/"A") ("n"/"N") ("t"/"T") ("o"/"O")("f"/"F") mws)	
	The descendantOf operator returns the set of all subtypes of the given concept (or set of concepts). In the brief syntax, the descendantOf operator is represented using the symbol "<". In the long syntax, the text "descendantOf" (case insensitive and followed by at least one white space) is also allowed.
BS: descendantOrSelfOf = "<<" **LS: descendantOrSelfOf** = "<<" / (("d"/"D") ("e"/"E") ("s"/"S") ("c"/"C") ("e"/"E") ("n"/"N") ("d"/"D") ("a"/"A") ("n"/"N") ("t"/"T") ("o"/"O") ("r"/"R") ("s"/"S")("e"/"E") ("l"/"L") ("f"/"F") ("o"/"O")("f"/"F") mws)	
	The descendantOrSelfOf operator returns the set of all subtypes of the given concept (or set of concepts), plus the concept (or set of concepts) itself. In the brief syntax, the descendantOrSelfOf operator is represented using the symbols "<<". In the long syntax, the text "descendantOrSelfOf" (case insensitive and followed by at least one white space) is also allowed.
BS: childOf = "<!" **LS: childOf** = "<!" / (("c"/"C") ("h"/"H") ("i"/"I") ("l"/"L") ("d"/"D") ("o"/"O") ("f"/"F") mws)	

	The childOf operator returns the set of all immediate children of the given concept (or set of concepts). In the brief syntax, the childOf operator is represented using the symbols "<!". In the long syntax, the text "childOf" (case insensitive and followed by at least one white space) is also allowed.
BS: **childOrSelfOf** = "<<!" **LS:** **childOrSelfOf** = "<<!" / (("c"/"C") ("h"/"H") ("i"/"I") ("l"/"L") ("d"/"D") ("o"/"O")("r"/"R") ("s"/"S")("e"/"E") ("l"/"L") ("f"/"F") ("o"/"O") ("f"/"F") mws)	
	The childOrSelfOf operator returns the set of all immediate children of the given concept (or set of concepts), plus the concept (or set of concepts) itself. In the brief syntax, the childOrSelfOf operator is represented using the symbols "<<!". In the long syntax, the text "childOrSelfOf" (case insensitive and followed by at least one white space) is also allowed.
BS: **ancestorOf** = ">" **LS:** **ancestorOf** = ">" / (("a"/"A") ("n"/"N") ("c"/"C") ("e"/"E") ("s"/"S") ("t"/"T") ("o"/"O") ("r"/"R") ("o"/"O")("f"/"F") mws)	
	The ancestorOf operator returns the set of all supertypes of the given concept (or set of concepts). In the brief syntax, the ancestorOf operator is represented using the symbol ">". In the long syntax, the text "ancestorOf " (case insensitive and followed by at least one white space) is also allowed.
BS: **ancestorOrSelfOf** = ">>" **LS:** **ancestorOrSelfOf** = ">>" / (("a"/"A") ("n"/"N") ("c"/"C") ("e"/"E") ("s"/"S") ("t"/"T") ("o"/"O") ("r"/"R") ("o"/"O") ("r"/"R") ("s"/"S") ("e"/"E") ("l"/"L" ("f"/"F") ("o"/"O")("f"/"F") mws)	
	The ancestorOrSelfOf operator returns the set of all supertypes of the given concept (or set of concepts), plus the concept (or set of concepts) itself. In the brief syntax, the ancestorOrSelfOf operator is represented using the symbols ">>". In the long syntax, the text "ancestorOrSelfOf" (case insensitive and followed by at least one white space) is also allowed.
BS: **parentOf** = ">!" **LS:** **parentOf** = ">!" / (("p"/"P") ("a"/"A") ("r"/"R") ("e"/"E") ("n"/"N") ("t"/"T") ("o"/"O") ("f"/"F") mws)	
	The parentOf operator returns the set of all immediate parents of the given concept (or set of concepts). In the brief syntax, the parentOf operator is represented using the symbols ">!". In the long syntax, the text "parentOf" (case insensitive and followed by at least one white space) is also allowed.
BS: **parentOrSelfOf** = ">>!" **LS:** **parentOrSelfOf** = ">>!" / (("p"/"P") ("a"/"A") ("r"/"R") ("e"/"E") ("n"/"N") ("t"/"T") ("o"/"O") ("r"/"R") ("s"/"S") ("e"/"E") ("l"/"L" ("f"/"F") ("o"/"O") ("f"/"F") mws)	
	The parentOrSelfOf operator returns the set of all immediate parents of the given concept (or set of concepts), plus the concept (or set of concepts) itself. In the brief syntax, the parentOrSelfOf operator is represented using the symbols ">>!". In the long syntax, the text "parentOrSelfOf" (case insensitive and followed by at least one white space) is also allowed.
BS/LS: **conjunction** = (("a"/"A") ("n"/"N") ("d"/"D") mws) / ","	
	A conjunction is represented either by the word "and" (case insensitive and followed by at least one white space), or by a comma.
BS/LS: **disjunction** = ("o"/"O") ("r"/"R") mws	
	A disjunction is represented by the word "or" (case insensitive and followed by at least one white space).

BS/LS: exclusion = ("m"/"M") ("i"/"I") ("n"/"N") ("u"/"U") ("s"/"S") mws	
	The exclusion operator is represented by the word "minus" (case insensitive and followed by at least one white space).
BS/LS: eclRefinement = subRefinement ws [conjunctionRefinementSet / disjunctionRefinementSet]	
	A refinement contains all the grouped and ungrouped attributes that refine the set of clinical meanings satisfied by the expression constraint. Refinements may represent the conjunction or disjunction of two smaller refinements, and may optionally be placed in brackets. Where both conjunction and disjunction are used, brackets are mandatory to disambiguate the intended meaning.
BS/LS: conjunctionRefinementSet = 1*(ws conjunction ws subRefinement)	
	A conjunction refinement set consists of one or more conjunction operators, each followed by a subRefinement.
BS/LS: disjunctionRefinementSet = 1*(ws disjunction ws subRefinement)	
	A disjunction refinement set consists of one or more disjunction operators, each followed by a subRefinement.
BS/LS: subRefinement = eclAttributeSet / eclAttributeGroup / "(" ws eclRefinement ws ")"	
	A subRefinement is either an attribute set, an attribute group or a bracketed refinement.
BS/LS: eclAttributeSet = subAttributeSet ws [conjunctionAttributeSet / disjunctionAttributeSet]	
	An attribute set contains one or more attribute name-value pairs separated by a conjunction or disjunction operator. An attribute set may optionally be placed in brackets.
BS/LS: conjunctionAttributeSet = 1*(ws conjunction ws subAttributeSet)	
	A conjunction attribute set consists of one or more conjunction operators, each followed by a subAttributeSet.
BS/LS: disjunctionAttributeSet = 1*(ws disjunction ws subAttributeSet)	
	A disjunction attribute set consists of one or more disjunction operators, each followed by a subAttributeSet.
BS/LS: subAttributeSet = eclAttribute / "(" ws eclAttributeSet ws ")"	
	A subAttributeSet is either an attribute or a bracketed attribute set.
BS/LS: eclAttributeGroup = ["[" cardinality "]" ws] "{" ws eclAttributeSet ws "}"	
	An attribute group contains a collection of attributes that operate together as part of the refinement of the containing expression constraint. An attribute group may optionally be preceded by a cardinality. An attribute group cardinality indicates the minimum and maximum number of attribute groups that must satisfy the given attributeSet constraint for the expression constraint to be satisfied.
BS/LS: eclAttribute = ["[" cardinality "]" ws] [reverseFlag ws] eclAttributeName ws (expressionComparisonOperator ws subExpressionConstraint / numericComparisonOperator ws "#" numericValue / stringComparisonOperator ws (typedSearchTerm / typedSearchTermSet) / booleanComparisonOperator ws booleanValue)	

	An attribute is a name-value pair expressing a single refinement of the containing expression constraint. Either the attribute value must satisfy (or not) the given expression constraint, the attribute value is compared with a given numeric value (integer or decimal) using a numeric comparison operator, the attribute value must match (or not match) the given typedSearchTerm or typedSearchTermSet, or the attribute value must be equal to (or not equal to) the given boolean value. The attribute may optionally be preceded by a cardinality constraint and/or a reverse flag.
BS/LS: cardinality = minValue to maxValue	
	The cardinality represents a constraint on the minimum and maximum number of times that the given attribute or attribute group may appear in a matching expression. The cardinality is enclosed in square brackets with the minimum cardinality appearing first, followed by a separator (two dots in the brief syntax), and then the maximum cardinality.
BS/LS: minValue = nonNegativeIntegerValue	
	A value that represents the minimum number of times that an attribute or attribute group may appear. The minimum cardinality must always be less than or equal to the maximum cardinality.
BS: to = ".." **LS: to** = ".." / (mws ("t"/"T") ("o"/"O") mws)	
	In the brief syntax, the minimum and maximum cardinality are separated by two dots (i.e. ".."). In the long syntax, the text "to" (case insensitive with at least one white space before and after) is also allowed between the two cardinalities.
BS/LS: maxValue = nonNegativeIntegerValue / many	
	A value that represents the maximum number of times that an attribute or attribute group may appear. A maximum cardinality of 'many' indicates that there is no limit on the number of times the attribute may appear.
BS: many = "*" **LS: many** = "*" / (("m"/"M") ("a"/"A") ("n"/"N") ("y"/"Y"))	
	In the brief syntax, a cardinality of 'many' is represented using the symbol "*". In the long syntax, the text "many" (case insensitive, with no trailing space) is also allowed.
BS: reverseFlag = "R" **LS: reverseFlag** = (("r"/"R") ("e"/"E") ("v"/"V") ("e"/"E") ("r"/"R") ("s"/"S") ("e"/"E") ("o"/"O") ("f"/"F")) / "R"	
	When a reverse flag is used on an attribute, the matching relationships are traversed in the reverse of the normal direction. This means that the target concept of each relationship must match the focus concept to which the attribute is applied, while the source concept of the relationship must match the attribute value. In the brief syntax, the reverse flag is represented using the character "R" (in uppercase). In the long syntax, the text "reverseOf " (case insensitive) is also allowed.
BS/LS: eclAttributeName = subExpressionConstraint	
	The attribute name is the name of an attribute (or relationship type) to which a value is applied to refine the meaning of a containing expression constraint. The attribute name is represented using a subExpressionConstraint, as defined above.

BS: **expressionComparisonOperator** = "=" / "!=" **LS:** **expressionComparisonOperator** = "=" / "!=" / ("n"/"N") ("o"/"O") ("t"/"T") ws "=" / "<>"	
	Attributes whose value is a concept may be compared to an expression constraint using either equals ("=") or not equals ("!="). In the long syntax "<>" and "not =" (case insensitive) are also valid ways to represent not equals.
BS: **numericComparisonOperator** = "=" / "!=" / "<=" / "<" / ">=" / ">" **LS:** **numericComparisonOperator** = "=" / "!=" / ("n"/"N") ("o"/"O") ("t"/"T") ws "=" / "<>" / "<=" / "<" / ">=" / ">"	
	Attributes whose value is numeric (i.e. integer or decimal) may be compared to a specific concrete value using a variety of comparison operators, including equals ("="), less than ("<"), less than or equals ("<="), greater than (">"), greater than or equals (">=") and not equals ("!="). In the long syntax "<>" and "not =" (case insensitive) are also valid ways to represent not equals.
BS: **timeComparisonOperator** = "=" / "!=" / "<=" / "<" / ">=" / ">" **LS:** **timeComparisonOperator** = "=" / "!=" / ("n"/"N") ("o"/"O") ("t"/"T") ws "=" / "<>" / "<=" / "<" / ">=" / ">"	
	Date and time values may be compared using a variety of comparison operators, , including equals ("="), less than ("<"), less than or equals ("<="), greater than (">"), greater than or equals (">=") and not equals ("!="). In the long syntax "<>" and "not =" (case insensitive) are also valid ways to represent not equals.
BS: **stringComparisonOperator** = "=" / "!=" **LS:** **stringComparisonOperator** = "=" / "!=" / ("n"/"N") ("o"/"O") ("t"/"T") ws "=" / "<>"	
	Attributes whose value is a string may be compared to an expression constraint using either equals ("=") or not equals ("!="). In the long syntax "<>" and "not =" (case insensitive) are also valid ways to represent not equals.
BS: **booleanComparisonOperator** = "=" / "!=" **LS:** **booleanComparisonOperator** = "=" / "!=" / ("n"/"N") ("o"/"O") ("t"/"T") ws "=" / "<>"	
	Attributes whose value is a boolean may be compared to an expression constraint using either equals ("=") or not equals ("!="). In the long syntax "<>" and "not =" (case insensitive) are also valid ways to represent not equals.
BS/LS: **filterConstraint** = descriptionFilterConstraint / conceptFilterConstraint / memberFilterConstraint	
	A filterConstraint is either a description filter constraint, a concept filter constraint, or a member filter constraint.
BS/LS: **descriptionFilterConstraint** = "{{" ws ["d", / "D"] ws descriptionFilter *(ws "," ws descriptionFilter) ws "}}"	
	A descriptionFilterConstraint is a constraint used to filter the concepts in the result set, according to whether or not the given conditions match at least one of the concept's descriptions. A description filter constraint is always enclosed in double curly braces. Within these braces, it should (preferably) start with the letter 'D' followed by one or more description filters.
BS/LS: **descriptionFilter** = termFilter / languageFilter / typeFilter / dialectFilter / moduleFilter / effectiveTimeFilter / activeFilter	
	A description filter is either a term filter, a language filter, a type filter, a dialect filter, a module filter, an effective time filter, or an active filter.
BS/LS: **termFilter** = termKeyword ws stringComparisonOperator ws (typedSearchTerm / typedSearchTermSet)	
	A termFilter starts with the 'term' keyword, followed by a string comparison operator and either a typed search term or a typed search term set (with optional white space between). For example: term = "respiratory".
BS/LS: **termKeyword** = ("t"/"T") ("e"/"E") ("r"/"R") ("m"/"M")	
	The 'term' keyword uses the text "TERM" (case insensitive).
BS/LS: **typedSearchTerm** = ([match ws ":" ws] matchSearchTermSet) / (wild ws ":" ws wildSearchTermSet)	
	A typed search term is either a match search term set or a wild search term set. A match search term set is optionally preceded by the text "match" and a colon. A wild search term set must be preceded by the text "wild" and a colon.

BS/LS: **typedSearchTermSet** = "(" ws typedSearchTerm *(mws typedSearchTerm) ws ")"	
	A typed search term set consists of one or more typed search terms separated by mandatory white space and enclosed in brackets.
BS/LS: **wild** = ("w"/"W") ("i"/"I") ("l"/"L") ("d"/"D")	
	A 'wildcard' search type is indicated by the word "wild" (case insensitive).
BS/LS: **match** = ("m"/"M") ("a"/"A") ("t"/"T") ("c"/"C") ("h"/"H")	
	A 'word prefix any order' search is indicated by the word "match" (case insensitive).
BS/LS: **matchSearchTerm** = 1*(nonwsNonEscapedChar / escapedChar)	
	A term used in a match search includes one or more of any non-whitespace printable character (other than double quotes or backslash) or an escaped character.
BS/LS: **matchSearchTermSet** = QM ws matchSearchTerm *(mws matchSearchTerm) ws QM	
	A term set in a match search includes one or more terms separated by mandatory whitespace and enclosed in quotation marks.
BS/LS: **wildSearchTerm** = 1*(anyNonEscapedChar / escapedWildChar)	
	A term used in a wildcard search includes one or more printable characters (other than double quotes or backslash) or an escaped character.
BS/LS: **wildSearchTermSet** = QM wildSearchTerm QM	
	A term set in a wildcard search includes a wildcard search term (optionally including whitespace) enclosed in quotation marks.
BS/LS: **languageFilter** = language ws booleanComparisonOperator ws (languageCode / languageCodeSet)	
	A language filter specifies the languages that a matching description may use. A language filter starts with the 'language' keyword, followed by a boolean comparison operator and either a single language code or a set of language codes.
BS/LS: **language** = ("l"/"L") ("a"/"A") ("n"/"N") ("g"/"G") ("u"/"U") ("a"/"A") ("g"/"G") ("e"/"E")	
	The 'language' keyword uses the text "LANGUAGE" (case insensitive).
BS/LS: **languageCode** = 2alpha	
	A language code is a 2 character alphanumeric string.
BS/LS: **languageCodeSet** = "(" ws languageCode *(mws languageCode) ws ")"	
	A language code set is one or more language codes, separated by mandatory whitespace, and enclosed in brackets.
BS/LS: **typeFilter** = typeIdFilter / typeTokenFilter	
	A type filter specifies the description types that a matching description may have. A type filter is either a typeId filter or a typeToken filter.
BS/LS: **typeIdFilter** = typeId ws booleanComparisonOperator ws (subExpressionConstraint / eclConceptReferenceSet)	
	A typeId filter starts with the 'typeId' keyword, followed by a boolean comparison operator, and either a subExpressionConstraint or a set of concept references.
BS/LS: **typeId** = ("t"/"T") ("y"/"Y") ("p"/"P") ("e"/"E") ("i"/"I") ("d"/"D")	
	The 'typeId' keyword uses the text "TYPEID" (case insensitive).
BS/LS: **typeTokenFilter** = type ws booleanComparisonOperator ws (typeToken / typeTokenSet)	
	A typeToken filter starts with the 'type' keyword, followed by a boolean comparison operator, and either a single type token or a set of type tokens.
BS/LS: **type** = ("t"/"T") ("y"/"Y") ("p"/"P") ("e"/"E")	
	The 'type' keyword uses the text "TYPE" (case insensitive).
BS/LS: **typeToken** = synonym / fullySpecifiedName / definition	
	A type token is either a 'synonym' token, a 'fully specified name' token or a 'definition' token.
BS/LS: **typeTokenSet** = "(" ws typeToken *(mws typeToken) ws ")"	
	A type token set is one or more type tokens, separated by mandatory whitespace and enclosed in brackets.

BS: synonym = ("s"/"S") ("y"/"Y") ("n"/"N") **LS: synonym** = ("s"/"S") ("y"/"Y") ("n"/"N") [("o"/"O") ("n"/"N") ("y"/"Y") ("m"/"M")]	
	A 'synonym' token uses the text "SYN" (case insensitive). In the long syntax, the text "Synonym" (case insensitive) may be used instead.
BS: fullySpecifiedName = ("f"/"F") ("s"/"S") ("n"/"N") **LS: fullySpecifiedName** = (("f"/"F") ("s"/"S") ("n"/"N")) / (("f"/"F") ("u"/"U") ("l"/"L") ("l"/"L") ("y"/"Y") ("s"/"S") ("p"/"P") ("e"/"E") ("c"/"C") ("i"/"I") ("f"/"F") ("i"/"I") ("e"/"E") ("d"/"D") ("n"/"N") ("a"/"A") ("m"/"M") ("e"/"E"))	
	A 'fully specified name' token uses the text "FSN" (case insensitive). In the long syntax, the text "FullySpecifiedName" (case insensitive) may be used instead.
BS: definition = ("d"/"D") ("e"/"E") ("f"/"F") **LS: definition** = ("d"/"D") ("e"/"E") ("f"/"F") [("i"/"I") ("n"/"N") ("i"/"I") ("t"/"T") ("i"/"I") ("o"/"O") ("n"/"N")]	
	A 'definition' token uses the text "DEF" (case insensitive). In the long syntax, the text "Definition" (case insensitive) may be used instead.
BS/LS: dialectFilter = (dialectIdFilter / dialectAliasFilter) [ws acceptabilitySet]	
	A dialect filter specifies the language reference sets to which a matching description must belong. A dialect filter consists of either a dialectId filter or a dialectAlias filter, optionally followed by a set of acceptability values.
BS/LS: dialectIdFilter = dialectId ws booleanComparisonOperator ws (subExpressionConstraint / dialectIdSet)	
	A dialectId filter starts with the 'dialectId' keyword, followed by a boolean comparison operator, and either a subExpressionConstraint or a set of dialectIds.
BS/LS: dialectId = ("d"/"D") ("i"/"I") ("a"/"A") ("l"/"L") ("e"/"E") ("c"/"C") ("t"/"T") ("i"/"I") ("d"/"D")	
	A 'dialectId' keyword uses the text "DIALECTID" (case insensitive).
BS/LS: dialectAliasFilter = dialect ws booleanComparisonOperator ws (dialectAlias / dialectAliasSet)	
	A dialectAlias filter starts with the 'dialect' keyword, followed by a boolean comparison operator, and either a single dialect alias or a set of dialect aliases.
BS/LS: dialect = ("d"/"D") ("i"/"I") ("a"/"A") ("l"/"L") ("e"/"E") ("c"/"C") ("t"/"T")	
	A 'dialect' keyword uses the text "DIALECT" (case insensitive).
BS/LS: dialectAlias = alpha *(dash / alpha / integerValue)	
	A dialect alias consists of a single alphanumeric character followed by zero or more alphanumeric characters, integer values or dashes.
BS/LS: dialectAliasSet = "(" ws dialectAlias [ws acceptabilitySet] *(mws dialectAlias [ws acceptabilitySet]) ws ")"	
	A dialect alias set is one or more dialect aliases followed by an optional acceptability set, separated by mandatory white space, and enclosed in brackets.
BS/LS: dialectIdSet = "(" ws eclConceptReference [ws acceptabilitySet] *(mws eclConceptReference [ws acceptabilitySet]) ws ")"	
	A dialect id set is one or more concept references followed by an optional acceptability set, separated by mandatory white space, and enclosed in brackets.
BS/LS: acceptabilitySet = acceptabilityConceptReferenceSet / acceptabilityTokenSet	
	An acceptability set specifies the acceptabilities that a matching description must have in the language reference set specified by the preceding dialect filter. An acceptability set is either a set of one or more concept references or an acceptabilityToken set.
BS/LS: acceptabilityConceptReferenceSet = "(" ws eclConceptReference *(mws eclConceptReference) ws ")"	
	An acceptability concept reference set is a set of one or more references to concepts that are a < 900000000000511003 |Acceptability|.
BS/LS: acceptabilityTokenSet = "(" ws acceptabilityToken *(mws acceptabilityToken) ws ")"	
	An acceptability token set is one or more acceptability tokens, separated by mandatory whitespace, and enclosed in brackets.

BS/LS: acceptabilityToken = acceptable / preferred	
	An acceptability token is either an acceptable token and a preferred token.
BS: acceptable = ("a"/"A") ("c"/"C") ("c"/"C") ("e"/"E") ("p"/"P") ("t"/"T") **LS: acceptable** = ("a"/"A") ("c"/"C") ("c"/"C") ("e"/"E") ("p"/"P") ("t"/"T") [("a"/"A") ("b"/"B") ("l"/"L") ("e"/"E")]	
	An acceptable token uses the text "ACCEPT" (case insensitive). In the long syntax, the text "Acceptable" (case insensitive) may be used instead.
BS: preferred = ("p"/"P") ("r"/"R") ("e"/"E") ("f"/"F") ("e"/"E") ("r"/"R") **LS: preferred** = ("p"/"P") ("r"/"R") ("e"/"E") ("f"/"F") ("e"/"E") ("r"/"R") [("r"/"R") ("e"/"E") ("d"/"D")]	
	A preferred token uses the text "PREFER" (case insensitive). In the long syntax, the text "Preferred" (case insensitive) may be used instead.
BS/LS: conceptFilterConstraint = "{{" ws ("c" / "C") ws conceptFilter *(ws "," ws conceptFilter) ws "}}"	
	A concept filter constraint is a constraint used to filter the concepts in the result set, according to whether or not the concept matches the given conditions. A concept filter constraint is always enclosed in double curly braces. Within these braces, it starts with the letter 'C' followed by one or more constraint filters.
BS/LS: conceptFilter = definitionStatusFilter / moduleFilter / effectiveTimeFilter / activeFilter	
	A concept filter is either a definition status filter, a module filter, an effective time filter or an active filter.
BS/LS: definitionStatusFilter = definitionStatusIdFilter / definitionStatusTokenFilter	
	A definition status filter is constraint that either filters the results of a query, based on each concept's definition status identifier or a token.
BS/LS: definitionStatusIdFilter = definitionStatusIdKeyword ws booleanComparisonOperator ws (subExpressionConstraint / eclConceptReferenceset)	
	A definition status filter is a constraint that filters the results of a query, based on whether or not each concept's definition status matches a given identifier. The filter starts with the keyword "definitionStatusId", followed by a boolean comparison operator and either a subexpression constraint or a set of concept references that are a subtype of 900000000000444006 |Definition status|.
BS/LS: definitionStatusIdKeyword = ("d"/"D") ("e"/"E") ("f"/"F") ("i"/"I") ("n"/"N") ("i"/"I") ("t"/"T") ("i"/"I") ("o"/"O") ("n"/"N") ("s"/"S") ("t"/"T") ("a"/"A") ("t"/"T") ("u"/"U") ("s"/"S") ("i"/"I") ("d"/"D")	
	The definition status id keyword is the text "definitionStatusId" (in any combination of upper or lower case).
BS/LS: definitionStatusTokenFilter = definitionStatusKeyword ws booleanComparisonOperator ws (definitionStatusToken / definitionStatusTokenSet)	
	A definition status filter is a constraint that filters the results of a query, based on whether or not each concept's definition status matches a given token.
BS/LS: definitionStatusKeyword = ("d"/"D") ("e"/"E") ("f"/"F") ("i"/"I") ("n"/"N") ("i"/"I") ("t"/"T") ("i"/"I") ("o"/"O") ("n"/"N") ("s"/"S") ("t"/"T") ("a"/"A") ("t"/"T") ("u"/"U") ("s"/"S")	
	The definition status keyword is the text "definitionStatus" (in any combination of upper or lower case).
BS/LS: definitionStatusToken = primitiveToken / definedToken	
	A definition status token is either a primitive token or a defined token.
BS/LS: definitionStatusTokenSet = "(" ws definitionStatusToken *(mws definitionStatusToken) ws ")"	
	A definition status token set consists of one or more definition status tokens separated by mandatory white space and enclosed in brackets.
BS/LS: primitiveToken = ("p"/"P") ("r"/"R") ("i"/"I") ("m"/"M") ("i"/"I") ("t"/"T") ("i"/"I") ("v"/"V") ("e"/"E")	
	A primitive token represents the definition status 900000000000074008 |Primitive| using the text "primitive" (in any combination of upper and lower case characters).
BS/LS: definedToken = ("d"/"D") ("e"/"E") ("f"/"F") ("i"/"I") ("n"/"N") ("e"/"E") ("d"/"D")	

	A defined token represents the definition status 900000000000073002 |Defined| using the text "defined" (in any combination of upper and lower case characters).
BS/LS: moduleFilter =moduleIdKeyword ws booleanComparisonOperator ws (subExpressionConstraint / eclConceptReferenceSet)	
	A module filter is a constraint that filters the results of a query based on the module to which each concept belongs. The filter starts with the keyword "moduleId", followed by a boolean comparison operator and either a subexpression constraint or a set of concept references that are a subtype of 900000000000443000 |Module|.
BS/LS: moduleIdKeyword = ("m"/"M") ("o"/"O") ("d"/"D") ("u"/"U") ("l"/"L") ("e"/"E") ("i"/"I") ("d"/"D")	
	The module id keyword is the text "moduleId" (in any combination of upper or lower case).
BS/LS: effectiveTimeFilter = effectiveTimeKeyword ws timeComparisonOperator ws (timeValue / timeValeSet)	
	An effective time filter is a constraint that filters the results of a query based on the effective time assigned to each concept.
BS/ LS: effectiveTimeKeyword = ("e"/"E") ("f"/"F") ("f"/"F") ("e"/"E") ("c"/"C") ("t"/"T") ("i"/"I") ("v"/"V") ("e"/"E") ("t"/"T") ("i"/"I") ("m"/"M") ("e"/"E")	
	The effective time keyword is the text "effectiveTime" (in any combination of upper or lower case).
BS/LS: timeValue = QM [year month day] QM	
	A time value is a 8 digit string that represents the year, month and day of a specific date.
BS/LS: timeValueSet = "(" ws timeValue *(mws timeValue) ws ")"	
	A time value set consists of one or more time values separated by mandatory white space and enclosed in brackets.
BS/LS: year = digitNonZero digit digit digit	
	A year is a 4 digit string starting with a non-zero digit.
BS/LS: month = "01" / "02" / "03" / "04" / "05" / "06" / "07" / "08" / "09" / "10" / "11" / "12"	
	A month is a 2 digit string from "01" to "12" that represents a specific month of the year (e.g. "01" represents January)
BS/LS: day = "01" / "02" / "03" / "04" / "05" / "06" / "07" / "08" / "09" / "10" / "11" / "12" / "13" / "14" / "15" / "16" / "17" / "18" / "19" / "20" / "21" / "22" / "23" / "24" / "25" / "26" / "27" / "28" / "29" / "30" / "31"	
	A day is a 2 digit string from "01" to "31" that represents a specific day within a month of a year.
BS/LS: activeFilter = activeKeyword ws booleanComparisonOperator ws activeValue	
	An active filter is a constraint that filters the results of a query based on the active status of each concept
BS/LS: activeKeyword = ("a"/"A") ("c"/"C") ("t"/"T") ("i"/"I") ("v"/"V") ("e"/"E")	
	The active keyword is the text "active" (in any combination of upper or lower case).
BS/LS: activeValue = activeTrueValue / activeFalseValue	
	An active value represents the active status of a concept, and is either true (i.e. the concept is active) or false (i.e. the concept is inactive).
BS/LS: activeTrueValue = "1" / "true"	
	An active true value is a value that represents an active concept. This value is either "1" or "true".
BS/LS: activeFalseValue = "0" / "false"	
	An active false value is a value that represents an inactive concept. This value is either "0" or "false".
BS/LS: memberFilterConstraint = "{{" ws ("m" / "M") ws memberFilter *(ws "," ws memberFilter) ws "}}"	
	A member filter constraint is a constraint used to filter the rows in one or more result sets, according to values of particular fields. A member filter constraint is always surrounded by double curly braces. Within these braces, it starts with the letter 'M' followed by one or more member filters.
BS/LS: memberFilter = memberFieldFilter / moduleFilter / effectiveTimeFilter / activeFilter	
	A member filter is either a member field filter, a module filter, an effective time filter, or an active filter.

BS/LS: memberFieldFilter = refsetFieldName ws (expressionComparisonOperator ws subExpressionConstraint / numericComparisonOperator ws "#" numericValue / stringComparisonOperator ws (typedSearchTerm / typedSearchTermSet) / booleanComparisonOperator ws booleanValue / ws timeComparisonOperator ws (timeValue / timeValueSet))	
	A member field filter always has three parts - (1) the reference set field name, (2) a comparison operator, and (3) the criteria on which to match the field's value. If the refset field is of type SNOMED CT concept, then an expression comparison operator is used, followed by a subexpression constraint. If the refset field is a numeric type, then a numeric comparison operator is used, followed by a hash symbol ("#") and a numeric value. If the refset field is of type string, then a string comparison operator is used, followed by a typed search term or a typed search term set. If the refset field is of type boolean, then a boolean comparison operator is used, followed by a boolean value. And if the refset field is of type dateTime, then a time comparison operator is used, followed by a time value or time value set.
BS/LS: historySupplement = "{{" ws "+" ws historyKeyword [historyProfileSuffix / ws historySubset] ws "}}"	
	A history supplement augments the results of the expression constraint with relevant inactive concepts. A history supplement is always surrounded by double curly braces. Within these braces, it starts with a plus symbol (i.e. "+"), followed by the history keyword. The history keyword is optionally followed by either a profile suffix, or a history subset.
BS/LS: historyKeyword = ("h"/"H") ("i"/"I") ("s"/"S") ("t"/"T") ("o"/"O") ("r"/"R") ("y"/"Y")	
	The history keyword is the word "HISTORY" (case insensitive).
BS/LS: historyProfileSuffix = historyMinimumSuffix / historyModerateSuffix / historyMaximumSuffix	
	A history profile suffix is either the suffix for history minimum, history moderate or history maximum.
BS/LS: historyMinimumSuffix = ("-"/"_")("m"/"M") ("i"/"I") ("n"/"N")	
	The history minimum suffix is "-MIN" (case insensitive). The suffix may start with either a hyphen (i.e. "-") or an underscore (i.e. "_").
BS/LS: historyModerateSuffix = ("-"/"_") ("m"/"M") ("o"/"O") ("d"/"D")	
	The history moderate suffix is "-MOD" (case insensitive). The suffix may start with either a hyphen (i.e. "-") or an underscore (i.e. "_").
BS/LS: historyMaximumSuffix = ("-"/"_") ("m"/"M") ("a"/"A") ("x"/"X")	
	The history maximum suffix is "-MAX" (case insensitive). The suffix may start with either a hyphen (i.e. "-") or an underscore (i.e. "_").
BS/LS: historySubset = "(" ws expressionConstraint ws ")"	
	A history subset is an expression constraint that defines a set of historical association reference sets, surrounded by round brackets. Only descendants of 900000000000522004 |Historical association reference set| may be included in a history subset.
BS/LS: numericValue = ["-"/"+"] (decimalValue / integerValue)	
	A numeric value is either an integer or a decimal. Positive numbers optionally start with a plus sign ("+"), while negative integers begin with a minus sign ("-").
BS/LS: stringValue = 1*(anyNonEscapedChar / escapedChar)	
	A string value includes one or more of any printable ASCII characters enclosed in quotation marks. Quotes and backslash characters within the string must be preceded by the escape character ("\").
BS/LS: integerValue = digitNonZero *digit / zero	
	An integer value is either starts with a non-zero digit followed by zero to many additional digits, or is the integer zero itself.
BS/LS: decimalValue = integerValue "." 1*digit	
	A decimal value starts with an integer. This is followed by a decimal point and one to many digits.
BS/LS: booleanValue = true / false	

	A boolean value is either true or false.
BS/LS: true = ("t"/"T") ("r"/"R") ("u"/"U") ("e"/"E")	
	A boolean value of true is represented by the word "true" (case insensitive).
BS/LS: false = ("f"/"F") ("a"/"A") ("l"/"L") ("s"/"S") ("e"/"E")	
	A boolean value of false is represented by the word "false" (case insensitive).
BS/LS: nonNegativeIntegerValue = (digitNonZero *digit) / zero	
	A non-negative integer value (i.e. positive integers or zero), without a preceding plus sign ("+").
BS/LS: sctId = digitNonZero 5*17(digit)	
	A SNOMED CT id is used to represent an attribute id or a concept id. The initial digit may not be zero. The smallest number of digits is six, and the maximum is 18.
BS/LS: ws = *(SP / HTAB / CR / LF / comment)	
	Optional whitespace characters (space, tab, carriage return, linefeed or a comment) are ignored everywhere in the expression except: 1. Whitespace within a conceptId is an error. **Note:** Whitespace before or after the last digit of a valid Identifier is ignored. 2. Non-consecutive spaces within a term are treated as a significant character of the term. **Note:** Whitespace before the first or after the last non-whitespace character of a term is ignored 3. Whitespace within the quotation marks of a concrete value is treated as a significant character.
BS/LS: mws = 1*(SP / HTAB / CR / LF / comment)	
	Mandatory whitespace (i.e. space, tab, carriage return, linefeed or a comment) is required after certain keywords, including "And" and "Or".
BS/LS: comment = "/*" *(nonStarChar / starWithNonLSlash) "*/"	
	A comment, which provides additional human-readable details about the expression constraint. Comments begin with a forward slash directly followed by a star (i.e. "/*") and end with a star directly followed by a forward slash (i.e. "*/").
BS/LS: nonStarChar = SP / HTAB / CR / LF / %x21-29 / %x2B-7E /UTF8-2 / UTF8-3 / UTF8-4	
	A character that is not a star (i.e. not %x2A).
BS/LS: starWithNonLSlash = %x2A nonLSlash	
	A star (i.e. "*") followed by a character that is not a forward slash (i.e. not "/").
BS/LS: nonLSlash = SP / HTAB / CR / LF / %x21-2E / %x30-7E /UTF8-2 / UTF8-3 / UTF8-4	
	A character that is not a forward slash (i.e. not "/").

BS/LS: SP = %x20	
	Space character.
BS/LS: HTAB = %x09	
	Tab character.
BS/LS: CR = %x0D	
	Carriage return character.
BS/LS: LF = %x0A	
	Line feed character.
BS/LS: QM = %x22	
	Quotation mark character.
BS/LS: BS = %x5C ; back slash	
	BS represents the backslash character "\".
BS/LS: star = %x2A ; asterisk	
	Star represents an asterisk "*".
BS/LS: digit = %x30-39	
	Any digit 0 through 9.
BS/LS: zero = %x30	
	The digit 0.
BS/LS: digitNonZero = %x31-39	
	Digits 1 through 9, but excluding 0. The first character of a concept identifier is constrained to a digit other than zero.
BS/LS: nonwsnonpipe= %x21-7B / %x7D-7E / UTF8-2 / UTF8-3 / UTF8-4	
	Non whitespace (and non pipe) includes printable ASCII characters (these are also valid UTF8 characters encoded as one octet) and also includes all UTF8 characters encoded as 2- 3- or 4-octet sequences. It excludes space (which is %x20) and the pipe character "
BS/LS: anyNonEscapedChar = SP / HTAB / CR / LF / %x20-21 / %x23-5B / %x5D-7E / UTF8-2 / UTF8-3 / UTF8-4	
	anyNonEscapedChar includes any printable ASCII characters which do not need to be preceded by an escape character (i.e. "\"). This includes valid UTF8 characters encoded as one octet and all UTF8 characters encoded as 2, 3 or 4 octet sequences. It does, however, exclude the quotation mark (") and the backslash (). See RFC 3629 (UTF-8, a transformation format of ISO 10646 authored by the Network Working Group).

BS/LS: **escapedChar** = BS QM / BS BS	
	The double quotation mark and the back slash character must both be escaped within a string-based concrete value by preceding them with a back slash.
BS/LS: **escapedWildChar** = BS QM / BS BS / BS star	
	An escapedWildChar is one of the characters that must be escaped in a wildcard search term (i.e. " or \ or *), preceded by a backslash (i.e. \). The character sequence is therefore either \" or \\ or *.
BS/LS: **nonwsNonEscapedChar** = %x21 / %x23-5B / %x5D-7E / UTF8-2 / UTF8-3 / UTF8-4	
	A nonwsNonEscapedChar is any printable ASCII, UTF8-2, UTF8-3 or UTF8-4 character, excluding double quotes ("), backslash (\), and space ( ).
BS/LS: **alpha** = %x41-5A / %x61-7A	
	An alpha is any uppercase or lowercase character from "A" to "Z" (and "a" to "z") inclusive.
BS/LS: **dash** = %x2D	
	A dash is a hyphen (i.e. "-").
BS/LS: **UTF8-2** = %xC2-DF UTF8-tail	
	UTF8 characters encoded as 2-octet sequences.
BS/LS: **UTF8-3** = %xE0 %xA0-BF UTF8-tail / %xE1-EC 2(UTF8-tail) / %xED %x80-9F UTF8-tail / %xEE-EF 2(UTF8-tail)	
	UTF8 characters encoded as 3-octet sequences.
BS/LS: **UTF8-4** = %xF0 %x90-BF 2(UTF8-tail) / %xF1-F3 3(UTF8-tail) / %xF4 %x80-8F 2(UTF8-tail)	
	UTF8 characters encoded as 4-octet sequences.
BS/LS: **UTF8-tail** = %x80-BF	
	UTF8 characters encoded as 8-octet sequences.

5.4 Order of Operation

This section explains the correct order of operation for unary operators, binary operators, filters and supplements.

Unary Operators

Unary operators (e.g. descendantOf, descendantOrSelfOf, ancestorOf, ancestorOrSelfOf, memberOf) are applied from inside to out (i.e. from right to left). For example, when the following expression constraint is processed, the memberOf operator is applied first to the Example problem list concepts reference set, and then the descendants of the referenced components are determined.

```
< ^ 700043003 |Example problem list concepts reference set|
```

Binary Operators

Whenever potential ambiguity in binary operator precedence may occur, round brackets must be used to clearly disambiguate the order in which these operators are applied. For example, the following expression constraint is not valid:

```
< 19829001 |Disorder of lung| OR ^ 700043003 |Example problem list concepts reference set|
  MINUS ^ 450976002 |Disorders and diseases reference set for GP/FP reason for encounter|
```

And must be expressed using brackets, as either:

```
(< 19829001 |Disorder of lung| OR ^ 700043003 |Example problem list concepts reference set| )
  MINUS ^ 450976002 |Disorders and diseases reference set for GP/FP reason for encounter|
```

or:

```
< 19829001 |Disorder of lung| OR (^ 700043003 |Example problem list concepts reference set|
  MINUS ^ 450976002 |Disorders and diseases reference set for GP/FP reason for encounter| )
```

When multiple exclusion operators (i.e. 'minus') are applied, brackets are similarly required. For example, the following expression constraint is not valid:

```
< 19829001 |Disorder of lung| MINUS ^ 700043003 |Example problem list concepts reference set|
  MINUS ^ 450976002 |Disorders and diseases reference set for GP/FP reason for encounter|
```

And must be expressed using brackets, as either:

```
(< 19829001 |Disorder of lung| MINUS ^ 700043003 |Example problem list concepts reference set| )
  MINUS ^ 450976002 |Disorders and diseases reference set for GP/FP reason for encounter|
```

or:

```
< 19829001 |Disorder of lung| MINUS (^ 700043003 |Example problem list concepts reference set|
  MINUS ^ 450976002 |Disorders and diseases reference set for GP/FP reason for encounter| )
```

However, when only a single binary operator is used, or when all binary operators are either conjunction (i.e. 'and') or disjunction (i.e. 'or'), brackets are not required. For example, all of the following expression constraints are valid without brackets:

```
< 19829001 |Disorder of lung| AND ^ 700043003 |Example problem list concepts reference set|
```

```
< 19829001 |Disorder of lung| OR ^ 700043003 |Example problem list concepts reference set|
```

```
< 19829001 |Disorder of lung| MINUS ^ 700043003 |Example problem list concepts reference set|
```

```
< 19829001 |Disorder of lung| OR ^ 700043003 |Example problem list concepts reference set|
  OR ^ 450976002 |Disorders and diseases reference set for GP/FP reason for encounter|
```

```
< 19829001 |Disorder of lung| AND ^ 700043003 |Example problem list concepts reference set|
  AND ^ 450976002 |Disorders and diseases reference set for GP/FP reason for encounter|
```

Please note that unary operators are always applied before binary operators.

Filter Constraints

Filter constraints (e.g. concept, description, or member filters) apply only to the sub-expression constraint part that is directly to the left of the filter.

For example, the following expression constraint will apply the term filter to only the descendants or self of 415582006 |Stenosis|. This expression constraint will match descendants of 404684003 |Clinical finding| with a finding site that is a descendant or self of 39057004 |Pulmonary valve structure|, and an associated morphology that is any descendant or self of 415582006 |Stenosis| which has a description matching the term "insufficiency". Therefore, the concept 123801008 |Heart valve stenosis and regurgitation (disorder)| will match this expression constraint because it has the associated morphology 708027006 |Valvular stenosis with valvular insufficiency|.

```
< 404684003 |Clinical finding| :
    363698007 |Finding site| =<< 39057004 |Pulmonary valve structure| ,
    116676008 |Associated morphology| =<< 415582006 |Stenosis| {{ term = "insufficiency" }}
```

To apply a filter to a sub-expression constraint, which includes a refinement or binary operators, the subexpression must be enclosed in brackets. For example, the following expression constraint will find all the descendants of clinical finding, with a finding site that is a descendant or self of 39057004 |Pulmonary valve structure| and an associated morphology that is a descendant or self of 415582006 |Stenosis|, and will then match only those clinical finding concepts that have a description that matches the term "insufficiency". Therefore, the concept 123801008 |Heart valve stenosis and regurgitation (disorder)| will **not** match this expression constraints, as it does not have a description that matches the term "insufficiency".

```
(< 404684003 |Clinical finding| :
    363698007 |Finding site| =<< 39057004 |Pulmonary valve structure| ,
    116676008 |Associated morphology| =<< 415582006 |Stenosis| ) {{ term = "insufficiency" }}
```

History Supplements

History supplements are applied only to the sub-expression constraint part that is directly to its left, after any filter constraints on this sub-expression constraint part have been applied.

For example, the following expression constraint will match all concepts that are **both** an active member of the 734139008 |Anatomy structure and part association reference set| **and** also either an active member of the 734138000 |Anatomy structure and entire association reference set| or an inactive concept associated with an active member of the 734138000 |Anatomy structure and entire association reference set| via the 900000000000527005 |SAME AS association reference set|. Because all active members of the 734139008 |Anatomy structure and part association reference set| are active, there will be no inactive concepts in the result set.

```
^ 734139008 |Anatomy structure and part association reference set|
  AND ^ 734138000 |Anatomy structure and entire association reference set|
    {{ + HISTORY ( 900000000000527005 |SAME AS association reference set| ) }}
```

To apply the history supplement to the entire sub-expression constraint above, the sub-expression constraint must be enclosed in round brackets. For example, the following expression constraint will match concepts that are **both** members of the 734139008 |Anatomy structure and part association reference set| **and** also members of the 734138000 |Anatomy structure and entire association reference set|; and it will also match on any inactive concept that is associated via a 900000000000527005 |SAME AS association reference set| to a member of both reference sets.

```
( ^ 734139008 |Anatomy structure and part association reference set|
  AND ^ 734138000 |Anatomy structure and entire association reference set| )
    {{ + HISTORY ( 900000000000527005 |SAME AS association reference set| ) }}
```

5.5 Character Collation for Term Filters

⚠ This page is published as **Draft for Trial Use**. The recommendations on this page will be reviewed and may be updated following feedback from implementation experiences.

To promote consistency between implementations of ECL, the following collation principles are recommended:

- **Search and match** - The default behaviour of a system implementing ECL queries with term filters, is to use locale-specific asymmetric searching at the secondary comparison strength level -as specified in the Unicode Technical Standard #10 - Unicode Collation Algorithm. This means that the search is, by default, case insensitive, with some language-specific character normalization behaviour.
  - *Asymmetric*: Asymmetric searches require characters in the query that are unmarked (i.e. the 'base letters') to match characters in the target that are either *marked* or *unmarked* (with the same base letter). However, a character in the query that is *marked* will only match a character in the target that is *marked* in the same way.
  - *Secondary strength*: Searches with a strength of secondary will only consider level 1 differences (e.g. "d" vs "e") and level 2 differences (e.g. "e" vs "é" in English). However, level 3 differences (e.g. "e" vs "E") are not considered. This provides the same effect as queries being case insensitive. For example, in English, "e" in the query will match both "e" and "E" in the target; and "E" in the query will similarly match both "e" and "E" in the target.
- **Language customizations** - Locale-based customizations of the standard are specified in the Unicode Common Locale Data Repository (CLDR). The unicode CLDR specifies the characters that are considered to be 'marked' variants of the base letters, identical base letters, and/or contractions in each specified language. The description terms in the substrate should be indexed separately for each language supported.

For example, the following search behaviour is expected in the locales specified below.

- In **English**, **Swedish** and **Danish**, the following search behaviour is expected:
  Note: No customizations are made in these 3 locales for the characters used in these searches. Therefore, the CLDR root collation order is used.

Search Term	Target Matches	Target does NOT Match
resume	resume, Resume, RESUME, résumé, rèsumè, Résumé, RÉSUMÉ, …	-
Resume	resume, Resume, RESUME, résumé, rèsumè, Résumé, RÉSUMÉ, …	-
résumé	résumé, Résumé, RÉSUMÉ, …	resume, Resume, RESUME, ...
Résumé	résumé, Résumé, RÉSUMÉ, …	resume, Resume, RESUME, ...

- In **English**, the following search behaviour is expected (based on the CLDR 'en' locale, which uses the CLDR root collation order):

Search Term	Target Matches	Target does NOT Match
sjogren	sjogren, Sjogren, SJOGREN, sjögren, Sjögren, SJÖGREN, sjøgren, Sjøgren, SJØGREN, ...	-
sjögren	sjögren, Sjögren, SJÖGREN, ...	sjogren, Sjogren, SJOGREN, sjøgren, Sjøgren, SJØGREN, ...
Angstrom	angstrom, Angstrom, ANGSTROM, ångström, Ångström, ÅNGSTRÖM, ångstrøm, Ångstrøm, ÅNGSTRØM, ...	ångstrœm, Ångstrœm, ÅNGSTRŒM, ...
Ångström	ångström, Ångström, ÅNGSTRÖM, ...	angstrom, Angstrom, ANGSTROM, ångstrøm, Ångstrøm, ÅNGSTRØM, ...
Ångstrøm	ångstrøm, Ångstrøm, ÅNGSTRØM, ...	angstrom, Angstrom, ANGSTROM, ångström, Ångström, ÅNGSTRÖM, ...
aangstrøm	aangstrøm, Aangstrøm, AANGSTRØM, ...	angstrom, Angstrom, ANGSTROM, ångström, Ångström, ÅNGSTRÖM, ångstrøm, Ångstrøm, ÅNGSTRØM, ångstrœm, Ångstrœm, ÅNGSTRŒM, ...

- In **Swedish**, the following search behaviour is expected (based on the customizations in the CLDR 'sv' locale):

Search Term	Target Matches	Target does NOT Match
sjogren	sjogren, Sjogren, SJOGREN, ...	sjögren, Sjögren, SJÖGREN, sjøgren, Sjøgren, SJØGREN, ...
sjögren	sjögren, Sjögren, SJÖGREN, sjøgren, Sjøgren, SJØGREN, ...	sjogren, Sjogren, SJOGREN , ...
Angstrom	angstrom, Angstrom, ANGSTROM, ...	ångström, Ångström, ÅNGSTRÖM, ångstrøm, Ångstrøm, ÅNGSTRØM, ångstrœm, Ångstrœm, ÅNGSTRŒM, aangström, Aangström, AANGSTRÖM, ...
Ångström	ångström, Ångström, ÅNGSTRÖM, ångstrøm, Ångstrøm, ÅNGSTRØM, ångstrœm, Ångstrœm, ÅNGSTRŒM, ...	angstrom, Angstrom, ANGSTROM, aangström, Aangström, AANGSTRÖM, ...
Ångstrøm	ångstrøm, Ångstrøm, ÅNGSTRØM, ...	angstrom, Angstrom, ANGSTROM, ångström, Ångström, ÅNGSTRÖM, ångstrœm, Ångstrœm, ÅNGSTRŒM, ...
aangstrøm	aangstrøm, Aangstrøm, AANGSTRØM, ...	angstrom, Angstrom, ANGSTROM, ångström, Ångström, ÅNGSTRÖM, ångstrøm, Ångstrøm, ÅNGSTRØM, ångstrœm, Ångstrœm, ÅNGSTRŒM, ...

- And in **Danish**, the following search behaviour is expected (based on the customizations in the CLDR 'da' locale):

Search Term	Target Matches	Target does NOT Match
sjogren	sjogren, Sjogren, SJOGREN, ...	sjögren, Sjögren, SJÖGREN, sjøgren, Sjøgren, SJØGREN, ...
sjögren	sjögren, Sjögren, SJÖGREN, ...	sjogren, Sjogren, SJOGREN, sjøgren, Sjøgren, SJØGREN, ...
Angstrom	angstrom, Angstrom, ANGSTROM, ...	ångström, Ångström, ÅNGSTRÖM, ångstrøm, Ångstrøm, ÅNGSTRØM, ångstrœm, Ångstrœm, ÅNGSTRŒM, aangstrøm, Aangstrøm, AANGSTRØM ...
Ångström	ångström, Ångström, ÅNGSTRÖM, aangström, Aangström, AANGSTRÖM, ...	angstrom, Angstrom, ANGSTROM, ångstrøm, Ångstrøm, ÅNGSTRØM, ångstrœm, Ångstrœm, ÅNGSTRŒM, ...
Ångstrøm	ångstrøm, Ångstrøm, ÅNGSTRØM, ångström, Ångström, ÅNGSTRÖM, aangstrøm, Aangstrøm, AANGSTRØM, aangström, Aangström, AANGSTRÖM, ...	angstrom, Angstrom, ANGSTROM, ångstrœm, Ångstrœm, ÅNGSTRŒM, ...
aangstrøm	ångstrøm, Ångstrøm, ÅNGSTRØM, ångström, Ångström, ÅNGSTRÖM, aangstrøm, Aangstrøm, AANGSTRØM, aangström, Aangström, AANGSTRÖM, ...	angstrom, Angstrom, ANGSTROM, ångstrœm, Ångstrœm, ÅNGSTRŒM, ...

6. Examples

The examples in this section illustrate the syntaxes proposed in Section 5.

6.1 Simple Expression Constraints

The simplest type of expression constraint contains a single concept optionally preceded by an expression constraint operator and/or membership function. Expression constraint operators (e.g. descendant of) traverse the hierarchical relationships in SNOMED CT to return the set of concepts that are directly or transitively connected to the focus concept. Membership functions return the set of concepts referenced by a reference set.

In this section we consider some of these simple examples.

Self

If no expression constraint operator or membership function is applied, the expression constraint is satisfied only by the specified concept. For example, the expression constraint below is satisfied only by the concept 404684003 |Clinical finding|.

```
404684003 |Clinical finding|
```

Please note that this expression constraint is equivalent to an expression that looks the same but is written in SNOMED CT Compositional Grammar.

Descendant of

A single 'less than' sign (i.e. "<") indicates that the expression constraint is satisfied by all descendants of the specified concept. The expression constraint below evaluates to the set of all subtypes (both direct children and transitive subtypes) of 404684003 |Clinical finding|, using the brief syntax.

```
< 404684003 |Clinical finding|
```

Using the long syntax, the above expression constraint may be represented as:

```
descendantOf 404684003 |Clinical finding|
```

The descendantOf function is primarily used on concepts, which serve as the 'grouper' of a set of values (e.g. |Clinical finding (finding)|, |Severities (qualifier value)|, |Unit (qualifier value)|). The descendantOf function may also be applied to other concepts, or to nested expression constraints (as discussed in 6.7 Nested Expression Constraints).

Descendant or Self of

Two consecutive 'less than' signs (i.e. "<<") indicates that the expression constraint is satisfied by all descendants of the specified concept plus the specified concept itself. The expression constraint below evaluates to the set of descendants of 73211009 |Diabetes mellitus|, plus the concept 73211009 |Diabetes mellitus| itself.

```
<< 73211009 |Diabetes mellitus|
```

Using the long syntax, the above expression constraint may be represented as:

```
descendantOrSelfOf 73211009 |Diabetes mellitus|
```

The descendantOrSelfOf function is primarily used for attribute values, which refer to a specific clinical value (e.g. 73211009 |Diabetes mellitus|, 73761001 |Colonoscopy|, 385055001 |Tablet dose form|), but any specialization of this value is also acceptable. The descendantOrSelfOf function may also be applied to other concepts, or to nested expression constraints (as discussed in 6.7 Nested Expression Constraints).

Child of

A 'less than' sign directly followed by an exclamation mark (i.e. "<!") indicates that the expression constraint is satisfied by the set of proximal children of the specified concept. The children of a concept are those concepts that are the source of a non-redundant 116680003 |is a| relationship whose target is the given concept. The expression constraint below, represented using the brief syntax, evaluates to the set of immediate children of the concept 404684003 |Clinical finding|.

```
<! 404684003 |Clinical finding|
```

Using the long syntax, the above expression constraint may be represented as:

```
childOf 404684003 |Clinical finding|
```

Please note that the childOf function may only be executed against a finite and pre-classified substrate, and that the results of this function are specific to the substrate used. The childOf function may also be applied to nested expression constraints (as discussed in 6.7 Nested Expression Constraints).

Child or Self of

Two consecutive 'less than' signs directly followed by an exclamation mark (i.e. "<<!") indicates that the expression constraint is satisfied by the set of proximal children of the specified concept plus the specified concept itself. The children of a concept are those concepts that are the source of a non-redundant 116680003 |is a| relationship whose target is the given concept. The expression constraint below, represented using the brief syntax, evaluates to the set of immediate children of the concept 404684003 |Clinical finding|, plus the concept 404684003 |Clinical finding| itself.

```
<<! 404684003 |Clinical finding|
```

Using the long syntax, the above expression constraint may be represented as:

```
childOrSelfOf 404684003 |Clinical finding|
```

Please note that the childOrSelfOf function may only be executed against a finite and pre-classified substrate, and that the results of this function are specific to the substrate used. The childOrSelfOf function may also be applied to nested expression constraints (as discussed in 6.7 Nested Expression Constraints).

Ancestor of

A single 'greater than' sign (i.e. ">") indicates that the expression constraint is satisfied by all ancestors of the specified concept. The expression constraint below, using the brief syntax, evaluates to the set of all supertypes (both direct parents and transitive supertypes) of 40541001 |Acute pulmonary edema|:

```
> 40541001 |Acute pulmonary edema|
```

Using the long syntax, the above expression constraint may be represented as:

```
ancestorOf 40541001 |Acute pulmonary edema|
```

Please note that the ancestorOf function may also be applied to nested expression constraints (as discussed in 6.7 Nested Expression Constraints).

Ancestor or Self of

Two consecutive 'greater than' signs (i.e. ">>") indicates that the expression constraint is satisfied by all ancestors of the specified concept plus the specified concept itself. The expression constraint below evaluates to the set of ancestors of 40541001 |Acute pulmonary edema|, plus the concept 40541001 |Acute pulmonary edema|.

```
>> 40541001 |Acute pulmonary edema|
```

Using the long syntax, the above expression constraint may be represented as:

```
ancestorOrSelfOf 40541001 |Acute pulmonary edema|
```

Please note that the ancestorOrSelfOf function may also be applied to nested expression constraints (as discussed in 6.7 Nested Expression Constraints).

Parent of

A 'greater than' sign directly followed by an exclamation mark (i.e. ">!") indicates that the expression constraint is satisfied by the set of proximal parents of the specified concept. The parents of a concept are those concepts that are the target of a non-redundant |is a| relationship whose source is the given concept. The expression constraint below, represented using the brief syntax, evaluates to the set of immediate parents of the concept 40541001 |Acute pulmonary edema|.

```
>! 40541001 |Acute pulmonary edema|
```

Using the long syntax, the above expression constraint may be represented as:

```
parentOf 40541001 |Acute pulmonary edema|
```

Please note that the parentOf function should only be executed against a finite and pre-classified substrate, and that the results of this function are specific to the substrate used. The parentOf function may also be applied to nested expression constraints (as discussed in 6.7 Nested Expression Constraints).

Parent or Self of

Two consecutive 'greater than' signs directly followed by an exclamation mark (i.e. ">>!") indicates that the expression constraint is satisfied by the set of proximal parents of the specified concept plus the specified concept itself. The parents of a concept are those concepts that are the target of a non-redundant |is a| relationship whose source is the given concept. The expression constraint below, represented using the brief syntax, evaluates to the set of immediate parents of the concept 40541001 |Acute pulmonary edema|, plus the concept 40541001 |Acute pulmonary edema| itself.

```
>>! 40541001 |Acute pulmonary edema|
```

Using the long syntax, the above expression constraint may be represented as:

```
parentOrSelfOf 40541001 |Acute pulmonary edema|
```

Please note that the parentOrSelfOf function should only be executed against a finite and pre-classified substrate, and that the results of this function are specific to the substrate used. The parentOrSelfOf function may also be applied to nested expression constraints (as discussed in 6.7 Nested Expression Constraints).

Member of

The memberOf function (by default) evaluates to the set of concepts that are referenced by the given reference set (i.e. the set of referencedComponentIds). Please note that this function may be applied only to reference sets whose referenced components are concepts. The SNOMED CT Expression Constraint Language does not support use of the memberOf function on reference sets whose referencedComponents are not concepts (i.e. descriptions or relationships).

The memberOf function is represented in the brief syntax using a 'caret' character (i.e. "^") and is usually followed by a single concept id for a concept-based reference set. For example, the following expression constraint is satisfied by the set of concepts which are members of 700043003 |Example problem list concepts reference set|:

```
^ 700043003 |Example problem list concepts reference set|
```

Using the long syntax the expression constraint is represented as:

```
memberOf 700043003 |Example problem list concepts reference set|
```

The expression constraints above both return the values in the referencedComponentId field of the given reference sets. However, it is also possible to specify one or more fields, whose values will be returned, by including the relevant field names is square brackets after the memberOf operator ("^" or "memberOf"). For example, the following expression constraint is equivalent to the brief syntax example above.

```
^ [referencedComponentId] 700043003 |Example problem list concepts reference set|
```

The value of other fields can also be returned by an expression constraint[1]. For example, the following expression constraint will return the targetComponentId values (i.e. the 'Entire' anatomy concepts) from the 734138000 |Anatomy structure and entire association reference set|.

```
^ [targetComponentId] 734138000 |Anatomy structure and entire association reference set|
```

It is also possible to return the values of more than one field in a reference set (e.g. a pair or tuple of values).[2][3][4] For example, to return both the source and target of the 816210007 |SNOMED CT to MedDRA simple map reference set|, the following expression constraint could be used:

```
^ [referencedComponentId, mapTarget] 816210007 |SNOMED CT to MedDRA simple map reference set|
```

To return all the non-metadata fields of a referenceSet (i.e. the values of the referencedComponentId and additional fields), a wildcard (i.e. "*" in the brief syntax, and "*" or "Any" in the long syntax) can be used. For example, the following expression constraint will return the referencedComponentId, mapGroup, mapPriority, mapRule, mapAdvice, mapTarget and correlationId for each row of the 447562003 |ICD-10 complex map reference set|.

```
^ [*] 447562003 |ICD-10 complex map reference set|
```

For more information on the use of reference set field names in ECL, please refer to Appendix E - Reference Set Fields.

Please note that it is also possible to apply the memberOf function to an expression constraint that returns a set of concept-based reference set concepts. For more information, please refer to 6.7 Nested Expression Constraints.

And for information about applying filter constraints to reference set members, please refer to 6.10 Member Filters.

Any

A single 'star' (i.e. "*") may be used in the place of a concept reference to represent any concept in the substrate. The expression constraint below evaluates to the set of all concepts in the given substrate.

```
*
```

Using the long syntax, the above expression constraint may also be represented as:

```
ANY
```

This wildcard character (or 'ANY' keyword) may be used anywhere within an expression constraint that a concept reference may be used. In many situations, the wildcard is equivalent to the following expression constraint:

```
<< 138875005 |SNOMED CT concept|
```

However, some situations exist in which the concept 138875005 |SNOMED CT concept| is not included in the substrate, and therefore cannot be used to determine the full set of concepts available. In other cases, the single character wildcard may serve as a convenient shortcut for the longer expression constraint above.

Please note that the following three expression constraints evaluate to the same set of concepts:

```
*
```

```
<< *
```

```
>> *
```

The two expression constraints below evaluate to all concepts in the substrate minus the root concept:

```
< *
```

```
<! *
```

And the two expression constraints below evaluate to all non-leaf concepts in the substrate:

```
> *
```

```
>! *
```

Finally, the expression constraint below evaluates to all concepts that are referenced by any reference set in the substrate:

```
^ *
```

---

1. **Note**: *If a reference set field is selected that does not exist in any of the identified reference sets, then the expression constraint should be considered erroneous.*
2. ***Note**: Returning the values of more than one reference set field may only be done as the final operation of an expression constraint.*
3. ***Note:** Attempts to select more than one reference set field on an inner subquery should result in an execution error.*
4. ***Note:** In some implementation contexts, the memberOf function may be restricted to return only a single field.*

6.2 Refinements

In this section, we illustrate how the set of matching concepts can be filtered using one or more simple attribute refinements. For more information on applying refinements to nested expression constraints, using nested attribute names and using nested attribute values, please refer to 6.7 Nested Expression Constraints.

Attributes

Adding an attribute refinement to an expression constraint restricts the set of valid clinical meanings to only those whose defining attributes satisfy the given refinement condition. Similarly to SNOMED CT Compositional Grammar, attribute refinements are placed after a 'colon' (i.e. ":") in the expression constraint.

The example below is satisfied only by the set of lung disorders, which have an associated morphology that is exactly equal to 79654002 |Edema|.

```
< 19829001 |Disorder of lung| :
   116676008 |Associated morphology| = 79654002 |Edema|
```

Using the long syntax, the above expression is represented as:

```
descendantOf 19829001 |Disorder of lung| :
   116676008 |Associated morphology| = 79654002 |Edema|
```

In many cases, however, the value of the matching attribute is allowed to be either the concept itself, *or* a descendant of that concept. In these cases, the descendantOrSelfOf operator is used prior to the concept representing the attribute value. For example, the expression constraint below (in brief and long syntaxes respectively) is satisfied only by the set of lung disorders, which have an associated morphology of 79654002 |Edema| or any descendant of 79654002 |Edema|.

```
< 19829001 |Disorder of lung| :
   116676008 |Associated morphology| = << 79654002 |Edema|
```

```
descendantOf 19829001 |Disorder of lung| :
   116676008 |Associated morphology| = descendantOrSelfOf 79654002 |Edema|
```

When more than one attribute is defined in an expression constraint, the attributes are normally separated by a comma. A comma between two attributes indicates a conjunction and implies that both attribute conditions must be true. For example, the expression constraint below, written in brief syntax, is satisfied only by the set of clinical findings, which have both a finding site of 39057004 |Pulmonary valve structure| (or a subtype of 39057004 |Pulmonary valve structure|) and an associated morphology of 'stenosis' (or a subtype of 'stenosis').

```
< 404684003 |Clinical finding| :
   363698007 |Finding site| =<< 39057004 |Pulmonary valve structure| ,
   116676008 |Associated morphology| =<< 415582006 |Stenosis|
```

Please note that attribute refinements may also be used when the focus concept is '*' (or ANY). The following expression constraint represents any concept that has a 246075003 |Causative agent| attribute whose value is 387517004 |Paracetamol|.

```
* : 246075003 |Causative agent| = 387517004 |Paracetamol|
```

Using the long syntax, the above expression may also be represented as:

```
ANY : 246075003 |Causative agent| = 387517004 |Paracetamol|
```

Attribute Groups

Similarly to SNOMED CT compositional grammar, expression constraints use curly braces (i.e. "{..}") to indicate that a set of attributes should be grouped together in an attribute group. For example, the expression constraint below is satisfied only by the set of clinical findings with an associated morphology of 'stenosis' (or descendant) at the finding site 'pulmonary valve structure' (or descendant), and also with an associated morphology of 'hypertrophy' (or descendant) at the finding site 'right ventricular structure' (or descendant).

```
< 404684003 |Clinical finding| :
  { 363698007 |Finding site| =<< 39057004 |Pulmonary valve structure| ,
    116676008 |Associated morphology| =<< 415582006 |Stenosis| },
  { 363698007 |Finding site| =<< 53085002 |Right ventricular structure| ,
    116676008 |Associated morphology| =<< 56246009 |Hypertrophy| }
```

Using the 'long syntax', the above expression constraint is represented as:

```
descendantOf 404684003 |Clinical finding| :
  { 363698007 |Finding site| = descendantOrSelfOf 39057004 |Pulmonary valve structure| ,
    116676008 |Associated morphology| = descendantOrSelfOf 415582006 |Stenosis| },
  { 363698007 |Finding site| = descendantOrSelfOf 53085002 |Right ventricular structure| ,
    116676008 |Associated morphology| = descendantOrSelfOf 56246009 |Hypertrophy| }
```

Attribute Constraint Operators

In some cases, an attribute concept has subtypes or supertypes in the |Concept model attribute| hierarchy. Where this occurs, it is possible to indicate that an attribute condition may be satisfied by matching one of the subtypes or supertypes of the given attribute. This is done adding a constraint operator directly before the attribute name concept. For example, the expression constraint below will not only match clinical findings that are |Associated with | a type of |Edema|, but also those that are |Due to|, |After| or the |Causative agent| of a type of |Edema|. This result

occurs because the 47429007 |Associated with| attribute concept has three subtypes: 255234002 |After|, 246075003 |Causative agent| and 42752001 |Due to|.

```
<< 404684003 |Clinical finding| :
  << 47429007 |Associated with| = << 267038008 |Edema|
```

This expression constraint is represented in the long syntax as:

```
descendantOrSelfOf 404684003 |Clinical finding| :
  descendantOrSelfOf 47429007 |Associated with| = descendantOrSelfOf 267038008 |Edema|
```

Similarly, the expression constraint below will not only match clinical findings that are |Due to| a type of |Edema|, but also those that have an |Associated with| relationship whose value is a type of |Edema|.

```
<< 404684003 |Clinical finding| :
  >> 246075003 |Causative agent| = << 267038008 |Edema|
```

This expression constraint is represented in the long syntax as:

```
descendantOrSelfOf 404684003 |Clinical finding| :
  ancestorOrSelfOf 246075003 |Causative agent| = descendantOrSelfOf 267038008 |Edema|
```

Concrete Values

The revised SNOMED CT Compositional Grammar allows attributes to be given concrete values (e.g. Strings, Integers, Decimal, Boolean). The SNOMED CT Expression Constraint Language supports the ability to compare these attribute values with a given concrete value.

When numeric concrete values (i.e. Integers and Decimals) are compared, a set of standard mathematical operators may be used. These mathematical operators are:

Operator	Name
=	Equals
!=	Not equals
<	Less than
<=	Less than or equals
>	Greater than
>=	Greater than or equals

Please note that the 'not equals' operator may alternatively be represented as "<>" and "not =" (case insensitive) in the long syntax.

The following expression constraint is satisfied by oral medicinal products, which contain amoxicillin and have a presentation strength greater than or equal to 250 mg.

```
< 763158003 |Medicinal product (product)| :
   411116001 |Has manufactured dose form (attribute)| = << 385268001 |Oral dose form (dose form)| ,
  { << 127489000 |Has active ingredient (attribute)| = << 372687004 |Amoxicillin (substance)| ,
      1142135004 |Has presentation strength numerator value (attribute)| >= #250,
     732945000 |Has presentation strength numerator unit (attribute)| = 258684004 |milligram (qualifier
value)| }
```

Please note that, as per SNOMED CT Compositional Grammar, integer and decimal values are preceded by a hash character (e.g. "#500"), while string values are surrounded by double quotes (e.g. "PANADOL").

To find those oral amoxicillin products that have a strength between 250 and 800 mg (inclusive), the following expression constraint may be used:

```
< 763158003 |Medicinal product (product)| :
   411116001 |Has manufactured dose form (attribute)| = << 385268001 |Oral dose form (dose form)| ,
  { << 127489000 |Has active ingredient (attribute)| = << 372687004 |Amoxicillin (substance)| ,
      1142135004 |Has presentation strength numerator value (attribute)| >= #250,
     1142135004 |Has presentation strength numerator value (attribute)| <= #800,
     732945000 |Has presentation strength numerator unit (attribute)| = 258684004 |milligram (qualifier
value)| }
```

Concrete values of type string and boolean may also be included in an expression constraint, and compared using an 'equal to' (i.e. "=") or 'not equal to' (i.e. "!=") operator. The following expression constraint is satisfied only by products with a product name equal to "PANADOL"[1].

```
< 373873005 |Pharmaceutical / biologic product| :
   3460481009 |Has product name| = "PANADOL"
```

The following expression constraint is satisfied only by products that are in the national benefit scheme (of the given country)[2].

```
< 373873005 |Pharmaceutical / biologic product| :
   859999999102 |Is in national benefit scheme| = TRUE
```

Reverse Attributes

In most cases, an attribute refinement is satisfied by those concepts, which are the source concept of a defining relationship whose destination concept matches the attribute value. In some cases, however, it may be necessary to select the destination concept of a relationship and constrain the source concept to a given attribute value. To achieve this, an expression constraint indicates that an attribute is to be constrained in the reverse order using a 'reverse flag'[3]. In the brief syntax, the reverse flag is represented by preceding the name of the attribute with a capital letter 'R'.

For example, the expression constraint below finds the set of anatomical structures, which are the finding site of a type of bone fracture (e.g. 85050009 |Humerus|, 71341001 |Femur|).

```
< 91723000 |Anatomical structure| :
  R 363698007 |Finding site| = < 125605004 |Fracture of bone|
```

The above expression constraint is represented in the long syntax as:

```
descendantOf 91723000 |Anatomical structure| :
  reverseOf 363698007 |Finding site| = descendantOf 125605004 |Fracture of bone|
```

Dotted Attributes

An alternative way of representing 'reversed attributes' is by applying the *dot notation* to represented them as *dotted attributes*. Using this alternative notation, " < 123456 |X| . 234567 |Y| " represents the set of attribute values (i.e. destination concepts) of the attribute "Y" for descendants or self of concept "X". This is therefore equivalent to " * : R 234567 |Y| = < 123456 |X| " using the reverse flag.

The previous expression constraint (which finds the set of body sites for any subtype of bone fracture) has an equivalent representation using the 'dot notation' of:

```
< 91723000 |Anatomical structure| AND (< 125605004 |Fracture of bone| . 363698007 |Finding site| )
```

Because all values of 363698007 |Finding site| must be < 91723000 |Anatomical structure| (according to the SNOMED CT concept model), this expression constraint can be further simplified to:

```
< 125605004 |Fracture of bone| . 363698007 |Finding site|
```

The next example finds the set of substances, which are an active ingredient in any product containing amoxicillin.

```
< 105590001 |Substance| :
  R << 127489000 |Has active ingredient| = < 27658006 |Product containing amoxicillin|
```

This expression constraint is represented in the long syntax as:

```
descendantOf 105590001 |Substance| :
   ReverseOf descendantOrSelfOf 127489000 |Has active ingredient| = descendantOf 27658006 |Product
containing amoxicillin|
```

An equivalent way of representing this constraint, using the 'dot notation' is:

```
< 105590001 |Substance| AND ( < 27658006 |Product containing amoxicillin| . << 127489000 |Has active
ingredient| )
```

or (using the SNOMED CT concept model to simplify):

```
< 27658006 |Product containing amoxicillin| . << 127489000 |Has active ingredient|
```

When more than one dot attribute is used in sequence, the dot notation is evaluated sequentially from left to right. For example, the following expression constraint represents the set of |Finding sites| of any concept that is |Associated with| a subtype of |Disorder of lung|.

```
< 19829001 |Disorder of lung| . < 47429007 |Associated with| . 363698007 |Finding site|
```

This expression constraint is evaluated by first finding the descendants of |Disorder of lung|, then finding the set of attribute values for these concepts (with an attribute type that is any subtype of |Associated with|), and then from these attribute value concepts, finding the value of any |Finding sites| attribute. Please note that the expression constraint above (with no brackets) is equivalent to the one below (with brackets added).

```
((< 19829001 |Disorder of lung| ) . < 47429007 |Associated with| ) . 363698007 |Finding site|
```

Any Attribute Name and Value

A single 'star' (i.e. "*") may be used in the place of an attribute name to represent any attribute in the substrate. The expression constraint below evaluates to the set of clinical findings which have any attribute with a value of 79654002 |Edema|.

```
< 404684003 |Clinical finding| : * = 79654002 |Edema|
```

Using the long syntax, the above expression constraint may also be represented as:

```
descendantOf 404684003 |Clinical finding| : ANY = 79654002 |Edema|
```

The 'star' symbol (i.e. "*") may also be used to represent any attribute value (either with or without refinement). The following expression constraint evaluates to the set of clinical findings which have an associated morphology (with any value).

```
< 404684003 |Clinical finding| : 116676008 |Associated morphology| = *
```

Using the long syntax, the above expression constraint may also be represented as:

```
descendantOf 404684003 |Clinical finding| : 116676008 |Associated morphology| = ANY
```

[1] Concrete values of type string are case sensitive and compared using the Unicode Collation Algorithm (http://www.unicode.org/reports/tr10/).

2 Please note that the concept 859999999102 |Is in national benefit scheme| is a fictitious attribute used here to illustrate boolean values.

3 It should be noted that using a reversed attribute joined by conjunction with a non-reversed attribute may lead to a nonsensical constraint (e.g. "<<a: {b=c, Rd=e}"). This is because the target concept of the reversed attribute must be matched with the source concept of the non-reversed attribute, which in turn must be the same as the source concept of the reversed attribute (being in the same attribute group). This would require the reversed attribute to be reflexive (i.e. the source and target concept to be the same).

6.3 Cardinality

Attribute cardinality

Overview

To support use cases such as the SNOMED CT concept model and terminology binding, expression constraints may constrain the number of times an attribute can be included in an expression or concept definition represented in the SNOMED CT distribution view[1] . This is done using a cardinality constraint, which consists of a minimum cardinality and a maximum cardinality (written "[X..Y]"). A minimum cardinality of X constrains the valid clinical meanings to those which have at least (i.e. >=) X non-redundant[2] attributes that match the given attribute criteria. A maximum cardinality of Y constrains the valid clinical meanings to those which have at most (i.e. <=) Y non-redundant[2] attributes that match the given attribute criteria. For example, a cardinality of "[1..5]" indicates that all clinical meanings that satisfy the given expression constraint must have at least one and at most five attributes that match the given attribute criteria.

The expression constraint below is satisfied only by products with one, two or three active ingredients.

```
< 373873005 |Pharmaceutical / biologic product| :
  [1..3] 127489000 |Has active ingredient| = < 105590001 |Substance|
```

Using the long syntax, this expression constraint may be represented as:

```
descendantOf 373873005 |Pharmaceutical / biologic product| :
  [1 to 3] 127489000 |Has active ingredient| = descendantOf 105590001 |Substance|
```

The following expression constraint is satisfied only by products which have exactly one active ingredient:

```
< 373873005 |Pharmaceutical / biologic product| :
  [1..1] 127489000 |Has active ingredient| = < 105590001 |Substance|
```

Unconstrained Cardinalities

A minimum cardinality of '0' indicates that there is *no* constraint on the minimum number of attributes that may match the given attribute criteria. For example, the following expression constraint is satisfied only by products with at most one active ingredient (i.e. the maximum cardinality is '1' and the minimum cardinality is unconstrained).

```
< 373873005 |Pharmaceutical / biologic product| :
  [0..1] 127489000 |Has active ingredient| =< 105590001 |Substance|
```

Using the long syntax, this may be represented as:

```
descendantOf 373873005 |Pharmaceutical / biologic product| :
  [0 to 1] 127489000 |Has active ingredient| = descendantOf 105590001 |Substance|
```

A maximum cardinality of '*' (or 'many') indicates that there is *no* constraint on the maximum number of attributes that may match the given attribute criteria. For example, the following expression constraint is satisfied only by products that have at least one active ingredient (i.e. the minimum cardinality is '1' and the maximum cardinality is unconstrained).

```
< 373873005 |Pharmaceutical / biologic product| :
  [1..*] 127489000 |Has active ingredient| =< 105590001 |Substance|
```

Using the long syntax, this may be represented as:

```
descendantOf 373873005 |Pharmaceutical / biologic product| :
  [1 to many] 127489000 |Has active ingredient| = descendantOf 105590001 |Substance|
```

A cardinality of [0..*] should therefore never be used as this indicates that the given attribute is not being constrained in any way, and is therefore a redundant part of the expression constraint.

Default Cardinalities

The default cardinality of each attribute, where not explicitly stated, is [1..*]. Therefore, the following two expression constraints are equivalent.

```
< 373873005 |Pharmaceutical / biologic product| :
  [1..*] 127489000 |Has active ingredient| =< 105590001 |Substance|
```

```
< 373873005 |Pharmaceutical / biologic product| :
  127489000 |Has active ingredient| =< 105590001 |Substance|
```

Non-redundant Attributes

As mentioned above, only non-redundant defining attributes are included in the cardinality count. Therefore, the following postcoordinated expression:

```
404684003 |Clinical finding| :
  { 116676008 |Associated morphology| = 72704001 |Fracture| ,
```

```
    363698007 |Finding site| = 299701004 |Bone of forearm| ,
    363698007 |Finding site| = 62413002 |Bone structure of radius| }
```

will successfully satisfy the expression constraint:

```
< 404684003 |Clinical finding| :
  [1..1] 363698007 |Finding site| =< 91723000 |Anatomical structure|
```

This is because 299701004 |Bone of forearm| is a supertype of 62413002 |Bone structure of radius| and therefore the attribute " 363698007 |Finding site| = 299701004 |Bone of forearm| " is redundant.

Attribute Cardinality in Groups

When the attributes to which cardinality are applied can be grouped, but braces are not used in the expression constraint, the cardinality constrains the number of times the attribute may be included in *any* attribute group. For example, the following expression constraint is satisfied by any clinical finding whose definition has two or more non-redundant finding sites, irrespective of which attribute group they are contained in.

```
< 404684003 |Clinical finding| :
  [2..*] 363698007 |Finding site| =< 91723000 |Anatomical structure|
```

In contrast, when braces are placed around an attribute with a given cardinality, there must exist at least one attribute group for which the given cardinality is satisfied by attributes in that group. For example, the following expression constraint is satisfied by any clinical finding whose definition contains an attribute group with two or more non-redundant finding sites.

```
< 404684003 |Clinical finding| :
  { [2..*] 363698007 |Finding site| =< 91723000 |Anatomical structure| }
```

Attribute Group Cardinality

Minimum and maximum cardinalities may also be applied to attribute groups. A minimum attribute group cardinality of X constrains the valid clinical meanings to those which have at least (i.e. >=) X non-redundant attribute groups that match the given attribute group criteria. A maximum cardinality of Y constrains the valid clinical meanings to those which have at most (i.e. <=) Y non-redundant attribute groups that match the given attribute group criteria. For example, a cardinality of "[1..2]" indicates that all clinical meanings that satisfy the given expression constraint must have at least one and at most two attribute groups that match the given attribute group criteria.

The expression constraint below is satisfied only by products with one, two or three attribute groups, which each contain at least one active ingredient relationship.

```
< 373873005 |Pharmaceutical / biologic product| :
  [1..3] { [1..*] 127489000 |Has active ingredient| =< 105590001 |Substance| }
```

Please note that the above expression constraint is equivalent to:

```
< 373873005 |Pharmaceutical / biologic product| :
  [1..3] { 127489000 |Has active ingredient| =< 105590001 |Substance| }
```

And may be written using the long syntax as:

```
descendantOf 373873005 |Pharmaceutical / biologic product| :
  [1 to 3] { [1 to many] 127489000 |Has active ingredient| =
    descendantOf 105590001 |Substance| }
```

Unconstrained Cardinalities

As with attribute cardinalities, a minimum cardinality of '0' indicates that there is *no* constraint on the minimum number of attribute groups that may match the given attribute group criteria. For example, the following expression constraint is satisfied only by products with at most one attribute group containing an active ingredient relationship (i.e. the maximum attribute group cardinality is '1' and the minimum attribute group cardinality is unconstrained).

```
< 373873005 |Pharmaceutical / biologic product| :
  [0..1] { 127489000 |Has active ingredient| =< 105590001 |Substance| }
```

Using the long syntax, this may be represented as:

```
descendantOf 373873005 |Pharmaceutical / biologic product| :
  [0 to 1] { 127489000 |Has active ingredient| = descendantOf 105590001 |Substance| }
```

A maximum cardinality of '*' (or 'many') indicates that there is *no* constraint on the maximum number of attribute groups that may match the given attribute group criteria. For example, the following expression constraint is satisfied only by products that have at least one attribute group containing an active ingredient relationship (i.e. the minimum attribute group cardinality is '1' and the maximum attribute group cardinality is unconstrained).

```
< 373873005 |Pharmaceutical / biologic product| :
  [1..*] { 127489000 |Has active ingredient| =< 105590001 |Substance| }
```

Using the long syntax, this may be represented as:

```
descendantOf 373873005 |Pharmaceutical / biologic product| :
  [1 to *] { 127489000 |Has active ingredient| = descendantOf 105590001 |Substance| }
```

A cardinality of [0..*] should therefore never be used as this indicates that the given attribute group is not being constrained in any way, and is therefore a redundant part of the expression constraint.

Default Cardinalities

As with attribute cardinality, the default attribute group cardinality, where not explicitly stated, is [1..*]. Therefore, the following four expression constraints are equivalent.

```
< 373873005 |Pharmaceutical / biologic product| :
  { 127489000 |Has active ingredient| =< 105590001 |Substance| }
```

```
< 373873005 |Pharmaceutical / biologic product| :
  { [1..*] 127489000 |Has active ingredient| =< 105590001 |Substance| }
```

```
< 373873005 |Pharmaceutical / biologic product| :
  [1..*] { 127489000 |Has active ingredient| =< 105590001 |Substance| }
```

```
< 373873005 |Pharmaceutical / biologic product| :
  [1..*] { [1..*] 127489000 |Has active ingredient| =< 105590001 |Substance| }
```

Non-redundant Attribute Groups

As mentioned above, only non-redundant defining attributes are included in the cardinality count. Therefore, the following postcoordinated expression:

```
< 404684003 |Clinical finding| :
  { 363698007 |Finding site| = 299701004 |Bone of forearm| },
  { 363698007 |Finding site| = 62413002 |Bone structure of radius| }
```

will successfully satisfy the expression constraint:

```
< 404684003 |Clinical finding| :
  [1..1] { 363698007 |Finding site| =< 91723000 |Anatomical structure| }
```

This is because 299701004 |Bone of forearm| is a supertype of 62413002 |Bone structure of radius| and therefore the attribute group " { 363698007 |Finding site| = 299701004 |Bone of forearm| } " is redundant.

Attribute and Attribute Group Cardinalities

Attribute cardinalities and attribute group cardinalities can be used together to achieve a combined effect. For example, to represent the set of clinical findings which have *no* attribute groups that contain two or more finding site attributes (in the same attribute group), the following expression constraint can be used:

```
< 404684003 |Clinical finding| :
  [0..0] { [2..*]  363698007 |Finding site|  =< 91723000 |Anatomical structure| }
```

Reverse Cardinalities

When a cardinality constraint is applied to a reversed refinement, it constrains the number of source concepts (matching the given criteria) for which each destination concept may be relevant attribute value.

For example, the following expression constraint represents the substances, which are the active ingredient of exactly three products.

```
< 105590001 |Substance| : [3..3] R  127489000 |Has active ingredient|  = *
```

If this expression constraint was executed against a simplified substrate containing the following seven relationships:

Source concept	Attribute	Destination concept
412458007 |Orphenadrine + aspirin + caffeine|	127489000 |Has active ingredient|	372714007 |Orphenadrine|
412458007 |Orphenadrine + aspirin + caffeine|	127489000 |Has active ingredient|	387458008 |Aspirin|
412458007 |Orphenadrine + aspirin + caffeine|	127489000 |Has active ingredient|	255641001 |Caffeine|
412096001 |Aspirin + codeine|	127489000 |Has active ingredient|	387458008 |Aspirin|
412096001 |Aspirin + codeine|	127489000 |Has active ingredient|	387494007 |Codeine|
424102008 |Acetaminophen+aspirin|	127489000 |Has active ingredient|	387517004 |Acetaminophen|
424102008 |Acetaminophen+aspirin|	127489000 |Has active ingredient|	387458008 |Aspirin|

then the result would include only the concept 387458008 |Aspirin|.

---

1 For more information about the SNOMED CT distribution view, please refer to the SNOMED CT Technical Implementation Guide. Please note that full normalization of expressions (as would be performed by a Description Logic classifier) is required prior to evaluation.

2 As defined in the SNOMED CT Technical Implementation Guide. [a b]

6.4 Conjunction and Disjunction

Compound Expression Constraints

Expression constraints can be built up from smaller parts using conjunction (i.e. AND) and disjunction (i.e. OR). The simplest example of this is where the conjunction or disjunction is used between two simple expressions. For example, the following expression constraint is satisfied only by clinical findings which are *both* a disorder of the lung *and* an edema of the trunk. This gives the same result as a mathematical *intersection* between the set of 19829001 |Disorder of lung| descendants and the set of 301867009 |Edema of trunk| descendants.

```
< 19829001 |Disorder of lung| AND < 301867009 |Edema of trunk|
```

Please note that all keywords are case insensitive, so the following two expression constraints are equivalent to the above:

```
< 19829001 |Disorder of lung| and < 301867009 |Edema of trunk|
```

```
< 19829001 |Disorder of lung| And < 301867009 |Edema of trunk|
```

The next expression constraint is satisfied only by clinical findings which are *either* a disorder of the lung *or* an edema of the trunk. This gives the same result as a mathematical *union* of the set of 19829001 |Disorder of lung| descendants and the set of 301867009 |Edema of trunk| descendants. For this reason, an *OR* operator will usually allow more valid clinical meanings than an *AND* operator.

```
< 19829001 |Disorder of lung| OR < 301867009 |Edema of trunk|
```

Conjunction and disjunction operators may also be combined with the use of the 'member of' function, as shown below:

```
< 19829001 |Disorder of lung| AND ^ 700043003 |Example problem list concepts reference set|
```

This expression constraint is satisfied only by concepts that belong to the 19829001 |Disorder of lung| hierarchy *and* are also members of the 700043003 |Example problem list concepts reference set|.

When more than one conjunction or more than one disjunction is used, round brackets can be optionally applied. For example, the following expression constraints are all valid and equivalent to each other:

```
< 19829001 |Disorder of lung| AND < 301867009 |Edema of trunk| AND
  ^ 700043003 |Example problem list concepts reference set|
```

```
(< 19829001 |Disorder of lung| AND < 301867009 |Edema of trunk| ) AND
    ^ 700043003 |Example problem list concepts reference set|
```

```
< 19829001 |Disorder of lung| AND (< 301867009 |Edema of trunk| AND
  ^ 700043003 |Example problem list concepts reference set| )
```

However, where a conjunction and disjunction are both used together, it is mandatory to use round brackets to disambiguate the meaning of the expression constraint. For example, the following expression constraint is ***not*** valid:

```
< 19829001 |Disorder of lung| AND < 301867009 |Edema of trunk| OR
  ^ 700043003 |Example problem list concepts reference set|
```

And must be expressed (depending on the intended meaning) as either:

```
(< 19829001 |Disorder of lung| AND < 301867009 |Edema of trunk| ) OR
  ^ 700043003 |Example problem list concepts reference set|
```

Or as:

```
< 19829001 |Disorder of lung| AND (< 301867009 |Edema of trunk| OR
  ^ 700043003 |Example problem list concepts reference set| )
```

Attribute Conjunction and Disjunction

Conjunction and disjunction may be used within refinements in a variety of ways. The most common way of using these operators in a refinement is to define the conjunction or disjunction of individual attributes.

For example, the expression constraint below, in which the comma between the two attributes represents conjunction, is satisfied only by clinical findings which have *both* a finding site of pulmonary valve structure (or subtype) *and* an associated morphology of stenosis (or subtype).

```
< 404684003 |Clinical finding| :
   363698007 |Finding site| = << 39057004 |Pulmonary valve structure| ,
   116676008 |Associated morphology| = << 415582006 |Stenosis|
```

This expression constraint can equivalently be expressed as:

```
< 404684003 |Clinical finding| :
   363698007 |Finding site| = << 39057004 |Pulmonary valve structure| AND
   116676008 |Associated morphology| = << 415582006 |Stenosis|
```

The following example uses the disjunction operator (OR) to represent the disjunction of two attributes. This constraint is satisfied only by clinical findings which have *either* an associated morphology of 'infarct' (or subtype) *or* are due to a myocardial infarction (or subtype).

```
< 404684003 |Clinical finding| :
    116676008 |Associated morphology| =<< 55641003 |Infarct| OR
    42752001 |Due to| =<< 22298006 |Myocardial infarction|
```

When more than one conjunction or more than one disjunction is used in a refinement, round brackets can be optionally applied. For example, the following expression constraints are all valid and equivalent to each other:

```
< 404684003 |Clinical finding| :
    363698007 |Finding site| =<< 39057004 |Pulmonary valve structure| AND
    116676008 |Associated morphology| =<< 415582006 |Stenosis| AND
    42752001 |Due to| =<< 445238008 |Malignant carcinoid tumor|
```

```
< 404684003 |Clinical finding| :
    ( 363698007 |Finding site| =<< 39057004 |Pulmonary valve structure| AND
      116676008 |Associated morphology| =<< 415582006 |Stenosis| ) AND
      42752001 |Due to| =<< 445238008 |Malignant carcinoid tumor|
```

```
< 404684003 |Clinical finding| :
    363698007 |Finding site| =<< 39057004 |Pulmonary valve structure| AND
    ( 116676008 |Associated morphology| =<< 415582006 |Stenosis| AND
      42752001 |Due to| =<< 445238008 |Malignant carcinoid tumor| )
```

However, where a conjunction and disjunction are both used together in a refinement, it is mandatory to use brackets to disambiguate the meaning of the expression constraint.

For example, the following expression constraint is ***not*** valid:

```
< 404684003 |Clinical finding| :
    363698007 |Finding site| =<< 39057004 |Pulmonary valve structure| AND
    116676008 |Associated morphology| =<< 415582006 |Stenosis| OR
    42752001 |Due to| =<< 445238008 |Malignant carcinoid tumor|
```

And must be expressed (depending on the intended meaning) as either:

```
< 404684003 |Clinical finding| :
    ( 363698007 |Finding site| = << 39057004 |Pulmonary valve structure| AND
      116676008 |Associated morphology| = << 415582006 |Stenosis| ) OR
      42752001 |Due to| = << 445238008 |Malignant carcinoid tumor|
```

Or as:

```
< 404684003 |Clinical finding| :
     363698007 |Finding site| = << 39057004 |Pulmonary valve structure| AND
    ( 116676008 |Associated morphology| = << 415582006 |Stenosis| OR
      42752001 |Due to| = << 445238008 |Malignant carcinoid tumor| )
```

Attribute Group Conjunction and Disjunction

Similarly, conjunction and disjunction may be defined between attribute groups. The following expression constraint is satisfied only by clinical findings which *either* have a finding site of pulmonary valve structure (or subtype) and an associated morphology of stenosis (or subtype), *OR* have a finding site of right ventricular structure (or subtype) and an associated morphology of hypertrophy (or subtype).

```
< 404684003 |Clinical finding| :
    { 363698007 |Finding site| = << 39057004 |Pulmonary valve structure| ,
      116676008 |Associated morphology| = << 415582006 |Stenosis| } OR
    { 363698007 |Finding site| = << 53085002 |Right ventricular structure| ,
      116676008 |Associated morphology| = << 56246009 |Hypertrophy| }
```

Attribute Value Conjunction and Disjunction

Conjunction and disjunction can also be applied to attribute values. The example below is satisfied only by members of the adverse drug reactions reference set for GP/FP health issue, which have a causative agent that is *either* a subtype of pharmaceutical / biologic product *or* a subtype of substance.

```
^ 450990004 |Adverse drug reactions reference set for GP/FP health issue| :
    246075003 |Causative agent| = (< 373873005 |Pharmaceutical / biologic product| OR < 105590001 |
Substance| )
```

Similarly, attribute values can also use conjunction. The following expression constraint is satisfied only by clinical findings with an associated morphology whose value is *both* a subtype (or self) of ulcer *and* a subtype (or self) of hemorrhage.

```
< 404684003 |Clinical finding| : 116676008 |Associated morphology| =
    (<< 56208002 |Ulcer| AND << 50960005 |Hemorrhage| )
```

For more information about nested attribute values and nested compound expression constraints, please refer to 6.7 Nested Expression Constraints.

6.5 Exclusion and Not Equals

Exclusion of Simple Expressions

Exclusion is supported in the SNOMED CT Expression Constraint Language by the binary operator 'MINUS'. Exclusion works in a similar manner to mathematical subtraction. For example, the following expression constraint returns the set of lung disorders which are not a descendant or self of edema of the trunk.

```
<< 19829001 |Disorder of lung| MINUS << 301867009 |Edema of trunk|
```

Logically, this expression constraint takes the set of descendants of 'disorder of lung' and subtracts the set of descendants of 'edema of trunk'. Please note that the keyword 'MINUS' is case insensitive.

Exclusion can also be applied to the membership of a reference set. For example, the following expression constraint returns the set of lung disorders which are not members of the cardiology reference set. That is, the set of descendants or self of 'disorder of lung' minus the set of members of the 'cardiology reference set'.

```
<< 19829001 |Disorder of lung| MINUS ^ 700043003 |Example problem list concepts reference set|
```

Please note that when more than one exclusion operator is used, or when an exclusion operator is used together with a conjunction or disjunction, round brackets must be used to disambiguate the intended meaning.

Exclusion of Attribute Values

Attribute values, represented by compound expression constraints, may also contain exclusions. When this occurs, the expression constraint is satisfied by any concept or expression which has at least one attribute (of the given type) whose value is satisfied by the compound constraint defined in the attribute value. For example, the expression constraint below represents the set of clinical findings, which have an associated morphology that is a descendant or self of ulcer and a descendant or self of hemorrhage, but not a descendant or self of obstruction.

```
< 404684003 |Clinical finding| : 116676008 |Associated morphology| =
    ((<< 56208002 |Ulcer| AND << 50960005 |Hemorrhage| ) MINUS << 26036001 |Obstruction| )
```

Not Equal to Attribute Value

It is also possible to simply state that an attribute value should not fall in a particular range. The example below is satisfied only by clinical findings which have an associated morphology that is not a descendant (or self) of obstruction.

```
< 404684003 |Clinical finding| :
    116676008 |Associated morphology| != << 26036001 |Obstruction|
```

Using the long syntax, this expression constraint can be represented as:

```
descendantOf 404684003 |Clinical finding| :
    116676008 |Associated morphology| NOT = descendantOrSelfOf 26036001 |Obstruction|
```

To prohibit an attribute from having a value in a particular range, a cardinality of [0..0] must be used. For example, the following expression constraint represents the set of clinical findings which have exactly zero (i.e. they do not have any) associated morphologies that are a descendant or self of obstruction.

```
< 404684003 |Clinical finding| :
    [0..0] 116676008 |Associated morphology| = << 26036001 |Obstruction|
```

To prohibit an attribute from having a value *outside* a particular range, a cardinality of [0..0] is used in conjunction with the 'not equal to' comparison operator. For example, the following expression constraint represents the set of clinical findings which have exactly zero associated morphologies that are *not* a descendant or self of obstruction. In other words, clinical findings for which *all* associated morphologies (if any exist) are descendants (or self) of obstruction.

```
< 404684003 |Clinical finding| :
    [0..0] 116676008 |Associated morphology| != << 26036001 |Obstruction|
```

If we also want to ensure that at least one associated morphology does exist (and all of these have a value which is a descendant or self of obstruction), then the following expression constraint can be used:

```
< 404684003 |Clinical finding| :
    [0..0] 116676008 |Associated morphology| != << 26036001 |Obstruction| and
    [1..*] 116676008 |Associated morphology| = << 26036001 |Obstruction|
```

Note that the cardinality on the second attribute may be omitted, as [1..*] is assumed by default.

6.6 Constraint Comments

Comments

SNOMED CT Expression Constraints may also include comments inline within the constraint string to explain, describe or document different aspects of the expression constraints. Each comment begins with a forward slash directly followed by a star (i.e. "/*") and ends with a star directly followed by a forward slash (i.e. "*/"). Comments may be placed anywhere in an expression constraint where whitespace (i.e. "ws") or mandatory whitespace (i.e. "mws") is allowed.

Comments have no effect on the machine processable interpretation of an expression constraint, as they should be ignored during evaluation. For example, the following two expression constraints (the first with comments, and the second without), will evaluate to exactly the same set of concepts:

```
/* Disorders of lung with edema */
< 19829001 |Disorder of lung| : /* Descendants of disorder of lung */
    116676008 |Associated morphology| = << 79654002 |Edema|
   /* Where the associated morphology is edema or a subtype */
```

```
< 19829001 |Disorder of lung| :
    116676008 |Associated morphology| = << 79654002 |Edema|
```

A comment may include both stars and forward slashes. However a star may never be directly followed by a forward slash within the middle of a comment, as this combination denotes the end of the comment.

6.7 Nested Expression Constraints

Expression constraints can be nested in a variety of ways to form nested expression constraints. These nested expression constraints use subexpressions, enclosed in round brackets, in the place of a simple concept reference.

Nested expression constraints can be created by:

- Applying constraint operators to an expression constraint
- Applying the memberOf function to an expression constraint
- Combining expression constraints using binary operators
- Adding dotted attributes to expression constraints
- Adding refinements to expression constraints
- Using expression constraints to represent valid attribute names
- Using expression constraints to represent valid attribute values

In this section, we describe each of these approaches to creating nested expression constraints.

Constraint Operators

When a constraint operator is applied to an expression constraint, the resulting set of matching expressions is the union of applying the constraint operator to each of its members.

For example, the following expression constraint represents all the members of the |Example problem list concepts reference set| plus the union of the descendants of each of these members.

```
<< (^ 700043003 |Example problem list concepts reference set| )
```

Please note that the brackets in the above expression constraint are optional. In this particular case, removing the brackets does not change the meaning of the constraint.

As another example, the following expression constraint represents the set of all descendants of the |Finding site| of |Fracture of bone|.

```
< ( 125605004 |Fracture of bone| . 363698007 |Finding site| )
```

Because the |Finding site| of |Fracture of bone| is 272673000 |Bone structure|, the above expression constraint is equivalent to:

```
< 272673000 |Bone structure|
```

Please note that this is *not* the same as the expression constraint:

```
< 125605004 |Fracture of bone| . 363698007 |Finding site|
```

which refers to the set of |Finding site| values for any descendant of |Fracture of bone|, and is instead equivalent to:

```
(< 125605004 |Fracture of bone| ). 363698007 |Finding site|
```

See the subsection below on Dotted Attributes for more information about expression constraints of this form.

MemberOf Function

The memberOf function may also be applied to an expression constraint that returns a set of concept-based reference set concepts. When this is done, the nested expression constraint (to which the memberOf function is applied) must always be enclosed in round brackets.

For example, the expression constraint below is satisfied by the set of concepts which are members of any subtype of |GP/FP health issue reference set|. In other words, it represents the union of applying the memberOf function to each of the descendants of |GP/FP health issue reference set|.

```
^ (< 450973005 |GP/FP health issue reference set| )
```

The expression constraint above evaluates to the same set of concepts as applying the memberOf function to each individual subtype of 450973005 |GP/FP health issue reference set|and then taking the union of these sets. Therefore, when applied to the 20170131 international edition of SNOMED CT, the above expression constraint evaluates to the same set of concepts as the following expression constraint.

```
^ 450990004 |Adverse drug reactions reference set for GP/FP health issue|
  OR ^ 450989008 |Allergies reference set for GP/FP health issue|
  OR ^ 450985002 |Disorders and diseases reference set for GP/FP health issue|
  OR ^ 450988000 |Family history reference set for GP/FP health issue|
  OR ^ 450991000 |Processes and procedures reference set for GP/FP health issue|
  OR ^ 450986001 |Results reference set for GP/FP health issue|
  OR ^ 450992007 |Social history reference set for GP/FP health issue|
  OR ^ 450984003 |Symptoms and signs reference set for GP/FP health issue|
```

Compound Expression Constraints

When conjunction (i.e. AND), disjunction (i.e. OR) or exclusion (i.e. MINUS) are applied to one or more complex subexpression constraints, brackets are usually required to nest the subexpression constraints.

For example, the following expression constraint uses brackets around the first complex operand (< 404684003 |Clinical finding| : 363698007 |Finding site| = << 39057004 |Pulmonary valve structure|) to apply the 'AND' operator to two expression constraints.

```
(< 404684003 |Clinical finding| :
   363698007 |Finding site| = << 39057004 |Pulmonary valve structure| )
    AND ^ 700043003 |Example problem list concepts reference set|
```

An equivalent expression constraint can be achieved by swapping the order of the operands, as shown below.

```
^ 700043003 |Example problem list concepts reference set|
  AND (< 404684003 |Clinical finding| :
     363698007 |Finding site| = << 39057004 |Pulmonary valve structure| )
```

Similarly, if both sides of the compound expression are complex expression constraints, then brackets may be required on both sides. For example:

```
(< 404684003 |Clinical finding| : 363698007 |Finding site| = << 39057004 |Pulmonary valve structure| )
  AND (< 64572001 |Disease| : 116676008 |Associated morphology| = << 415582006 |Stenosis| )
```

Dotted Attributes

Dotted attributes can also be applied to a nested subexpression constraint. When this is done, the resulting subexpression represents the union of the values of the given dotted attribute for any expression that matches the given nested subexpression constraint.

For example, the following expression constraint represents the set of all substances that are the |Direct substance| of a |Specimen collection| procedure that is |Using device| equal to a subtype (or self) of |Catheter|.

```
  (<< 17636008 |Specimen collection| : 424226004 |Using device| = << 19923001 |Catheter| ) . 363701004 |
Direct substance|
```

When executed against the 20170131 international edition of SNOMED CT, the above expression constraint matches the following three concepts:

```
  78014005 |Urine|
  87612001 |Blood|
  4635002 |Arterial blood|
```

Refinement

As mentioned in 6.2 Refinements, it is possible to apply refinements to nested expression constraints. When a refinement is applied to a complex subexpression constraint, the subexpression constraint must be enclosed in brackets.

For example, the expression constraint below represents the set of all clinical findings and events which occur after some procedure.

```
(<< 404684003 |Clinical finding (finding)| OR << 272379006 |Event (event)| ):
   255234002 |After| = << 71388002 |Procedure (procedure)|
```

Attribute Names

In some cases, the valid set of attribute names can be represented using an expression constraint. For example, the expression constraint below represents the set of bone fractures that have no additional defining attributes (besides |Finding site| and |Associated morphology|).

```
<< 125605004 |Fracture of bone| :
  [0..0] ((<< 410662002 |Concept model attribute| MINUS 363698007 |Finding site| )
    MINUS 116676008 |Associated morphology| ) = *
```

Within this expression constraint, the subexpression:

```
(<< 410662002 |Concept model attribute| MINUS 363698007 |Finding site| ) MINUS 116676008 |Associated
morphology|
```

represents the set of attributes that must match the given refinement condition (in this case, these attributes must not appear in the concept definition of matching concepts due to the cardinality of [0..0]).

Attribute Values

Similarly to the SNOMED CT Compositional Grammar, it is also possible to nest expression constraints within an attribute value. Please note that when the attribute value is a simple expression constraint (as per the above examples), brackets are not required around the value. However, when the attribute value is either an expression constraint with a refinement, or a compound expression constraint with a binary operator, then brackets must be placed around the attribute value. For example, the following expression constraint represents the set of clinical findings which are associated with another clinical finding that has an associated morphology of 'infarct' (or subtype).

```
< 404684003 |Clinical finding| :
   47429007 |Associated with| = (< 404684003 |Clinical finding| :
     116676008 |Associated morphology| = << 55641003 |Infarct| )
```

In this example, brackets are required around the nested attribute value " < 404684003 |Clinical finding| : 116676008 |Associated morphology| = << 55641003 |Infarct| ".

6.8 Description Filters

In this section, we illustrate how description filters can be applied to expression constraints to further restrict the matching concepts.

Overview

Description filter constraints provide the ability to limit the set of concepts, that satisfy a given expression constraint, based on the descriptions associated with each concept. Only concepts that have at least one matching description for each filter criteria will be included in the set of matching concepts. Descriptions can be filtered based on their term, type, language, dialect, acceptability in a given dialect, module, effectiveTime and active status. Description filters are specified inside double curly braces, and optionally being with the letter "D". Any filter that does not specify its type is, by default, assumed to be a description filter.

In the following sections, we explain each type of description filter criteria.

Term Filter

Term filters enable an expression constraint to match on only those concepts with an associated description whose term matches the given search term. For example, the following expression constraint is satisfied by SNOMED CT concepts with a description matching the search terms "heart" and "att". This expression constraint works like a term search performed in a SNOMED CT browser. Please note that the "D" (either upper or lower case) at the start of the filter indicates that this is a description filter constraint, rather than a concept filter constraint (see 6.9 Concept Filter Constraints). If the type of a filter constraint is not specified (as in most of the examples below), then it is assumed that the constraint is a description constraint.

```
* {{ D term = "heart att" }}
```

By default, term filters match using a word-prefix-any-order match technique. This means that each string value in the search term must match the start of a word in the concept's description term, but that these words may appear in any order. This word-prefix-any-order match technique can be explicitly specified in the term filter, using the keyword "match:" before the search term. For example, the following four expression constraints are equivalent, and are each satisfied only by diseases with a description term that includes both a word starting with "heart" **and** a word starting with "att" (in any order).

```
< 64572001 |Disease| {{ term = "heart att"}}
```

```
< 64572001 |Disease| {{ term = "heart", term = "att"}}
```

```
< 64572001 |Disease| {{ term = match:"heart att"}}
```

```
< 64572001 |Disease| {{ term = "att heart"}}
```

To indicate that a matching description may match either one search term or another, a search term set may be used.

The example below matches only those diseases with a description term containing **either** a word starting with "heart" **or** a word starting with "card" (or both).

```
< 64572001 |Disease| {{ term = ("heart" "card")}}
```

The other match technique that may be used is a wildcard match. This uses an asterisk ('*') to indicate zero to many characters in the given position, and is specified using the keyword "wild:" before the search term.

For example, the expression constraint below will match only diseases with a description term starting with "cardi" and ending with "opathy" with any number of characters between. This term filter would therefore match on terms such as "cardiopathy", "cardiomyopathy" and "cardiac channelopathy", but would **not** match on terms like "atrial cardiopathy" or "Cardiomyopathy (disorder)".

```
< 64572001 |Disease| {{ term = wild:"cardi*opathy"}}
```

It is also possible to mix the match techniques in a search term set. For example, the expression constraint below will match those diseases with a description term that either contains a word starting with "gas", or ending with "itis" - e.g. "gastric flu", "gastritis", or "tonsillitis".

```
< 64572001 |Disease| {{ term = (match:"gas" wild:"*itis")}}
```

If more than one filter is applied, then **all** filters (surrounded in double braces) must match at least one description of a concept, for that concept to satisfy the constraint. The descriptions that match each of the filters can either be the same description, or different descriptions on the same concept.

The expression constraint below matches those diseases which have **both** a description that contains a word starting "eye" **and** a description that ends with "itis". For example, this constraint would match the concept 9826008 |Conjunctivitis (disorder)| (with synonyms "Pink eye disease" and "Conjunctivitis") and the concept 15680481000119104 |Viral conjunctivitis of bilateral eyes (disorder)| (with synonyms "Bilateral viral conjunctivitis" and "Viral conjunctivitis of both eyes"), but would **not** match the concept 45261009 |Viral conjunctivitis (disorder)| (which does not have a synonym matching the word prefix "eye").

```
< 64572001 |Disease| {{ term = "eye"}} {{ term = wild:"*itis"}}
```

Language Filter

Language filters enable an expression constraint to match on only those concepts with a matching description in a specified language. Language filters use the keyword "language", followed by a comparison operator (e.g. "=" or "!="), and the ISO 639-1 two-character language code (in upper or lowercase).

The expression constraint below matches only those diseases with a Swedish description containing the word prefix "hjärt" - e.g. 41884003 |hjärtpolyp| from the Swedish Edition (20200531)

```
< 64572001 |Disease| {{ term = "hjärt", language = sv }}
```

The expression constraint below matches only those diseases with a Swedish description containing the word prefix "hjärt" and an English description containing the word prefix "heart" - e.g. 84114007 |hjärtsvikt| (with English synonym "Heart failure") from the Swedish Edition (20200531).

```
< 64572001 |Disease| {{ term = "hjärt", language = sv }} {{ term = "heart", language = en }}
```

Description Type Filter

Type filters enable an expression constraint to match on only those concepts with a matching description of a specified type. Type filters may either use the keyword "type" with the values "fsn", "syn" or "def", or may use the keyword "typeId" with a concept value that is < 900000000000446008 |Description type| .

The following table lists the valid description type keywords in both the brief and full syntax, and their equivalent concept reference alternatives. Please note that the full syntax accepts both the brief and full syntax keywords. If additional description types are required, these must be specified in a filter using the 'typeId' keyword with the corresponding concept reference.

Type Keyword		TypeId
Brief Syntax	**Full Syntax**	Concept Reference
fsn	fullySpecifiedName	900000000000003001 |Fully specified name|
syn	synonym	900000000000013009 |Synonym|
def	definition	900000000000550004 |Definition|

For example, the expression constraint below matches all the subtypes of |Heart disease|, that have a fully specified name containing the word prefix "heart".

```
< 56265001 |Heart disease| {{ term = "heart", type = fsn }}
```

The following two expression constraints are equivalent, and both match only the subtypes of |Heart disease|, which have a Swedish synonym containing the word prefix "hjärt".

```
< 56265001 |Heart disease| {{ term = "hjärt", language = SV, type = syn }}
```

```
< 56265001 |Heart disease| {{ term = "hjärta", language = sv, typeId = 900000000000013009 |synonym| }}
```

The two equivalent expression constraints below match the subtypes of |Heart disease|, which either have a synonym containing the word prefix "heart", or a fully specified name containing the word prefix "heart".

```
< 56265001 |Heart disease| {{ term = "heart", type = (syn fsn) }}
```

```
< 56265001 |Heart disease| {{ term = "heart", typeId = ( 900000000000013009 |Synonym|
900000000000003001 |Fully specified name| ) }}
```

Dialect Filter

Dialect filters enable an expression constraint to match on only those concepts with a matching description in a specified language reference set. Dialect filters may either use the keyword "dialect" with a value that represents a valid alias for a specific language reference set, or may use the keyword "dialectId" with a concept value that is < 900000000000506000 |Language type reference set| . Please refer to Appendix C - Dialect Aliases for a selection of valid dialect aliases for known language reference sets.

For example, the two equivalent expression constraints below will match all subtypes of |Disease| that have a description in the Australian English language reference set.

```
< 64572001 |Disease| {{ dialect = en-au }}
```

```
< 64572001 |Disease| {{ dialectId = 32570271000036106 |Australian English language reference set| }}
```

The expression constraint below matches all diseases with a description in the New Zealand English language reference set that has a word starting with "cardio".

```
< 64572001 |Disease| {{ term = "cardio", dialect = en-nz }}
```

In some situations, multiple language reference sets need to be used together to identify an appropriate set of concepts. A filter constraint may include a list of dialects to specify that a matching description may belong to any of the given language reference sets.

For example, the following expression constraint matches all diseases that have a description in either the en-nhs-clinical or en-nhs-pharmacy language reference sets, where that description contains a word starting with the prefix "card".

```
< 64572001 |Disease| {{ term = "card", dialect = ( en-nhs-clinical en-nhs-pharmacy ) }}
```

Acceptability Filter

Acceptability filters enable an expression constraint to match on only those concepts with a matching description that has the specified acceptability in the specified language reference set. Acceptability filters must always be applied to a specified dialect. As such, they are represented by placing the required acceptability in brackets after the value of the dialect filter. Acceptabilities can be indicated using either one of the keywords below, or using a concept value that is < 900000000000511003 |Acceptability| . The following table lists the valid acceptability keywords in both the brief and full syntax, and their equivalent concept reference alternatives. Please note that the full syntax accepts both the brief and full syntax keywords.

Acceptability Keyword		AcceptabilityId
Brief Syntax	**Full Syntax**	Concept Reference
prefer	preferred	900000000000548007 |Preferred|
accept	acceptable	900000000000549004 |Acceptable|

For example, the following two expression constraints both match all descendants of disease with a description that matches the word prefix 'box', has the type 'synonym', and has an acceptability of 'preferred' in the en-us

language reference set. In other words, this expression constraint matches diseases with a US English preferred term that uses the word prefix 'box'.

```
< 64572001 |Disease| {{ term = "box", type = syn, dialect = en-us (prefer) }}
```

```
< 64572001 |Disease| {{ term = "box", typeId = 900000000000013009 |Synonym| , dialect = en-us ( 900000000000548007 |Preferred| ) }}
```

Multiple dialect filters may be used with different acceptabilities applied to each. For example, the expression constraint below matches on diseases, which have a synonym with word prefix "box" that is preferred in the en-nhs-clinical language reference set **and** is acceptable in the en-gb language reference set.

```
< 64572001 |Disease| {{ term = "box", type = syn, dialect = en-nhs-clinical (prefer), dialect = en-gb (accept) }}
```

To support alternative acceptabilities in more than one language reference set, a dialect set can be used. For example, the following two equivalent expression constraints match on diseases, which have a synonym with word prefix "box" that is **either** preferred in the en-gb language reference set **or** preferred in the en-nhs-clinical language reference set.

```
< 64572001 |Disease| {{ term = "box", type = syn, dialect = ( en-gb (prefer) en-nhs-clinical (prefer) ) }}
```

```
< 64572001 |Disease| {{ term = "box", type = syn, dialect = ( en-gb en-nhs-clinical ) (prefer) }}
```

Filters with Negation

Filters can use negation in a number of ways. The simplest approach is to use the 'not equal to' comparison operator (e.g. "!=") before the value.

For example, the following expression constraint matches on subtypes of |Fracture of bone| that do not use the word prefix "fracture" in their US English preferred term.

```
< 125605004 |Fracture of bone| {{ term != "fracture", type = syn, dialect = en-us (prefer)}}
```

If we remove the type and acceptability filters, as shown below, the remaining expression constraint matches on those subtypes of |Fracture of bone| which have any US English description that does not contain the word prefix "fracture". Concepts including 263171005 |Fractured nasal bones| (with synonym "Broken nose") will match the constraint below.

```
< 125605004 |Fracture of bone| {{ term != "fracture", dialect = en-us}}
```

To find the set of concepts, for which **all** descriptions match some specified criteria, the expression constraint must use the MINUS operation to exclude concepts that have a non-matching description. For example, the expression constraint below matches all subtypes of |Fracture of bone|, for which **every** description contains the word prefix

"fracture". Please note that the filter only applies to the descendants of 125605004 |Fracture of bone| (i.e. the subexpression directly proceeding the filter).

```
< 125605004 |Fracture of bone| MINUS < 125605004 |Fracture of bone| {{ term != "fracture"}}
```

This expression constraint can be simplified to the equivalent one below, using the wildcard character '*' (which represents any concept in the substrate).

```
< 125605004 |Fracture of bone| MINUS * {{ term != "fracture"}}
```

Using a similar principle, the expression constraint below matches all concepts that do not have a preferred term specified in the en-nz language reference set.

```
* MINUS * {{ type = syn, dialect = en-nz (prefer) }}
```

Module Filter

Description module filters enable an expression constraint to match on only those concepts with a matching description that belongs to a specified module. Module filters use the keyword "moduleId" with a concept reference that is < 900000000000443000 |Module| .

For example, the expression constraint below matches all subtypes of 195967001 |Asthma| with a description that belongs to the US National Library of Medicine maintained module.

```
< 195967001 |Asthma| {{ D moduleId = 731000124108 |US National Library of Medicine maintained module| }}
```

And the expression constraint below matches all subtypes of 404684003 |Clinical finding| with a definition that belongs to the international core module.

```
< 404684003 |Clinical finding| {{ D type = def, moduleId = 900000000000207008 |SNOMED CT core module| }}
```

Effective Time Filter

Description effective time filters enable an expression constraint to match on only those concepts with a description that has an effectiveTime matching the specified criteria. Effective time filters may use any of the date comparison operators shown below:

Operator	Name
=	Equals
!=	Not equals
<	Before the given date
<=	Before or on the given date
>	After the given date
>=	After or on the given date

Please note that the value of an effective time filter (if present) must be a 8 digit date, formatted according to ISO 8601's basic calendar date format (i.e. YYYYMMDD). If the effectiveTime of the description in the substrate includes a time and/or time zone designator, these should be ignored when performing the comparison.

For example, the following expression constraint matches all subtypes of 125605004 |Fracture of bone| with a description that has an effective time of 31st January 2021.

```
< 125605004 |Fracture of bone| {{ D effectiveTime = "20210131" }}
```

And the following expression constraint matches all subtypes of 125605004 |Fracture of bone| with a description that has any effective time that is *not* 31st January 2021.

```
< 125605004 |Fracture of bone| {{ D effectiveTime != "20210131" }}
```

Similarly, greater than, less than, greater than or equals and less than or equals operators may be used in an effectiveTime filter. For example, the following expression constraint matches all subtypes of 125605004 |Fracture of bone| with a description that has an effectiveTime of 31st July 2019 or later (i.e. more recent).

```
< 125605004 |Fracture of bone| {{ D effectiveTime >= "20190731" }}
```

And the following expression constraint matches all subtypes of 125605004 |Fracture of bone| with a description that has an effective time of 31st July 2019 or earlier.

```
< 125605004 |Fracture of bone| {{ D effectiveTime <= "20190731" }}
```

The effectiveTime filter can also use sets of effective times. For example, the following expression constraint matches all subtypes of 125605004 |Fracture of bone| with a description that has an effectiveTime of either 31st January 2019, 31st July 2019, 31st January 2020, or 31st July 2020.

```
< 125605004 |Fracture of bone| {{ D effectiveTime = ("20190131" "20190731" "20200131" "20200731" }}
```

And the expression constraint below matches all subtypes of 125605004 |Fracture of bone| with a description, which does *not* have any of the following effective times: 31st January 2019, 31st July 2019, 31st January 2020 or 31st July 2020.

```
< 125605004 |Fracture of bone| {{ D effectiveTime != ("20190131" "20190731" "20200131" "20200731" }}
```

To match concepts with unpublished descriptions, to which an effectiveTime has not been assigned, an effectiveTime value of "" can be used. For example, the following expression constraint matches all subtypes of 125605004 |Fracture of bone| with a description to which an effectiveTime has not yet been assigned.

```
< 125605004 |Fracture of bone| {{ D effectiveTime = "" }}
```

Please note that description effectiveTime filters, which use the comparison operators "<" and ">", will **not** match any descriptions with an effectiveTime = "".

Active Filter

Description active filters enable an expression constraint to match on only those concepts with a description that has a matching active status. Descriptions are either active (i.e. active = 1 or active = "true") or inactive (i.e. active = 0 or active = "false"). By default, only active descriptions are included in the substrate.

For example, the following expression constraints return all concepts in the International Patient Summary reference set, which have an active description.

```
^ 816080008 |International Patient Summary| {{ D active = 1 }}
```

```
^ 816080008 |International Patient Summary| {{ D active = true }}
```

And the following expression constraints return all concepts in the International Patient Summary reference set, which have an inactive description.

```
^ 816080008 |International Patient Summary| {{ D active = 0 }}
```

```
^ 816080008 |International Patient Summary| {{ D active = false }}
```

6.9 Concept Filters

In this section, we illustrate how concept filters can be applied to expression constraints to further restrict the matching concepts.

Overview

Concept filter constraints provide the ability to limit the set of concepts that satisfy a given expression constraint, based on the properties of each concept. Only concepts with properties that match the criteria specified in the concept filter constraint will be included in the set of matching concepts. Concepts can be filtered based on their definition status, module, effectiveTime, and active status. In the following sections we explain each of these concept filter criteria.

Definition Status Filter

Definition status filters enable an expression constraint to match on only those concepts with a matching definition status. Definition status filters may either use the keyword 'definitionStatus' with the values "defined" or "primitive", or may use the keyword "definitionStatusId" with a concept value that is `< 900000000000444006 |Definition status|` .

The following table lists the valid definitionStatus tokens and their equivalent definitionStatusId concept reference alternatives. If additional definition statuses are required, these must be specified in a filter using the 'definitionStatusId' keyword with the corresponding concept reference.

definitionStatus (token)	definitionStatusId (concept reference)
primitive	900000000000074008 |Not sufficiently defined by necessary conditions definition status|

definitionStatus (token)	definitionStatusId (concept reference)
defined	900000000000073002 |Sufficiently defined by necessary conditions definition status|

For example, the expression constraints below match all the primitive subtypes of |Heart disease|.

```
< 56265001 |Heart disease| {{ C definitionStatus = primitive }}
```

```
< 56265001 |Heart disease| {{ C definitionStatusId = 900000000000074008 |Primitive| }}
```

Similarly, the two expression constraints below match all the fully defined subtypes of |Heart disease|.

```
< 56265001 |Heart disease| {{ C definitionStatus = defined }}
```

```
< 56265001 |Heart disease| {{ C definitionStatusId = 900000000000073002 |Defined| }}
```

Please note that Concept filters and Description Filters can be used together to filter the results of an expression constraint based on both the properties of each concept and the properties of their descriptions. For example the following expression constraint matches all primitive subtypes of 64572001 |Disease|, which have at least one description term that includes a word starting with "heart".

```
< 64572001 |Disease| {{ C definitionStatus = primitive }} {{ D term = "heart"}}
```

Module Filter

Module filters enable an expression constraint to match on only those concepts that belong to a specified module

. Module filters use the keyword "moduleId" with a concept reference that is < 900000000000443000 |Module| .

For example, the expression constraint below matches all subtypes of 195967001 |Asthma| that belong to the US National Library of Medicine maintained module.

```
< 195967001 |Asthma| {{ C moduleId = 731000124108 |US National Library of Medicine maintained module| }}
```

And the expression constraint below matches all primitive subtypes of 195967001 |Asthma| that belong to the international core module.

```
< 195967001 |Asthma| {{ C definitionStatus = primitive, moduleId = 900000000000207008 |SNOMED CT core module| }}
```

Effective Time Filter

Effective time filters enable an expression constraint to match on only those concepts with an effectiveTime that matches the specified criteria. Effective time filters may use any of the date comparison operators shown below:

Operator	Name
=	Equals
!=	Not equals
<	Before the given date
<=	Before or on the given date
>	After the given date
>=	After or on the given date

Please note that the value of an effective time filter (if present) must be a 8 digit date, formatted according to ISO 8601's basic calendar date format (i.e. YYYYMMDD). If the effectiveTime of the concept in the substrate includes a time and/or time zone designator, these should be ignored when performing the comparison.

For example, the following expression constraint matches all subtypes of 125605004 |Fracture of bone| with an effective time of 31st January 2021.

```
< 125605004 |Fracture of bone| {{ C effectiveTime = "20210131" }}
```

And the following expression constraint matches all subtypes of 125605004 |Fracture of bone| with any effective time that is *not* 31st January 2021.

```
< 125605004 |Fracture of bone| {{ C effectiveTime != "20210131" }}
```

Similarly, greater than, less than, greater than or equals and less than or equals operators may be used in an effectiveTime filter. For example, the following expression constraint matches all subtypes of 125605004 |Fracture of bone| with an effectiveTime of 31st July 2019 or later (i.e. more recent).

```
< 125605004 |Fracture of bone| {{ C effectiveTime >= "20190731" }}
```

And the following expression constraint matches all subtypes of 125605004 |Fracture of bone| with an effective time of 31st July 2019 or earlier.

```
< 125605004 |Fracture of bone| {{ C effectiveTime <= "20190731" }}
```

The effectiveTime filter can also use sets of effective times. For example, the following expression constraint matches all subtypes of 125605004 |Fracture of bone| with an effectiveTime of either 31st January 2019, 31st July 2019, 31st January 2020, or 31st July 2020.

```
< 125605004 |Fracture of bone| {{ C effectiveTime = ("20190131" "20190731" "20200131" "20200731" }}
```

And the expression constraint below matches all subtypes of 125605004 |Fracture of bone| which does *not* have any of the following effective times: 31st January 2019, 31st July 2019, 31st January 2020 or 31st July 2020.

```
< 125605004 |Fracture of bone| {{ C effectiveTime != ("20190131" "20190731" "20200131" "20200731" }}
```

To match unpublished concepts to which an effectiveTime has not been assigned, an effectiveTime value of "" can be used. For example, the following expression constraint matches all subtypes of 125605004 |Fracture of bone| to which an effectiveTime has not yet been assigned.

```
< 125605004 |Fracture of bone| {{ C effectiveTime = "" }}
```

Please note that effectiveTime filters, which use the comparison operators "<" and ">", will **not** match any concepts with an effectiveTime = "".

Active Filter

Active filters enable an expression constraint to match on only those concepts with a matching active status. Concepts are either active (i.e. active = 1 or active = "true") or inactive (i.e. active = 0 or active = "false"). By default, both active and inactive concepts are included in the substrate. This allows inactive members of a reference set to be retrieved (e.g. for historical reference sets, in which the referenced component is intended to be inactive). However, because only active relationships are included in the default substrate, as soon as a refinement or hierarchical operator is used, only active concepts are matched.

For example, the following expression constraints returns only active concepts in the International Patient Summary reference set.

```
^ 816080008 |International Patient Summary| {{ C active = 1 }}
```

```
^ 816080008 |International Patient Summary| {{ C active = true }}
```

And the following expression constraints return only inactive concepts in the International Patient Summary reference set.

```
^ 816080008 |International Patient Summary| {{ C active = 0 }}
```

```
^ 816080008 |International Patient Summary| {{ C active = false }}
```

Please note that module filters are not intended to replace the use of simple reference sets to organize content of a particular type. Module filters are instead intended to be used for purposes related to the management of extensions or editions.

In this section, we illustrate how concept filters can be applied to expression constraints to further restrict the matching concepts.

Overview

Concept filter constraints provide the ability to limit the set of concepts that satisfy a given expression constraint, based on the properties of each concept. Only concepts with properties that match the criteria specified in the concept filter constraint will be included in the set of matching concepts. Concepts can be filtered based on their definition status, module, effectiveTime, and active status. In the following sections we explain each of these concept filter criteria.

Definition Status Filter

Definition status filters enable an expression constraint to match on only those concepts with a matching definition status. Definition status filters may either use the keyword 'definitionStatus' with the values "defined" or "primitive", or may use the keyword "definitionStatusId" with a concept value that is < 900000000000444006 |Definition status| .

The following table lists the valid definitionStatus tokens and their equivalent definitionStatusId concept reference alternatives. If additional definition statuses are required, these must be specified in a filter using the 'definitionStatusId' keyword with the corresponding concept reference.

definitionStatus (token)	definitionStatusId (concept reference)
primitive	900000000000074008 |Not sufficiently defined by necessary conditions definition status|
defined	900000000000073002 |Sufficiently defined by necessary conditions definition status|

For example, the expression constraints below match all the primitive subtypes of |Heart disease|.

```
< 56265001 |Heart disease| {{ C definitionStatus = primitive }}
```

```
< 56265001 |Heart disease| {{ C definitionStatusId = 900000000000074008 |Primitive| }}
```

Similarly, the two expression constraints below match all the fully defined subtypes of |Heart disease|.

```
< 56265001 |Heart disease| {{ C definitionStatus = defined }}
```

```
< 56265001 |Heart disease| {{ C definitionStatusId = 900000000000073002 |Defined| }}
```

Please note that Concept filters and Description Filters can be used together to filter the results of an expression constraint based on both the properties of each concept and the properties of their descriptions. For example the following expression constraint matches all primitive subtypes of 64572001 |Disease|, which have at least one description term that includes a word starting with "heart".

```
< 64572001 |Disease| {{ C definitionStatus = primitive }} {{ D term = "heart"}}
```

Module Filter

Module filters enable an expression constraint to match on only those concepts that belong to a specified module[2]. Module filters use the keyword "moduleId" with a concept reference that is < 900000000000443000 |Module| .

For example, the expression constraint below matches all subtypes of 195967001 |Asthma| that belong to the US National Library of Medicine maintained module.

```
< 195967001 |Asthma| {{ C moduleId = 731000124108 |US National Library of Medicine maintained module| }}
```

And the expression constraint below matches all primitive subtypes of 195967001 |Asthma| that belong to the international core module.

```
< 195967001 |Asthma| {{ C definitionStatus = primitive, moduleId = 900000000000207008 |SNOMED CT core module| }}
```

Effective Time Filter

Effective time filters enable an expression constraint to match on only those concepts with an effectiveTime that matches the specified criteria. Effective time filters may use any of the date comparison operators shown below:

Operator	Name
=	Equals
!=	Not equals
<	Before the given date
<=	Before or on the given date
>	After the given date
>=	After or on the given date

Please note that the value of an effective time filter (if present) must be a 8 digit date, formatted according to ISO 8601's basic calendar date format (i.e. YYYYMMDD). If the effectiveTime of the concept in the substrate includes a time and/or time zone designator, these should be ignored when performing the comparison.

For example, the following expression constraint matches all subtypes of 125605004 |Fracture of bone| with an effective time of 31st January 2021.

```
< 125605004 |Fracture of bone| {{ C effectiveTime = "20210131" }}
```

And the following expression constraint matches all subtypes of 125605004 |Fracture of bone| with any effective time that is *not* 31st January 2021.

```
< 125605004 |Fracture of bone| {{ C effectiveTime != "20210131" }}
```

Similarly, greater than, less than, greater than or equals and less than or equals operators may be used in an effectiveTime filter. For example, the following expression constraint matches all subtypes of 125605004 |Fracture of bone| with an effectiveTime of 31st July 2019 or later (i.e. more recent).

```
< 125605004 |Fracture of bone| {{ C effectiveTime >= "20190731" }}
```

And the following expression constraint matches all subtypes of 125605004 |Fracture of bone| with an effective time of 31st July 2019 or earlier.

```
< 125605004 |Fracture of bone| {{ C effectiveTime <= "20190731" }}
```

The effectiveTime filter can also use sets of effective times. For example, the following expression constraint matches all subtypes of 125605004 |Fracture of bone| with an effectiveTime of either 31st January 2019, 31st July 2019, 31st January 2020, or 31st July 2020.

```
< 125605004 |Fracture of bone| {{ C effectiveTime = ("20190131" "20190731" "20200131" "20200731" }}
```

And the expression constraint below matches all subtypes of 125605004 |Fracture of bone| which does *not* have any of the following effective times: 31st January 2019, 31st July 2019, 31st January 2020 or 31st July 2020.

```
< 125605004 |Fracture of bone| {{ C effectiveTime != ("20190131" "20190731" "20200131" "20200731" }}
```

To match unpublished concepts to which an effectiveTime has not been assigned, an effectiveTime value of "" can be used. For example, the following expression constraint matches all subtypes of 125605004 |Fracture of bone| to which an effectiveTime has not yet been assigned.

```
< 125605004 |Fracture of bone| {{ C effectiveTime = "" }}
```

Please note that effectiveTime filters, which use the comparison operators "<" and ">", will **not** match any concepts with an effectiveTime = "".

Active Filter

Active filters enable an expression constraint to match on only those concepts with a matching active status. Concepts are either active (i.e. active = 1 or active = "true") or inactive (i.e. active = 0 or active = "false"). By default, both active and inactive concepts are included in the substrate. This allows inactive members of a reference set to be retrieved (e.g. for historical reference sets, in which the referenced component is intended to be inactive). However, because only active relationships are included in the default substrate, as soon as a refinement or hierarchical operator is used, only active concepts are matched.

For example, the following expression constraints returns only active concepts in the International Patient Summary reference set.

```
^ 816080008 |International Patient Summary| {{ C active = 1 }}
```

```
^ 816080008 |International Patient Summary| {{ C active = true }}
```

And the following expression constraints return only inactive concepts in the International Patient Summary reference set.

```
^ 816080008 |International Patient Summary| {{ C active = 0 }}
```

```
^ 816080008 |International Patient Summary| {{ C active = false }}
```

Please note that module filters are not intended to replace the use of simple reference sets to organize content of a particular type. Module filters are instead intended to be used for purposes related to the management of extensions or editions.

6.10 Member Filters

In this section, we illustrate how filters can be applied to a set of reference set members to restrict the matching values.

Overview

Member filters provide the ability to filter the rows of a reference set, based on the value of specific fields in the reference set. These filters are specified inside double curly braces, and begin with the letter "M".

Member Filters

To apply a member filter to one or more reference sets, the fields of those reference sets are matched against specified criteria. Only reference set members whose field values match the given criteria will be included in the results.

For example, the following expression constraint will match all referencedComponentIds (i.e. SNOMED CT concept id) from the active 447562003 |ICD-10 complex map reference set| rows, which map to the ICD-10 code "J45.9". When applied to the July 2021 international edition, this will match 59 concepts, including 195967001 |Asthma|, 707447008 |Exacerbation of severe persistent asthma (disorder)| and 401193004 |Asthma confirmed (situation)|.

```
^ 447562003 |ICD-10 complex map reference set| {{ M mapTarget = "J45.9" }}
```

Multiple field constraints can be applied within a reference set member filter. For example, the following expression constraint will return the referencedComponentId from the 447562003 |ICD-10 complex map reference set| rows, which have a mapGroup of "2", a mapPriority of "1" and a mapTarget of "J45.9".

```
^ 447562003 |ICD-10 complex map reference set|
{{ M mapGroup = #2, mapPriority = #1, mapTarget = "J45.9" }}
```

Other comparison operators may also be used, when defining field criteria. The available operators depend on the field's datatype, as shown in the table below.

Data type	Comparison Operators	
	Brief syntax	**Long syntax**
SCTID / Expression	= , !=	= , != , NOT = , <>
Integer / Decimal	= , != , <= , < , >= , >	= , != , NOT = , <> , <= , < , >= , >

String	= , !=	= , != , NOT = , <>
Boolean	= , !=	= , != , NOT = , <>
Time	= , != , <= , < , >= , >	= , != , NOT = , <> , <= , < , >= , >

In addition, reference set fields of type 'string' may be filtered using the same word-prefix-any-order and wildcard techniques used by the description term filters. For example, the following expression constraint will match all referencedComponentId from the active 447562003 |ICD-10 complex map reference set| rows that have a mapGroup not equal to 2, a mapPriority less than 2, and a mapTarget that starts with the letter "J".

```
  ^ 447562003 |ICD-10 complex map reference set| {{ M mapGroup !
= #2, mapPriority < #2, mapTarget = wild:"J*" }}
```

Member filters can also be used in combination with the memberOf function to support the selection of other fields of a reference set (see 6.1 Simple Expression Constraints). For example, the following expression constraint returns the active SNOMED CT concept that is considered to be the same as the inactive concept 67415000 |Hay asthma|

```
  ^ [targetComponentId] 900000000000527005 |SAME AS association reference set|
{{ M referencedComponentId = 67415000 |Hay asthma| }}
```

For more information on the use of reference set field names in ECL, please refer to Appendix E - Reference Set Fields.

For additional ways of specifying queries over the historical association reference sets, please refer to 6.11 History Supplements.

6.11 History Supplements

In this section, we illustrate how history supplements can be applied to an expression constraint to supplement the results with relevant inactive concepts. History supplements are specified inside double curly braces, and being with a plus sign (i.e. "+") followed by the word "HISTORY".

Background

When capturing new clinical data in an electronic health record (EHR), it is good practice to only allow active SNOMED CT concept identifiers to be recorded. However, SNOMED CT is a dynamic and evolving terminology that must remain consistent with current clinical practice and our evolving understanding of disease processes and treatments. As a result, content may change, become outdated, or need remodelling. As SNOMED CT evolves, concepts that were previously recorded in the EHR may subsequently be inactivated. For legal reasons, it is important that the concepts used at the time the data was recorded should persist in the health records. For this reason, the number of inactive SNOMED CT identifiers in an EHR may increase over time.

As most ECL queries typically return only active SNOMED CT concept identifiers, it may not be possible to retrieve health records containing inactive identifiers using a standard expression constraint. One solution to this challenge, is to execute the expression constraint over an old SNOMED CT edition, in which all required concepts were active. However, given that the logical definitions in SNOMED CT typically improve over time, it is generally accepted that the best ECL results can be obtained using the most recent edition. Therefore, a query approach utilising the most recent edition of SNOMED CT is preferred in many cases.

When a SNOMED CT concept is inactivated, the author first allocates an appropriate reason for the inactivation, and then links the inactivated concept to one or more replacements using historical association reference sets. These historical associations provide a clear understanding of the level of semantic equivalence between the inactivated concept and its replacements where they exist. Vendors can use these historical associations to supplement the

active concepts in their query results, with inactive concepts which are linked via appropriate historical associations to the active query results.

On this page, we describe how 'history supplements' can be added to an ECL query, to augment the query results with relevant inactive concepts, and how the resulting queries can be used to retrieve a more complete set of matching health records.

History Supplements

Overview

The member filter syntax, described in 6.10 Member Filters, can be used to augment the results of an expression constraint with a set of inactive concepts that are related via an historical association reference set. For example, the following expression constraint can be used to find all the active descendants (and self) of the concept 195967001 |Asthma|, plus any inactive concept that is linked to an active descendant (or self) of 195967001 |Asthma | via a historical |SAME AS association reference set| member.

```
<< 195967001 |Asthma| OR
^ 900000000000527005 |SAME AS association reference set| {{ M targetComponentId = << 195967001 |
Asthma| }}
```

The ECL **history supplement** syntax can be used to simplify queries with this structure. For example, the above query can be expressed in a a shorter form as:

```
<< 195967001 |Asthma| {{ + HISTORY ( 900000000000527005 |SAME AS association reference set| ) }}
```

Template

The general template for history supplements is shown below.

```
[[+ecl @ecl_query]] {{ + HISTORY ( [[+ecl @history_refset_query]] ) }}
```

This general template for history supplements is equivalent to the expanded version shown below. Please note that the first and last slot in this template have the same name, which indicates that they must be populated with the same value (which in this case is the ECL query being performed).

```
[[+ecl @ecl_query]] OR
^ [[+ecl @history_refset_query]] {{ M targetComponentId = [[+ecl @ecl_query]] }}
```

Please note that this history template does not support the 900000000000525002 |MOVED FROM association reference set|, as the referencedComponentId refers to the active concept, while the targetComponentId refers to the inactive concept (which is the opposite of typical historical associations). If supporting |MOVED FROM| historical associations, it is recommended that these be added to the 900000000000527005 |SAME AS association reference set|, to ensure that the template pattern above can be consistently applied.

Also note that the 900000000000524003 |MOVED TO association reference set| can be ignored for the purposes of executing historical ECL queries.

Profiles

To help implementers of clinical systems write suitable ECL queries that include an appropriate set of inactive concepts, three history supplement profiles are provided. These profiles are designed to support a range of use cases, depending on the level of precision and recall required for inactive content. The three history supplement profiles are described in the table below.

History Profile	Purpose	Historical Association Reference Sets
HISTORY-MIN	Minimum: To support use cases requiring a high level of precision, only historical associations that have a one-to-one equivalence with their replacement are used. Example use case: Clinical decision support	900000000000527005 |SAME AS association reference set|
HISTORY-MOD	*Moderate*: To support use cases that must balance precision with recall, only historical associations that • Have a one-to-one equivalence with their replacement • Have a one-to-many equivalence with their replacement, or • Are replaced by a concept that represents the intended original meaning closely enough to be clinically useful are used. Example use cases: Clinical research, clinical audit	900000000000527005 |SAME AS association reference set| 900000000000526001 |REPLACED BY association reference set| 900000000000528000 |WAS A association reference set| 1186924009 |PARTIALLY EQUIVALENT TO association reference set|
HISTORY-MAX HISTORY (*)	*Maximum*: To support use cases that require the highest level of recall, where precision is not as important, all possible historical associations are used. Example use case: Identifying patients for manual review.	< 900000000000522004 |Historical association reference set|

For example, if a high level of precision is required, then the HISTORY-MIN profile may be used. The expression constraint below matches descendants or self of 195967001 |Asthma|, plus any inactive concept that is associated with a descendant or self of 195967001 |Asthma| in the 900000000000527005 |SAME AS association reference set| or the 900000000000525002 |MOVED FROM association reference set|

```
<< 195967001 |Asthma| {{ + HISTORY-MIN }}
```

The above expression constraint is equivalent to the one below, with an expanded history supplement.

```
<< 195967001 |Asthma| {{ + HISTORY ( 900000000000527005 |SAME AS association reference set| ) }}
```

Use cases that must balance the precision of associated inactive concepts with the level of recall, may use the HISTORY-MOD supplement. The following two expression constraint, which use the history supplement profile and the expanded history supplement respectively, are equivalent.

```
<< 195967001 |Asthma| {{ + HISTORY-MOD }}
```

```
<< 195967001 |Asthma| {{ + HISTORY ( 900000000000527005 |SAME AS association reference set| OR 900000000000526001 |REPLACED BY association reference set| OR 900000000000528000 |WAS A association reference set| OR 1186924009 |PARTIALLY EQUIVALENT TO association reference set| ) }}
```

And finally, use cases that require the highest level of recall, may use the HISTORY-MAX supplement profile. This profile uses all possible historical association reference sets to find any potentially relevant inactive concept. The following four expression constraints, which use (a) the history supplement profile, (b) the expanded history supplement, (c) the ANY wildcard symbol ('*'), and (d) the 'history' keyword on its own, are all equivalent. Please note that the 900000000000524003 |MOVED TO association reference set| does not need to be included in the execution of this query, because the targetComponentId is assigned a namespace concept

```
<< 195967001 |Asthma| {{ + HISTORY-MAX }}
```

```
<< 195967001 |Asthma| {{ + HISTORY (< 900000000000522004 |Historical association reference set| ) }}
```

```
<< 195967001 |Asthma| {{ + HISTORY (*) }}
```

```
<< 195967001 |Asthma| {{ + HISTORY }}
```

Use Case Examples

Here are two use cases that illustrate how these history supplements may be used in practice:

Use Case 1

A clinical system is trying to count the number of patients who have had any type of referral to a service. The system attempts to use the following ECL query to find patient records with a matching procedure.

```
<< 306206005 |Referral to service (procedure)|
```

This query is successfully used to finds patient records containing active referral concepts, such as 308461008 |Referral to radiology service (procedure)|.

However, it is discovered that there are 738,090 patient records coded with the inactive SNOMED CT concept 183598009 |Refer to Radiology department (procedure)|, which should also be included in the patient count. The clinical system, therefore, adjusts its expression constraint query as shown below, to add a history supplement that includes all inactive concepts with the same meaning as one of the active referral concepts.

```
<< 306206005 |Referral to service (procedure)| {{ + HISTORY-MIN }}
```

Because the expression constraint " << 306206005 |Referral to service (procedure)| " matches the active concept 308461008 |Referral to radiology service (procedure)|, and a SAME AS association exists between the inactive

concept 183598009 |Refer to Radiology department (procedure)| and the active concept 308461008 |Referral to radiology service (procedure)|, the above expression constraint will include the inactive concept 183598009 |Refer to Radiology department (procedure)|, and therefore successfully find the additional 738,090 patient records in which this inactive referral procedure is recorded.

Use Case 2

A clinician is trying to find all patients with any type of breast pain. Knowing that she will be reviewing the patient records prior to acting upon the information, she decides to use a maximal approach to searching historical records. She therefore uses the following ECL query:

```
<< 53430007 |Pain of breast (finding)| {{ + HISTORY-MAX }}
```

She is delighted to see that patient records containing the inactive concept 315251009 |Unilateral mastalgia (situation)| are retrieved, as these are indeed relevant to her query. Behind the scenes, the clinical system was able to identify that this inactive concept may be relevant, because it is linked to the active concepts 1010235008 |Pain of left breast| and 1010237000 |Pain of right breast| (which are both a type of |Pain of breast|) via the |POSSIBLY EQUIVALENT TO association reference set|.

---

1 Note that this template uses the template syntax defined in the SNOMED CT Template Syntax specification, with the addition of an 'ECL' replacement type to indicate that the respective slot must be replaced by a valid ECL expression constraint. This extended template slot syntax is then used within an expression constraint to informally illustrate the pattern required when expanding a history supplement.

7. Implementation Considerations

When implementing the SNOMED CT Expression Constraint Language, the factors that need to be taken into consideration depend on what tasks are being performed. For example, implementations may require expression constraints to be authored, parsed, validated, executed, stored, displayed or exchanged.

The subsections below look at each of these tasks individually and provide a summary of the factors that should be considered prior to implementation. Please note that the guidance provided below is not a step-by-step how-to manual, but instead provides some general insights that we hope are helpful in implementing this language specification.

7.1 Authoring

Authoring SNOMED CT Expression Constraints can be performed using two main techniques:

1. *Language-based authoring*: This technique involves the author constructing a SNOMED CT Expression Constraint using one of the syntaxes defined in Chapter 5.
2. *Form-based authoring*: This technique involves the author entering values into separate fields of a form, and the clinical system automatically composing the values together into a syntactically correct SNOMED CT Expression Constraint.

Language-Based Authoring

Language-based authoring is useful for situations in which ad hoc expression constraints must be defined which don't necessarily conform to a consistent structure. For example, some expression constraints (e.g. those that define terminology bindings or predefined queries) may be authored by software developers during the design, development or customization of a clinical application. Other expression constraints (e.g. those used to define intentional reference sets or validation queries) may be defined by terminologists during the process of developing a SNOMED CT extension. Expression constraints may also be authored by users who wish to retrieve or analyse information stored in patient records using SNOMED CT (e.g. for clinical, epidemiological or research queries).

To use language-based authoring, the user must be familiar with the basic features of the Expression Constraint Language syntax. There are, however, a number of ways in which a tool can support the user while creating expression constraints, including:

- Validating the syntactical correctness of the expression constraint as it is authored;
- Checking the expression constraint for conformance against the concept model;
- Automatically populating or correcting the term associated with a concept reference;
- Providing integrated tools to search the SNOMED CT hierarchy for concept references to include in the expression constraint;
- Filtering the concept search to those concepts which are valid to use at the given point in the expression constraint (e.g. only showing attribute concepts, or those within the valid range of the given attribute); and
- Suggesting the set of valid operators or characters that may be used at a given point in the expression constraint;

Form-Based Authoring

Form-based authoring is particularly useful when non-technical users need to create constraints or queries which have a consistent structure. In these situations, it may be useful to either:

- Create an 'expression constraint template' in which the attribute values are populated with the values that the user enters into the associated fields of the form;
- Create a form-driven query tool to support a useful subset of possible query structures.

One scenario in which the first form-based approach may be used is when there is a terminology-based dependency between the values of two fields on a user interface. For example, Figure 4 illustrates a simplified Procedures form in which the coded value entered into the *Procedure Type* field must be a descendant of the coded value entered into the *Procedure Category* field. When a *Procedure Category* of "Surgery" (i.e. 387713003 |Surgical procedure|) is

selected, the expression constraint " < 387713003 |Surgical procedure|" is used to populate the value list for the *Procedure Type* field.

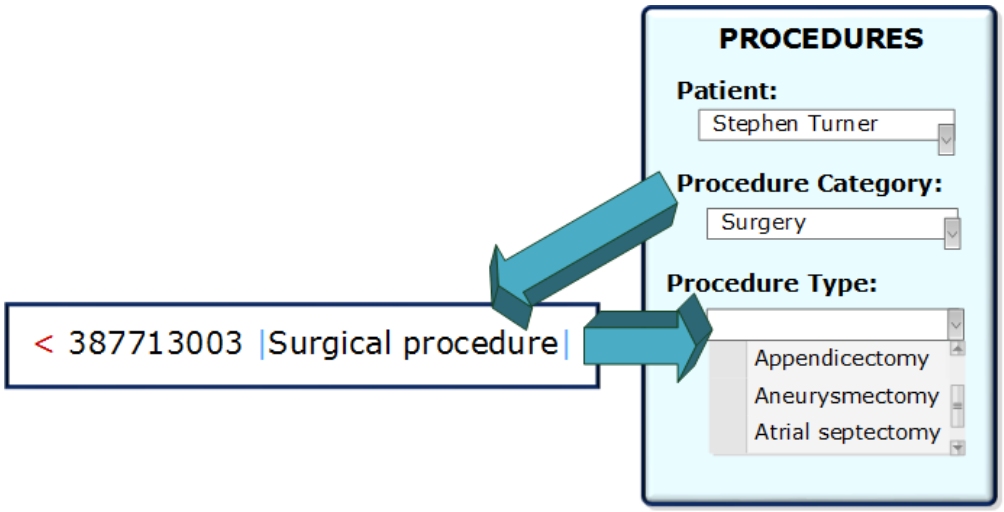

Figure 4: Authoring using expression constraint templates

The second form-based authoring technique mentioned above is a form-driven query tool. Figure 5 below illustrates a very simple form-driven query tool, in which the user selects the required operator (e.g. 'ancestorOf', 'descendantOf', 'memberOf') and operand (e.g. 'Example Problem List') and then defines one or more attribute refinements.

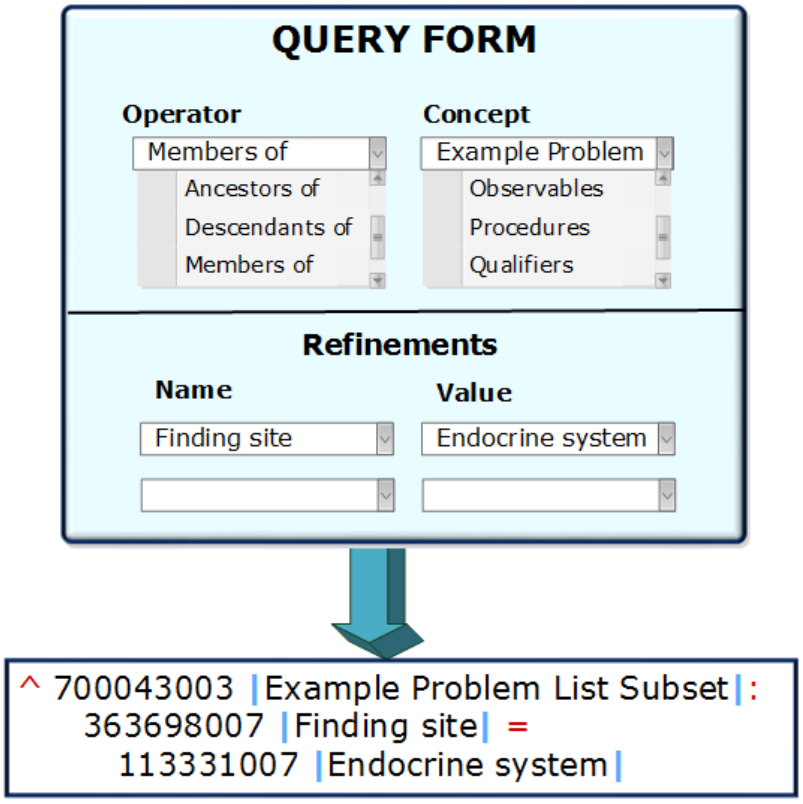

Figure 5: Authoring using a form-driven query tool

7.2 Parsing

Parsing is the process of analysing a string of characters according to the rules of a formal grammar. Parsing a SNOMED CT Expression Constraint involves processing the expression constraint string using one of the ABNF syntax specifications defined in Chapter 5, and breaking it into its constituent parts. This creates a representation of the expression constraint that can be further processed. Parsing an expression constraint is required to perform syntactic validation, concept model validation or execution. It should be noted, when parsing, that all keywords in the language are case insensitive.

A number of parser development tools are available which can generate a parser from a context-free grammar written in ABNF, such as the one defined in this document. These tools include:

- APG
- aParse
- abnfgen

Please note, the ABNF syntax defined in this specification was tested using the APG Parser Generator[1].

Other non-ABNF parser generators are also available which can be used with an alternate syntax representation – for example:

- ANTLR
- XText
- ACE

Some of these tools (e.g. XText and ACE) can also be used to generate authoring environments with features such as syntax highlighting and autocompletion.

Alternatively, an expression constraint parser can be created manually using a programming language such as Perl or C++.

---

[1] www.coasttocoastresearch.com

7.3 Validating

SNOMED CT Expression Constraints can be automatically validated to ensure that they conform to a variety of rules, including:

- Expression constraints must conform to one of the syntaxes defined in Chapter 5. Syntactic validation can be performed using an expression parser, as described in Section 7.2;
- Expression constraints must conform to the concept model. This validation can be performed by comparing the parsed expression constraint against the rules defined in the SNOMED CT concept model;
- All concept references included in the expression constraint must be valid. In most cases this means that the concept references must refer to active concepts in the given version and edition of SNOMED CT;
- All concept references used to refer to attribute names must be a descendant of 246061005 |Attribute|;
- All concept references to which a memberOf function is applied must be a descendant of 900000000000455006 |Reference set|;
- All concept references to which a memberOf function is applied must contain only referencedComponentIds that refer to concepts.

Please note that some of these rules may not apply in all environments.

7.4 Executing

SNOMED CT Expression Constraints must be evaluated against a given SNOMED CT substrate in order to instantiate the matching set of concepts or expressions. There are a number of possible implementation strategies for the execution of SNOMED CT Expression Constraints, which depend in part on the storage format of the substrate. For example:

- Store SNOMED CT in a relational database, and translate each SNOMED CT Expression Constraint into one or more SQL statements;
- Store SNOMED CT in an RDF store, and translate each SNOMED CT Expression Constraint into a SPARQL query;
- Store SNOMED CT in an XML database, and translate each SNOMED CT Expression Constraint into one or more XQL statements;
- Write a bespoke query execution engine (e.g. in Java or C++) to return matching concepts or expressions.

Each of these strategies requires that the expression constraints are first parsed (and preferably validated) prior to execution.

7.5 Storing

Storing SNOMED CT Expression Constraints in an expression constraint library may be done for a variety of purposes, including:

- To enable expression constraints to be re-executed (without re-authoring) after updates are made to the SNOMED CT substrate or the expression constraint itself;
- To provide a library of terminology binding constraints against which record instances will be validated;
- To provide a library of concept model constraints against which terminology artefacts (e.g. extensions, expressions) will be validated;
- To provide a library of predefined queries that may be shared by multiple users;
- To provide a library of terminology binding constraints that may be shared within a standards community.

A library of SNOMED CT Expression Constraints may be implemented using a number of techniques, including:

- Creating a Query specification reference set that records the expression constraint as the 'query';
- Creating a customized RF2 reference set with one or more new attributes that allow the expression constraint string and relevant metadata to be recorded;
- Creating a table in a relational database to store the SNOMED CT Expression Constraint and associated metadata;
- Creating a text file with a consistent structural format to store the SNOMED CT Expression Constraint and associated metadata;

In many cases it is useful to assign a unique identifier to each expression constraint in the library, so that they can be indexed and referenced for faster retrieval.

7.6 Displaying

A number of options exist for displaying SNOMED CT Expression Constraints, including:

- Displaying the expression constraint using SNOMED CT Expression Constraint Language in its originally authored and stored form;
- Converting the expression constraint to use either all symbols (as per the Brief Syntax), or all human-readable operators (as per alternate text introduced in the Long Syntax);
- Enhancing the expression constraint by adding in terms that may have been omitted, or replacing the existing terms with either local-dialect Preferred Terms or Fully Specified Names;
- Hiding the SNOMED CT identifiers for each concept and displaying only the Preferred Terms;
- Enhancing the display by using different font colors for each different part of the expression constraint (e.g. identifiers, terms, vertical bars, and operators), and by using whitespace in a way that improves the readability of the expression;
- Automatically transforming the expression constraint into a human-readable string using a predefined algorithm. For example, a simple algorithm may convert the symbols to text and remove the concept identifiers – e.g. "Descendants of fracture of bone: Finding site = Descendants or self of arm". More sophisticated algorithms may use pattern matching and predefined templates to construct a more natural string;
- Representing the operators, operands and attribute values of the expression constraint by populating a structured form. This approach is primarily suited to expression constraints with a consistent template, where the form can be pre-designed.

Which of these options is most appropriate to use when displaying expression constraints, will depend on a number of factors, including the type of users that will be viewing the constraints, the scope of the required constraint functionality, and the capabilities of the system implementation.

7.7 Exchanging

SNOMED CT Expression Constraints can be shared between systems and users via a number of methods, including:

- Exchanging an expression constraint string which conforms to the Brief Syntax of the Expression Constraint Language;
- Exchanging an expression constraint identifier, which can be unambiguously interpreted by the receiving system. If this approach is adopted it is recommended that an expression constraint repository is used to ensure that both the sending and receiving systems have a shared and consistent understanding of the meaning of each expression constraint.

Irrespective of the method used, it is recommended that the Brief Syntax of the SNOMED CT Expression Constraint Language be used as the normative syntax for the interoperable sharing of expression constraints.

Appendix A – Examples Of Valid Expressions

This appendix provides examples of expressions (both precoordinated and postcoordinated) which satisfy each of the expression constraints that were introduced in Chapter 6. This list of examples is not intended to be exhaustive, but rather to provide a representative sample to help clarify the meaning of each constraint. It is assumed that each particular usage of an expression constraint will clearly identify whether or not postcoordinated expressions are part of the valid substrate. Please refer to the SNOMED CT Languages Github repository for a set of text files containing each of these examples.

A.1 Simple Expression Constraints - Valid Expressions

Expression Constraint	**Valid Expression** [1]	
	Precoordinated	**Postcoordinated**
404684003 |Clinical finding|	404684003 |Clinical finding|	-
< 404684003 |Clinical finding|	64572001 |Disease| 56265001 |Heart disease|	404684003 |Clinical finding| : 363698007 |Finding site| = 80891009 |Heart structure|
<< 73211009 |Diabetes mellitus|	73211009 |Diabetes mellitus| 46635009 |Diabetes mellitus type 1| 105401000119101 |Diabetes mellitus due to pancreatic injury|	73211009 |Diabetes mellitus| : 42752001 |Due to| = 61823004 |Injury of pancreas|
<! 404684003 |Clinical finding|	64572001 |Disease| 267038008 |Edema|	404684003 |Clinical finding| : 116676008 |Associated morphology| = 79654002 |Edema| [2]
> 40541001 |Acute pulmonary edema|	111273006 |Acute respiratory disease| 404684003 |Clinical finding| 138875005 |SNOMED CT concept|	64572001 |Disease| : 116676008 |Associated morphology| = 79654002 |Edema| , 363698007 |Finding site| = 39607008 |Lung structure|
>> 40541001 |Acute pulmonary edema|	40541001 |Acute pulmonary edema| 111273006 |Acute respiratory disease| 404684003 |Clinical finding| 138875005 |SNOMED CT concept|	64572001 |Disease| : 263502005 |Clinical course| = 424124008 |Sudden onset AND/OR short duration| , { 116676008 |Associated morphology| = 40829002 |Acute edema| , 363698007 |Finding site| = 39607008 |Lung structure| }
>! 40541001 |Acute pulmonary edema|	111273006 |Acute respiratory disease| 19242006 |Pulmonary edema|	19829001 |Disorder of lung| : { 116676008 |Associated morphology| = 79654002 |Edema| , 363698007 |Finding site| = 39607008 |Lung structure| } [3]

^ 700043003 |Example problem list concepts reference set|	394659003 |Acute coronary syndrome|	-
	194828000 |Angina|	
	29857009 |Chest pain|	
*	138875005 |SNOMED CT concept|	404684003 |Clinical finding| : 363698007 |Finding site| = 80891009 |Heart structure|
	404684003 |Clinical finding|	71388002 |Procedure| : 405813007 |Procedure site - Direct| = 66754008 |Appendix structure|
	322236009 |Paracetamol 500mg tablet|	373873005 |Pharmaceutical / biologic product| : { 127489000 |Has active ingredient| = 412031009 |Paracetamol or derivative| }

1. Where necessary, these examples make some assumptions about the membership of the example reference sets.
2. Please note that this makes the assumption that the given expression constraint is executed against a finite set of expressions that has been pre-classified (e.g. in an expression repository), and that after classification there are no intermediate expressions between this expression and 404684003 |Clinical finding|.
3. Please note that this makes the assumption that the given expression constraint is executed against a finite set of expressions that has been pre-classified (e.g. in an expression repository), and that after classification there are no intermediate expressions between 40541001 |Acute pulmonary edema| and this expression.

A.2 Refinements - Valid Expressions

Expression Constraint	**Valid Expression** [1] [2]	
	Precoordinated	**Postcoordinated**
< 19829001 |Disorder of lung| : 116676008 |Associated morphology| = 79654002 |Edema|	11468004 |Postoperative pulmonary edema|	210051003 |Injury to heart and lung| : 116676008 |Associated morphology| = 79654002 |Edema|
	276637009 |Hemorrhagic pulmonary edema|	
< 19829001 |Disorder of lung| : 116676008 |Associated morphology| = << 79654002 |Edema|	233709006 |Toxic pulmonary edema|	275504005 |Lung cyst| : 116676008 |Associated morphology| = 103619005 |Inflammatory edema|
	233711002 |Oxygen-induced pulmonary edema|	19829001 |Disorder of lung| : 116676008 |Associated morphology| = 40829002 |Acute edema|

< 404684003 |Clinical finding| : 363698007 |Finding site| = << 39057004 |Pulmonary valve structure| , 116676008 |Associated morphology| = << 415582006 |Stenosis|	56786000 |Pulmonic valve stenosis|	56786000 |Pulmonic valve stenosis| : 363698007 |Finding site| = 90318009 |Structure of anulus fibrosus of pulmonary artery| , 116676008 |Associated morphology| = 88015002 |Partial stenosis|
	86299006 |Tetralogy of Fallot|	404684003 |Clinical finding| : 363698007 |Finding site| = 39057004 |Pulmonary valve structure| , 116676008 |Associated morphology| = 415582006 |Stenosis|
* : 246075003 |Causative agent| = 387517004 |Paracetamol|	295124009 |Paracetamol overdose|	404684003 |Clinical finding| : 246075003 |Causative agent| = 387517004 |Paracetamol|
	292042007 |Adverse reaction to paracetamol|	
< 404684003 |Clinical finding| : { 363698007 |Finding site| = << 39057004 |Pulmonary valve structure| , 116676008 |Associated morphology| = << 415582006 |Stenosis| }, { 363698007 |Finding site| = << 53085002 |Right ventricular structure| , 116676008 |Associated morphology| = << 56246009 |Hypertrophy| }	86299006 |Tetralogy of Fallot|	404684003 |Clinical finding| : { 363698007 |Finding site| = 31689007 |Structure of cusp of pulmonic valve| , 116676008 |Associated morphology| = 415582006 |Stenosis| }, { 363698007 |Finding site| = 53085002 |Right ventricular structure| , 116676008 |Associated morphology| = 125521000 |Acute hypertrophy| }
	204351007 |Fallot's trilogy|	
<< 404684003 |Clinical finding| : << 47429007 |Associated with| = << 267038008 |Edema|	230580009 |Myxedema neuropathy|	95356008 |Mucosal ulcer| : 42752001 |Due to| = 19242006 |Pulmonary edema|
< 27658006 |Amoxicillin| : 411116001 |Has dose form| = << 385055001 |Tablet dose form| , { 179999999100 |Has basis of strength| = (219999999102 |Amoxicillin only| : 189999999103 |Has strength magnitude| >= #200, 199999999101 |Has strength unit| = 258684004 |mg|)}	374644001 |Amoxicillin trihydrate 200 mg tablet|	27658006 |Amoxicillin| : 411116001 |Has dose form| = 421026006 |Oral tablet| , { 127489000 |Has active ingredient| = 96068000 |Amoxicillin trihydrate| , 179999999100 |Has basis of strength| = (219999999102 |Amoxicillin only| : 189999999103 |Has strength magnitude| = #500, 199999999101 |Has strength unit| = 258684004 |mg|)}

< 27658006 |Amoxicillin| : 411116001 |Has dose form| = << 385055001 |Tablet dose form| , { 179999999100 |Has basis of strength| = (219999999102 |Amoxicillin only| : 189999999103 |Has strength magnitude| >= #500, 189999999103 |Has strength magnitude| <= #800, 199999999101 |Has strength unit| = 258684004 |mg|) }	374646004 |Amoxicillin 500 mg tablet|	27658006 |Amoxicillin| : 411116001 |Has dose form| = 421026006 |Oral tablet| , { 179999999100 |Has basis of strength| = (219999999102 |Amoxicillin only| : 189999999103 |Has strength magnitude| = #750, 199999999101 |Has strength unit| = 258684004 |mg|)}
< 373873005 |Pharmaceutical / biologic product| : 209999999104 |Has trade name| = "PANADOL"	259999999103 |PANADOL [paracetamol] tablet|	373873005 |Pharmaceutical / biologic product| : { 127489000 |Has active ingredient| = 412031009 |Paracetamol or derivative| }, 209999999104 |Has trade name| = "PANADOL"
< 91723000 |Anatomical structure| : R 363698007 |Finding site| = < 125605004 |Fracture of bone|	85050009 |Humerus|	85050009 |Humerus| : 272741003 |Laterality| = 7771000 |Left|
	71341001 |Femur|	71341001 |Femur| : 272741003 |Laterality| = 24028007 |Right|
< 125605004 |Fracture of bone| . 363698007 |Finding site|	85050009 |Humerus|	85050009 |Humerus| : 272741003 |Laterality| = 7771000 |Left|
	71341001 |Femur|	71341001 |Femur| : 272741003 |Laterality| = 24028007 |Right|
< 105590001 |Substance| : R << 127489000 |Has active ingredient| = < 27658006 |Product containing amoxicillin|	395938000 |Clavulanate potassium|	-
	387137007 |Omeprazole|	
< 27658006 |Product containing amoxicillin| . << 127489000 |Has active ingredient|	395938000 |Clavulanate potassium|	-
	387137007 |Omeprazole|	
< 404684003 |Clinical finding| : * = 79654002 |Edema|	19242006 |Pulmonary edema|	404684003 |Clinical finding| : 116676008 |Associated morphology| = 79654002 |Edema|
	97341000119105 Proliferative retinopathy with retinal edema due to type	

< 404684003 |Clinical finding| : 116676008 |Associated morphology| = *	19242006 |Pulmonary edema|	404684003 |Clinical finding| : 116676008 |Associated morphology| = 79654002 |Edema|
	263225007 |Hip fracture|	404684003 |Clinical finding| : 116676008 |Associated morphology| = 72704001 |Fracture|

1. Please note that some of these examples are based on a hypothetical drug concept model. These examples are not intended to reflect any specific drug model.
2. SNOMED CT identifiers with the '9999999' namespace were created for example only, and should not be used in a production environment.

A.3 Cardinality - Valid Expressions

Expression Constraint	**Valid Expression** [1]	
	Precoordinated	**Postcoordinated**
< 373873005 |Pharmaceutical / biologic product| : [1..3] 127489000 |Has active ingredient| =< 105590001 |Substance|	322236009 |Paracetamol 500mg tablet|	373873005 |Pharmaceutical / biologic product| : { 127489000 |Has active ingredient| = 412031009 |Paracetamol or derivative| }
	404826002 |Benzocaine + butamben + tetracaine hydrochloride|	373873005 |Pharmaceutical / biologic product| : { 127489000 |Has active ingredient| = 412031009 |Paracetamol or derivative| }, { 127489000 |Has active ingredient| = 387494007 |Codeine| }
< 373873005 |Pharmaceutical / biologic product| : [1..1] 127489000 |Has active ingredient| =< 105590001 |Substance|	370166004 |Aspirin 325mg tablet|	373873005 |Pharmaceutical / biologic product| : { 127489000 |Has active ingredient| = 412031009 |Paracetamol or derivative| }
< 373873005 |Pharmaceutical / biologic product| : [0..1] 127489000 |Has active ingredient| =< 105590001 |Substance|	279999999108 |Inert tablet|	373873005 |Pharmaceutical / biologic product| : { 127489000 |Has active ingredient| = 412031009 |Paracetamol or derivative| }
	370166004 |Aspirin 325mg tablet|	
< 373873005 |Pharmaceutical / biologic product| : [1..*] 127489000 |Has active ingredient| =< 105590001 |Substance|	7947003 |Aspirin|	373873005 |Pharmaceutical / biologic product| : { 127489000 |Has active ingredient| = 412031009 |Paracetamol or derivative| }, { 127489000 |Has active ingredient| = 255641001 |Caffeine| }, { 127489000 |Has active ingredient| = 387458008 |Aspirin| }
	437867004 |Chlorphenamine + dextromethorphan + paracetamol + pseudoephedrine|	

< 404684003 |Clinical finding| : [1..1] 363698007 |Finding site| = < 91723000 |Anatomical structure|	125596004 |Injury of elbow|	404684003 |Clinical finding| : { 116676008 |Associated morphology| = 72704001 |Fracture| , 363698007 |Finding site| = 299701004 |Bone of forearm| , 363698007 |Finding site| = 62413002 |Bone structure of radius| }
< 404684003 |Clinical finding| : [2..*] 363698007 |Finding site| = < 91723000 |Anatomical structure|	86299006 |Tetralogy of Fallot|	404684003 |Clinical finding| : { 116676008 |Associated morphology| = 72704001 |Fracture| , 363698007 |Finding site| = 299701004 |Bone of forearm| }, { 116676008 |Associated morphology| = 72704001 |Fracture| , 363698007 |Finding site| = 702468001 |Bone structure of lower leg| }
< 404684003 |Clinical finding| : { [2..*] 363698007 |finding site| = < 91723000 |Anatomical structure| }	-	64572001 |Disease| : { 116676008 |Associated morphology| = 396351009 |Congenital septal defect| , 363698007 |Finding site| = 25943004 |Structure of atrioventricular node| , 363698007 |Finding site| = 113262008 |Thoracic aorta structure| } { 116676008 |Associated morphology| = 90141005 |Congenital hypertrophy| , 363698007 |Finding site| = 244384009 |Entire right ventricle| }
< 373873005 |Pharmaceutical / biologic product| : [1..3] { [1..*] 127489000 |Has active ingredient| =< 105590001 |Substance| }	322236009 |Paracetamol 500mg tablet|	373873005 |Pharmaceutical / biologic product| : { 127489000 |Has active ingredient| = 412031009 |Paracetamol or derivative| }
	404826002 |Benzocaine + butamben + tetracaine hydrochloride|	373873005 |Pharmaceutical / biologic product| : { 127489000 |Has active ingredient| = 412031009 |Paracetamol or derivative| }, { 127489000 |Has active ingredient| = 387494007 |Codeine| }
< 373873005 |Pharmaceutical / biologic product| : [0..1] { 127489000 |Has active ingredient| =< 105590001 |Substance| }	111115279999999108 |Inert tablet|	373873005 |Pharmaceutical / biologic product| : { 127489000 |Has active ingredient| = 412031009 |Paracetamol or derivative| }
	370166004 |Aspirin 325mg tablet|	
< 373873005 |Pharmaceutical / biologic product| : [1..*] { 127489000 |Has active ingredient| =< 105590001 |Substance| }	370166004 |Aspirin 325mg tablet|	373873005 |Pharmaceutical / biologic product| : { 127489000 |Has active ingredient| = 412031009 |Paracetamol or derivative| }, { 127489000 |Has active ingredient| = 387494007 |Codeine| }
< 404684003 |Clinical finding| : [1..1] { 363698007 |Finding site| = < 91723000 |Anatomical structure| }	125596004 |Injury of elbow|	404684003 |Clinical finding| : { 363698007 |Finding site| = 299701004 |Bone of forearm| }, { 363698007 |Finding site| = 62413002 |Bone structure of radius| }

< 404684003 |Clinical finding| : [0..0] { [2..*] 363698007 |Finding site| = < 91723000 |Anatomical structure| }	86299006 |Tetralogy of Fallot|	404684003 |Clinical finding| : 363698007 |Finding site| = 39057004 |Pulmonary valve structure| , 116676008 |Associated morphology| = 415582006 |Stenosis|

1 The SNOMED CT identifiers created with the '9999999' namespace are for example only, and should not be used in a production environment.

2 As mentioned earlier, only non-redundant defining attributes are included in the cardinality count. Because 62413002 |Bone structure of radius| is a subtype of 299701004 |Bone of forearm|, the refinement " 363698007 |Finding site| = 299701004 |Bone of forearm|" is redundant.

A.4 Conjunction and Disjunction - Valid Expressions

Expression Constraint	**Valid Expression** [1]	
	Precoordinated	**Postcoordinated**
< 19829001 |Disorder of lung| AND < 301867009 |Edema of trunk|	233709006 |Toxic pulmonary edema|	233709006 |Toxic pulmonary edema| : 116676008 |Associated morphology| = 40829002 |Acute edema| , 363698007 |Finding site| = 278985004 |Fissure of right lung|
	61233003 |Silo-fillers' disease|	
< 19829001 |Disorder of lung| OR < 301867009 |Edema of trunk|	363358000 |Malignant tumour of lung|	233709006 |Toxic pulmonary edema| : 116676008 |Associated morphology| = 40829002 |Acute edema|
	19242006 |Pulmonary edema|	
< 19829001 |Disorder of lung| AND ^ 700043003 |Example problem list concepts reference set|	1001000119102 |Pulmonary embolism with pulmonary infarction|	
< 404684003 |Clinical finding| : 363698007 |Finding site| = << 39057004 |Pulmonary valve structure| AND 116676008 |Associated morphology| = << 415582006 |Stenosis|	91442002 |Rheumatic pulmonary valve stenosis|	56786000 |Pulmonic valve stenosis| : 363698007 |Finding site| = 90318009 |Structure of anulus fibrosus of pulmonary artery| , 116676008 |Associated morphology| = 88015002 |Partial stenosis|
	86299006 |Tetralogy of Fallot|	

< 404684003 |Clinical finding| : 116676008 |Associated morphology| = << 55641003 |Infarct| OR 42752001 |Due to| = << 22298006 |Myocardial infarction|	45456005 |Renal infarct| 703326006 |Mitral regurgitation due to acute myocardial infarction|	95281009 |Sudden cardiac death| : 42752001 |Due to| = 22298006 |Myocardial infarction|
< 404684003 |Clinical finding| : { 363698007 |Finding site| =<< 39057004 |Pulmonary valve structure| , 116676008 |Associated morphology| =< 415582006 |Stenosis| } OR { 363698007 |Finding site| = << 53085002 |Right ventricular structure| , 116676008 |Associated morphology| =< 56246009 |Hypertrophy| }	85971001 |Rheumatic pulmonary valve stenosis with insufficiency| 86299006 |Tetralogy of Fallot|	56786000 |Pulmonic valve stenosis| : 363698007 |Finding site| = 90318009 |Structure of anulus fibrosus of pulmonary artery| , 116676008 |Associated morphology| = 88015002 |Partial stenosis|
^ 450990004 |Adverse drug reactions reference set for GP/FP health issue| : 246075003 |Causative agent| = (< 373873005 |Pharmaceutical / biologic product| OR < 105590001 |Substance|)	294811002 |Corticotrophic hormone allergy| 293584003 |Paracetamol allergy| 293585002 |Salicylate allergy|	-
< 404684003 |Clinical finding| : 116676008 |Associated morphology| = (<< 56208002 |Ulcer| AND << 50960005 |Hemorrhage|)	12847006 |Acute duodenal ulcer with hemorrhage|	64572001 |Disease| : { 116676008 |Associated morphology| = 55075001 |Bleeding ulcer| , 363698007 |Finding site| = 14374004 |Structure of lymphatic vessel of oesophagus| }

[1] Where necessary, these examples make some assumptions about the membership of the example reference sets.

A.5 Exclusion and Not Equals - Valid Expressions

Expression Constraint	Valid Expression [1]	
	Precoordinated	**Postcoordinated**

<< 19829001 |Disorder of lung| MINUS
  << 301867009 |Edema of trunk|
	372146004 |Acute chest syndrome|
	413839001 |Chronic lung disease|
		27819004 |Pulmonary ossification| :
		  { 116676008 |Associated morphology| =
		  18115005 |Pathologic calcification| ,
		  363698007 |Finding site| =
		  31094006 |Structure of lobe of lung| }

<< 19829001 |Disorder of lung| MINUS
  ^ 700043003 |Example problem list concepts reference set|
	233613009 |Fungal pneumonia|	27819004 |Pulmonary ossification| :
		  { 116676008 |Associated morphology| =
		  18115005 |Pathologic calcification| ,
		  363698007 |Finding site| =
		  31094006 |Structure of lobe of lung| }

< 404684003 |Clinical finding| :
  116676008 |Associated morphology| =
  ((<< 56208002 |Ulcer| AND
  << 50960005 |Hemorrhage|) MINUS
  << 26036001 |Obstruction|)
	15902003 |Gastric ulcer with hemorrhage|	64572001 |Disease| :
		  { 116676008 |Associated morphology| =
		  55075001 |Bleeding ulcer| ,
		  363698007 |Finding site| =
		  14374004 |Structure of lymphatic vessel of esophagus| }

< 404684003 |Clinical finding| :
  116676008 |Associated morphology| !=
  << 26036001 |Obstruction|
	233613009 |Fungal pneumonia|
	46708007 |Acute gastric ulcer with hemorrhage AND obstruction|
		64572001 |Disease| :
		  { 116676008 |Associated morphology| =
		  26036001 |Obstruction| ,
		  363698007 |Finding site| =
		  422897007 |Vascular structure of stomach| }
		  { 116676008 |Associated morphology| =
		  45771005 |Acute bleeding ulcer| ,
		  363698007 |Finding site| =
		  422897007 |Vascular structure of stomach| }

< 404684003 |Clinical finding| : [0..0]
  116676008 |Associated morphology| =
  << 26036001 |Obstruction|
	233613009 |Fungal pneumonia|
	15902003 |Gastric ulcer with hemorrhage|
		64572001 |Disease| :
		  { 116676008 |Associated morphology| =
		  55075001 |Bleeding ulcer| ,
		  363698007 |Finding site| =
		  14374004 |Structure of lymphatic vessel of oesophagus| }

< 404684003 |Clinical finding| : [0..0]
  116676008 |Associated morphology| !=
  << 26036001 |Obstruction|
	244815007 |Pyloric obstruction|
	84906002 |Local cyanosis|
		64572001 |Disease| :
		  { 116676008 |Associated morphology| =
		  26036001 |Obstruction| ,
		  363698007 |Finding site| =
		  314600001 |Choledochoenterostomy stoma| }

< 404684003 |Clinical finding| :
  [0..0] 116676008 |Associated morphology|
  != << 26036001 |Obstruction| AND
  [1..*] 116676008 |Associated morphology|
  = << 26036001 |Obstruction|
	244815007 |Pyloric obstruction|	64572001 |Disease| :
		  { 116676008 |Associated morphology| =
		  26036001 |Obstruction| ,
		  363698007 |Finding site| =
		  314600001 |Choledochoenterostomy stoma| }

[1] Where necessary, these examples make some assumptions about the membership of the example reference sets.

A.6 Nested Expression Constraints - Valid Expressions

Expression Constraint	Valid Expression	
	Precoordinated	**Postcoordinated**
<< (^ 700043003 |Example problem list concepts reference set|)	394659003 |Acute coronary syndrome|	194828000 |Angina| : 255234002 |After| = 22298006 |Myocardial infarction|
	194828000 |Angina|	
	371807002 |Atypical angina|	
^ (< 450973005 |GP/FP health issue reference set|)	140004 |Chronic pharyngitis|	-
	297009 |Acute myringitis|	
(< 404684003 |Clinical finding| : 363698007 |Finding site| =<< 39057004 |Pulmonary valve structure|) AND ^ 700043003 |Example problem list concepts reference set|	204351007 |Fallot's trilogy|	-
	457652006 |Calcification of pulmonary valve|	
(< 404684003 |Clinical finding| : 363698007 |Finding site| =<< 39057004 |Pulmonary valve structure|) AND (< 64572001 |Disease| : 116676008 |Associated morphology| =<< 415582006 |Stenosis|)	204351007 |Fallot's trilogy|	19036004 |Rheumatic heart valve stenosis| : { 363698007 |Finding site| = 39057004 |Pulmonary valve structure| , 116676008 |Associated morphology| = 415582006 |Stenosis| }
	56786000 |Pulmonic valve stenosis|	
(<< 17636008 |Specimen collection| : 424226004 |Using device| =<< 19923001 |Catheter|) . 363701004 |Direct substance|	78014005 |Urine|	-
	87612001 |Blood|	
(<< 404684003 |Clinical finding (finding)| OR << 272379006 |Event (event)|): 255234002 |After| =<< 71388002 |Procedure (procedure)|	235948002 |Postoperative acute pancreatitis|	64572001 |Disease| : { 370135005 |Pathological process| = 441862004 |Infectious process| , 255234002 |After| = 387713003 |Surgical procedure| , 116676008 |Associated morphology| = 112633009 |Surgical would| }
	441795000 |Infected seroma after surgical procedure|	
<< 125605004 |Fracture of bone| : [0..0] ((<< 410662002 |Concept model attribute| MINUS 363698007 |Finding site|) MINUS 116676008 |Associated morphology|) = *	125605004 |Fracture of bone|	64572001 |Disease| : { 363698007 |Finding site| = 71341001 |Bone structure of femur| , 116676008 |Associated morphology| = 20946005 |Fracture, closed| }
	439987009 |Open fracture of bone|	

< 404684003 |Clinical finding| : 47429007 |Associated with| = (< 404684003 |Clinical finding| : 116676008 |Associated morphology| = << 55641003 |Infarct|)	71023004 |Pericarditis secondary to acute myocardial infarction|	3238004 |Pericarditis (disorder)| : 47429007 |Associated with| = 57054005 |Acute myocardial infarction|

[1] Where necessary, these examples make some assumptions about the membership of the example reference sets.

Appendix B – Examples Of Invalid Expressions

This appendix provides examples of expressions (both precoordinated and postcoordinated) which **do not** satisfy the given expression constraints from Chapter 6. This list of examples is not intended to be exhaustive, but rather to provide a useful sample to help clarify the meaning of these constraint. Please refer to the SNOMED CT Languages Github repository for a set of text files containing each of these examples.

B.1 Simple Expression Constraints - Invalid Expressions

Expression Constraint	**INVALID Expression** [1]	
	Precoordinated	**Postcoordinated**
404684003 |Clinical finding|	56265001 |Heart disease|	404684003 |Clinical finding| : 363698007 |Finding site| = 80891009 |Heart structure|
	71388002 |Procedure|	
< 404684003 |Clinical finding|	404684003 |Clinical finding|	71388002 |Procedure| : 405813007 |Procedure site - Direct| = 80891009 |Heart structure|
	71388002 |Procedure|	
<< 73211009 |Diabetes mellitus|	71388002 |Procedure|	404684003 |Clinical finding| : 363698007 |Finding site| = 113331007 |Structure of endocrine system|
	362969004 |Disorder of endocrine system|	
<! 404684003 |Clinical finding|	404684003 |Clinical finding|	404684003 |Clinical finding| : 116676008 |Associated morphology| = 79654002 |Edema| , 363698007 |Finding site| = 80891009 |Heart structure| [2]
	233709006 |Toxic pulmonary edema|	
> 40541001 |Acute pulmonary edema|	40541001 |Acute pulmonary edema|	40541001 |Acute pulmonary edema| : 246112005 |Severity| = 24484000 |Severe|
	233709006 |Toxic pulmonary edema|	
	304527002 |Acute asthma|	
>> 40541001 |Acute pulmonary edema|	233709006 |Toxic pulmonary edema|	40541001 |Acute pulmonary edema| : 246112005 |Severity| = 24484000 |Severe|
	304527002 |Acute asthma|	
>! 40541001 |Acute pulmonary edema|	404684003 |Clinical finding|	64572001 |Disease| : 263502005 |Clinical course| = 424124008 |Sudden onset AND/OR short duration| [3]
	267038008 |Edema|	
^ 700043003 |Example problem list concepts reference set|	6143009 |Diabetic education|	71388002 |Procedure| : 405813007 |Procedure site - Direct| = 80891009 |Heart structure|
	75367002 |Blood pressure|	
*	-	-

	-	-
	-	-

1. Where necessary, these examples make some assumptions about the membership of the example reference sets.
2. Please note that this makes the assumption that the given expression constraint is executed against a finite set of expressions that has been pre-classified (e.g. in an expression repository), and that after classification there is at least one intermediate expression between this expression and 404684003 |Clinical finding|.
3. Please note that this makes the assumption that the given expression constraint is executed against a finite set of expressions that has been pre-classified (e.g. in an expression repository), and that after classification there is at least one intermediate expression between 40541001 |Acute pulmonary edema| and this expression.

B.2 Refinements - Invalid Expressions

Expression Constraint	**INVALID Expression** [1] [2]	
	Precoordinated	**Postcoordinated**
< 19829001 |Disorder of lung| : 116676008 |Associated morphology| = 79654002 |Edema|	19829001 |Disorder of lung|	19829001 |Disorder of lung| : 116676008 |Associated morphology| = 44132006 |Abscess|
	73452002 |Abscess of lung|	19829001 |Disorder of lung| : 116676008 |Associated morphology| = 40829002 |Acute edema|
	233711002 |Oxygen-induced pulmonary edema|	
< 19829001 |Disorder of lung| : 116676008 |Associated morphology| =< 79654002 |Edema|	19829001 |Disorder of lung|	6141006 |Retinal edema| : 116676008 |Associated morphology| = 103619005 |Inflammatory edema|
	73452002 |Abscess of lung|	19829001 |Disorder of lung| : 116676008 |Associated morphology| = 44132006 |Abscess|
	6141006 |Retinal edema|	
< 404684003 |Clinical finding| : 363698007 |Finding site| =< 39057004 |Pulmonary valve structure| , 116676008 |Associated morphology| =< 415582006 |Stenosis|	404684003 |Clinical finding|	448643005 |Abnormality of pulmonary valve| : 116676008 |Associated morphology| = 44132006 |Abscess|
	448643005 |Abnormality of pulmonary valve|	404684003 |Clinical finding| : 363698007 |Finding site| = 61853006 |Spinal canal structure| , 116676008 |Associated morphology| = 415582006 |Stenosis|
	431238002 |Abscess of pulmonary valve|	

* : 246075003 |Causative agent| = 387517004 |Paracetamol|	46093004 |Paracetamol measurement|	404684003 |Clinical finding| : 246075003 |Causative agent| = 372687004 |Amoxicillin|
< 404684003 |Clinical finding| : { 363698007 |Finding site| = << 39057004 |Pulmonary valve structure| , 116676008 |Associated morphology| = << 415582006 |Stenosis| }, { 363698007 |Finding site| = << 53085002 |Right ventricular structure| , 116676008 |Associated morphology| = << 56246009 |Hypertrophy| }	404684003 |Clinical finding| 56786000 |Pulmonary valve stenosis|	404684003 |Clinical finding| : { 363698007 |Finding site| = 39057004 |Pulmonary valve structure| , 116676008 |Associated morphology| = 56246009 |Hypertrophy| }, { 363698007 |Finding site| = 53085002 |Right ventricular structure| , 116676008 |Associated morphology| = 415582006 |Stenosis| }
<< 404684003 |Clinical finding| : << 47429007 |Associated with| = << 267038008 |Edema|	404684003 |Clinical finding|	95356008 |Mucosal ulcer| : 42752001 |Due to| = 59901004 |Cheek biting|
< 27658006 |Amoxicillin| : 411116001 |Has dose form| = << 385055001 |Tablet dose form| , { 179999999100 |Has basis of strength| = (219999999102 |Amoxicillin only| : 189999999103 |Has strength magnitude| >= #200, 199999999101 |Has strength unit| = 258684004 |mg|) }	269999999100 |Amoxicillin capsule| 374233002 |Amoxicillin trihydrate 125 mg chewable tablet|	27658006 |Amoxicillin| : 411116001 |Has dose form| = 421026006 |Oral tablet| , { 179999999100 |Has basis of strength| = (219999999102 |Amoxicillin only| : 189999999103 |Has strength magnitude| = 175, 199999999101 |Has strength unit| = 258684004 |mg|)}

< 27658006 |Amoxicillin| : 411116001 |Has dose form| =<< 385055001 |Tablet dose form| , { 179999999100 |Has basis of strength| = (219999999102 |Amoxicillin only| : 189999999103 |Has strength magnitude| >= #500, 189999999103 |Has strength magnitude| <= #800, 199999999101 |Has strength unit| = 258684004 |mg|) }	269999999100 |Amoxicillin capsule|	27658006 |Amoxicillin| : 411116001 |Has dose form| = 421026006 |Oral tablet| , { 179999999100 |Has basis of strength| = (219999999102 |Amoxicillin only| : 189999999103 |Has strength magnitude| = #850, 199999999101 |Has strength unit| = 258684004 |mg|)}
	374647008 |Amoxicillin 875 mg tablet|	
< 373873005 |Pharmaceutical / biologic product| : 209999999104 |Has trade name| = "PANADOL"	373873005 |Pharmaceutical / biologic product|	373873005 |Pharmaceutical / biologic product| : { 127489000 |Has active ingredient| = 412031009 |Paracetamol or derivative| , 209999999104 |Has trade name| = "PANADEINE"}
	322236009 |Paracetamol 500mg tablet|	
< 91723000 |Anatomical structure| : R 363698007 |Finding site| =< 125605004 |Fracture of bone|	34080009 |Malleus structure|	34080009 |Malleus structure| : 272741003 |Laterality| = 7771000 |Left|
	10200004 |Liver structure|	10200004 |Liver structure| : 272741003 |Laterality| = 24028007 |Right|
< 125605004 |Fracture of bone| . 363698007 |Finding site|	34080009 |Malleus structure|	34080009 |Malleus structure| : 272741003 |Laterality| = 7771000 |Left|
	10200004 |Liver structure|	10200004 |Liver structure| : 272741003 |Laterality| = 24028007 |Right|
< 105590001 |Substance| : R << 127489000 |Has active ingredient| = < 27658006 |Product containing amoxicillin|	105590001 |Substance|	373873005 |Pharmaceutical / biologic product| : 127489000 |Has active ingredient| = 372687004 |Amoxicillin|

	387517004 |Paracetamol|	
249999999101 |TRIPHASIL tablet| . 127489000 |Has active ingredient|	105590001 |Substance|	373873005 |Pharmaceutical / biologic product| : 127489000 |Has active ingredient| = 126109000 |Levonorgestrel|
	387517004 |Paracetamol|	
< 404684003 |Clinical finding| : * = 79654002 |Edema|	263225007 |Hip fracture|	404684003 |Clinical finding| : 116676008 |Associated morphology| = 72704001 |Fracture|
	385933006 |Edema control education|	
< 404684003 |Clinical finding| : 116676008 |Associated morphology| = *	195967001 |Asthma|	404684003 |Clinical finding| : 363698007 |Finding site| = 80891009 |Heart structure|
	73211009 |Diabetes mellitus|	404684003 |Clinical finding| : 246075003 |Causative agent| = 372687004 |Amoxicillin|

1 Please note that some of these examples are based on a hypothetical drug concept model.
2 The SNOMED CT identifiers created with the '9999999' namespace are for example only, and should not be used in a production environment.

B.3 Cardinality - Invalid Expressions

Expression Constraint	**INVALID Expression** [1]	
	Precoordinated	**Postcoordinated**
< 373873005 |Pharmaceutical / biologic product| : [1..3] 127489000 |Has active ingredient| = < 105590001 |Substance|	279999999108 |Inert tablet|	373873005 |Pharmaceutical / biologic product| : { 127489000 |Has active ingredient| = 412031009 |Paracetamol or derivative| }, { 127489000 |Has active ingredient| = 387494007 |Codeine| }, { 127489000 |Has active ingredient| = 255641001 |Caffeine| }, { 127489000 |Has active ingredient| = 44068004 |Doxylamine| }
	437867004 |Chlorphenamine + dextromethorphan + paracetamol + pseudoephedrine|	
< 373873005 |Pharmaceutical / biologic product| : [1..1] 127489000 |Has active ingredient| = < 105590001 |Substance|	279999999108 |Inert tablet|	373873005 |Pharmaceutical / biologic product| : { 127489000 |Has active ingredient| = 412031009 |Paracetamol or derivative| }, { 127489000 |Has active ingredient| = 387494007 |Codeine| }
	412556009 |Paracetamol + codeine|	

< 373873005 |Pharmaceutical / biologic product| : [0..1] 127489000 |Has active ingredient| = < 105590001 |Substance|	412556009 |Paracetamol + codeine|	373873005 |Pharmaceutical / biologic product| : { 127489000 |Has active ingredient| = 412031009 |Paracetamol or derivative| }, { 127489000 |Has active ingredient| = 387494007 |Codeine| }
< 373873005 |Pharmaceutical / biologic product| : [1..*] 127489000 |Has active ingredient| = < 105590001 |Substance|	279999999108 |Inert tablet|	373873005 |Pharmaceutical / biologic product| : 411116001 |Has dose form| = 385055001 |Tablet|
< 404684003 |Clinical finding| : [1..1] 363698007 |Finding site| = < 91723000 |Anatomical structure|	75857000 |Fracture of radius and ulna| 40733004 |Infectious disease|	404684003 |Clinical finding| : { 116676008 |Associated morphology| = 72704001 |Fracture| , 363698007 |Finding site| = 62413002 |Bone structure of radius| , 363698007 |Finding site| = 23416004 |Bone structure of ulna| }
< 404684003 |Clinical finding| : [2..*] 363698007 |Finding site| = < 91723000 |Anatomical structure|	23406007 |Arm fracture| 40733004 |Infectious disease|	404684003 |Clinical finding| : { 116676008 |Associated morphology| = 72704001 |Fracture| , 363698007 |Finding site| = 702468001 |Bone structure of lower leg| }
< 404684003 |Clinical finding| : { [2..*] 363698007 |Finding site| = < 91723000 |Anatomical structure| }	75857000 |Fracture of radius and ulna|	64572001 |Disease| : { 116676008 |Associated morphology| = 396351009 |Congenital septal defect| , 363698007 |Finding site| = 113262008 |Thoracic aorta structure| } { 116676008 |Associated morphology| = 90141005 |Congenital hypertrophy| , 363698007 |Finding site| = 244384009 |Entire right ventricle| }
< 373873005 |Pharmaceutical / biologic product| : [1..3] { [1..*] 127489000 |Has active ingredient| = < 105590001 |Substance| }	279999999108 |Inert tablet| 437867004 |Chlorphenamine + dextromethorphan + paracetamol + pseudoephedrine|	373873005 |Pharmaceutical / biologic product| : { 127489000 |Has active ingredient| = 412031009 |Paracetamol or derivative| }, { 127489000 |Has active ingredient| = 387494007 |Codeine| }, { 127489000 |Has active ingredient| = 255641001 |Caffeine| }, { 127489000 |Has active ingredient| = 44068004 |Doxylamine| }
< 373873005 |Pharmaceutical / biologic product| : [0..1] { 127489000 |Has active ingredient| = < 105590001 |Substance| }	412556009 |Paracetamol + codeine|	373873005 |Pharmaceutical / biologic product| : { 127489000 |Has active ingredient| = 412031009 |Paracetamol or derivative| }, { 127489000 |Has active ingredient| = 387494007 |Codeine| }
< 373873005 |Pharmaceutical / biologic product| : [1..*] { 127489000 |Has active ingredient| = < 105590001 |Substance| }	279999999108 |Inert tablet|	373873005 |Pharmaceutical / biologic product| : 411116001 |Has dose form| = 385055001 |Tablet|

< 404684003 |Clinical finding| : [1..1] { 363698007 |Finding site| = < 91723000 |Anatomical structure| }	75857000 |Fracture of radius and ulna|	404684003 |Clinical finding| : { 116676008 |Associated morphology| = 72704001 |Fracture| , 363698007 |Finding site| = 62413002 |Bone structure of radius| }, { 116676008 |Associated morphology| = 72704001 |Fracture| , 363698007 |Finding site| = 23416004 |Bone structure of ulna| }
	40733004 |Infectious disease|	
< 404684003 |Clinical finding| : [0..0] { [2..*] 363698007 |Finding site| = < 91723000 |Anatomical structure| }	-	64572001 |Disease| : { 116676008 |Associated morphology| = 396351009 |Congenital septal defect| , 363698007 |Finding site| = 25943004 |Structure of atrioventricular node| , 363698007 |Finding site| = 113262008 |Thoracic aorta structure| } { 116676008 |Associated morphology| = 90141005 |Congenital hypertrophy| , 363698007 |Finding site| = 244384009 |entire right ventricle| }

The SNOMED CT identifiers created with the '9999999' namespace are for example only, and should not be used in a production environment.

B.4 Conjunction and Disjunction - Invalid Expressions

Expression Constraint	**INVALID Expression**	
	Precoordinated	**Postcoordinated**
< 19829001 |Disorder of lung| AND < 301867009 |Edema of trunk|	73452002 |Abscess of lung|	248508001 |Abdominal wall edema| : 116676008 |Associated morphology| = 40829002 |Acute edema|
	248508001 |Abdominal wall edema|	
< 19829001 |Disorder of lung| OR < 301867009 |Edema of trunk|	19829001 |Disorder of lung|	128121009 |Disorder of trunk| : 116676008 |Associated morphology| = 44132006 |Abscess|
	301867009 |Edema of trunk|	
	128121009 |Disorder of trunk|	
< 19829001 |Disorder of lung| AND ^ 700043003 |Example problem list concepts reference set|	73452002 |Abscess of lung|	19829001 |Disorder of lung| : 116676008 |Associated morphology| = 44132006 |Abscess|
< 404684003 |Clinical finding| : 363698007 |Finding site| =<< 39057004 |Pulmonary valve structure| AND 116676008 |Associated morphology| =<< 415582006 |Stenosis|	301104003 |Pulmonary valve finding|	404684003 |Clinical finding| : 116676008 |Associated morphology| = 88015002 |Partial stenosis|
	60573004 |Aortic valve stenosis|	

< 404684003 |Clinical finding| : 116676008 |Associated morphology| = < 55641003 |Infarct| OR 42752001 |Due to| = << 22298006 |Myocardial infarction|	368009 |Heart valve disorder| 461089003 |Cardiac abnormality due to heart abscess|	95281009 |Sudden cardiac death| : 42752001 |Due to| = 10633002 |Acute congestive heart failure|
< 404684003 |Clinical finding| : { 363698007 |Finding site| = << 39057004 |Pulmonary valve structure| , 116676008 |Associated morphology| = < 415582006 |Stenosis| } OR { 363698007 |Finding site| = << 53085002 |Right ventricular structure| , 116676008 |Associated morphology| = < 56246009 |Hypertrophy| }	93075009 |Congenital hypertrophy of pulmonary valve| 204370002 |Stenosis of infundibulum of right ventricle|	404684003 |Clinical finding| : 363698007 |Finding site| = 39057004 |Pulmonary valve structure| , 116676008 |Associated morphology| = 56246009 |Hypertrophy|
^ 450990004 |Adverse drug reactions reference set for GP/FP health issue| : 246075003 |Causative agent| = (< 373873005 |Pharmaceutical / biologic product| OR < 105590001 |Substance|)	87628006 |Bacterial infectious disease| 609328004 |Allergic disposition| 10629471000119106 |Allergic rhinitis caused by mould|	609328004 |Allergic disposition| : 246075003 |Causative agent| = 84489001 |Mold|
< 404684003 |Clinical finding| : 116676008 |Associated morphology| = (<< 56208002 |Ulcer| AND << 50960005 |Hemorrhage|)	196652006 |Acute duodenal ulcer| 74474003 |Gastrointestinal haemorrhage|	64572001 |Disease| : 116676008 |Associated morphology| = 405719001 |Chronic ulcer|

1 Where necessary, these examples make some assumptions about the membership of the example reference sets.

B.5 Exclusion and Not Equals - Invalid Expressions

Expression Constraint	**INVALID Expression**	
	Precoordinated	**Postcoordinated**
<< 19829001 |Disorder of lung| MINUS << 301867009 |Edema of trunk|	27719009 |Acute gastrointestinal hemorrhage| 19242006 |Pulmonary edema|	19829001 |Disorder of lung| : { 116676008 |Associated morphology| = 40829002 |Acute edema| , 363698007 |Finding site| = 22943007 |Trunk structure| }
<< 19829001 |Disorder of lung| MINUS ^ 700043003 |Example problem list concepts reference set|	67599009 |Pulmonary congestion|	67599009 |Pulmonary congestion| : 363698007 |Finding site| = 3341006 |Right lung structure|

< 404684003 |Clinical finding| : 116676008 |Associated morphology| = ((<< 56208002 |Ulcer| AND << 50960005 |Hemorrhage|) MINUS << 26036001 |Obstruction|)	397825006 |Gastric ulcer| 235670001 |Gastric stomal obstruction|	64572001 |Disease| : 116676008 |Associated morphology| = 26036001 |Obstruction|
< 404684003 |Clinical finding| : 116676008 |Associated morphology| != << 26036001 |Obstruction|	81060008 |Intestinal obstruction| 56265001 |Heart disease|	64572001 |Disease| : 116676008 |Associated morphology| = 26036001 |Obstruction| , 363698007 |Finding site| = 422897007 |Vascular structure of stomach|
< 404684003 |Clinical finding| : [0..0] 116676008 |Associated morphology| = << 26036001 |Obstruction|	81060008 |Intestinal obstruction| 234059001 |Venous stenosis|	64572001 |Disease| : { 116676008 |Associated morphology| = 26036001 |Obstruction| , 363698007 |Finding site| = 422897007 |Vascular structure of stomach| } { 116676008 |Associated morphology| = 45771005 |Acute bleeding ulcer| , 363698007 |Finding site| = 422897007 |Vascular structure of stomach| }
< 404684003 |Clinical finding| : [0..0] 116676008 |Associated morphology| != << 26036001 |Obstruction|	196652006 |Acute duodenal ulcer| 8377001 |Hernia, with obstruction|	64572001 |Disease| : { 116676008 |Associated morphology| = 26036001 |Obstruction| , 363698007 |Finding site| = 422897007 |Vascular structure of stomach| } { 116676008 |Associated morphology| = 45771005 |Acute bleeding ulcer| , 363698007 |Finding site| = 422897007 |Vascular structure of stomach| }
< 404684003 |Clinical finding| : [0..0] 116676008 |Associated morphology| != << 26036001 |Obstruction| AND [1..*] 116676008 |Associated morphology| = << 26036001 |Obstruction|	196652006 |Acute duodenal ulcer| 8377001 |Hernia, with obstruction|	64572001 |Disease| : { 116676008 |Associated morphology| = 26036001 |Obstruction| , 363698007 |Finding site| = 422897007 |Vascular structure of stomach| } { 116676008 |Associated morphology| = 45771005 |Acute bleeding ulcer| , 363698007 |Finding site| = 422897007 |vascular structure of stomach| }
	56265001 |Heart disease|	64572001 |Disease| : { 116676008 |Associated morphology| = 45771005 |Acute bleeding ulcer| , 363698007 |Finding site| = 422897007 |Vascular structure of stomach| }

B.6 Nested Expression Constraints - Invalid Expressions

Expression Constraint	Valid Expression	
	Precoordinated	**Postcoordinated**
<< (^ 700043003 |Example problem list concepts reference set|)	6143009 |Diabetic education|	71388002 |Procedure| : 405813007 |Procedure site - Direct| = 80891009 |Heart structure|
	75367002 |Blood pressure|	
^ (< 450973005 |GP/FP health issue reference set|)	80146002 |Appendectomy|	-
	305342007 |Admission to ward|	
(< 404684003 |Clinical finding| : 363698007 |Finding site| =<< 39057004 |Pulmonary valve structure|) AND ^ 700043003 |Example problem list concepts reference set|	125605004 |Fracture of bone|	404684003 |Clinical finding| : 363698007 |Finding site| = 17401000 |Cardiac valve structure|
	195967001 |Asthma|	
(< 404684003 |Clinical finding| : 363698007 |Finding site| =<< 39057004 |Pulmonary valve structure|) AND (< 64572001 |Disease| : 116676008 |Associated morphology| =<< 415582006 |Stenosis|)	301104003 |Pulmonary valve finding|	404684003 |Clinical finding| : 363698007 |Finding site| = 39057004 |Pulmonary valve structure|
	76107001 |Spinal stenosis|	64572001 |Disease| : 116676008 |Associated morphology| = 415582006 |Stenosis|
(<< 17636008 |Specimen collection| : 424226004 |Using device| =<< 19923001 |Catheter|) . 363701004 |Direct substance|	57617002 |Urine specimen collection|	17636008 |Specimen collection| : 424226004 |Using device| = 19923001 |Catheter|
	122575003 |Urine specimen|	
(<< 404684003 |Clinical finding (finding)| OR << 272379006 |Event (event)|): 255234002 |After| =<< 71388002 |Procedure (procedure)|	293690005 |Peppermint oil allergy|	404684003 |Clinical finding| : 255234002 |After| = 417163006 |Injury|
	82510005 |Posttraumatic vertigo|	
<< 125605004 |Fracture of bone| : [0..0] ((<< 410662002 |Concept model attribute| MINUS 363698007 |Finding site|) MINUS 116676008 |Associated morphology|) = *	704333004 |Pathological fracture of hand due to osteoporosis|	125605004 |Fracture of bone| : 42752001 |Due to| = 417163006 |Injury|
	722571004 |Linear fracture of skull due to birth trauma|	
< 404684003 |Clinical finding| : 47429007 |Associated with| = (< 404684003 |Clinical finding| : 116676008 |Associated morphology| =<< 55641003 |Infarct|)	3238004 |Pericarditis|	64572001 |Disease| : 47429007 |Associated with| = (404684003 |Clinical finding| : 363698007 |Finding site| = 277712000 |Cardiac internal structure|)

1. Where necessary, these examples make some assumptions about the membership of the example reference sets.

Appendix C - Dialect Aliases

This appendix provides a list of example aliases that may be used to specify a particular dialect in an ECL filter constraint. Please refer to the 'Dialect Filter' section on 6.8 Description Filters for more information.

The table below lists the valid 'dialect' filter values and their equivalent 'dialectId' filter values, for a selection of known language reference sets. The dialect aliases shown below use the following format (defined using ABNF):

dialectAlias = language ["-" realm ["-" useCase]]

language = alpha alpha ; conforms to ISO 639-1 2 character language codes

realm = alpha *alpha ; if realm is a country then conforms to ISO 3166-1 2 character country codes

useCase = alpha *(alpha / integerValue) ; the clinical scope or context of use

dialect	dialectId
da-dk	554461000005103 |Danish language reference set|
en-au	32570271000036106 |Australian English language reference set|
en-ca	19491000087109 |Canada English language reference set|
en-gb	900000000000508004 |Great Britain English language reference set|
en-ie	21000220103 |Irish language reference set|
en-nz	271000210107 |New Zealand English language reference set|
en-nz-pat	281000210109 |New Zealand English patient friendly terms language reference set|
en-us	900000000000509007 |United States of America English language reference set|
en-int-gmdn	608771002 |GMDN language reference set|
en-nhs-clinical	999001261000000100 |National Health Service realm language reference set (clinical part)|
en-nhs-dmd	999000671000001103 |National Health Service dictionary of medicines and devices realm language reference set|
en-nhs-pharmacy	999000691000001104 |National Health Service realm language reference set (pharmacy part)|
en-uk-drug	999000681000001101 |United Kingdom Drug Extension Great Britain English language reference set|
en-uk-ext	999001251000000103 |United Kingdom Extension Great Britain English language reference set|
es	450828004 |Conjunto de referencias de lenguaje castellano para América Latina|
es-uy	5641000179103 |Conjunto de referencias de lenguaje castellano para Uruguay|
et-ee	71000181105 |Estonian language reference set|
de	722130004 |German language reference set|
fr	722131000 |French language reference set|
fr-be	21000172104 |Belgian French language reference set|
fr-ca	20581000087109 |Canada French language reference set|
ja	722129009 | Japanese language reference set|
mi	291000210106 |Maori language reference set|
nl-be	31000172101 |Belgian Dutch language reference set|
nl-nl	31000146106 |Netherlands Dutch language reference set|
nb-no	61000202103 |Norwegian Bokmål language reference set|
nn-no	91000202106 |Norwegian Nynorsk language reference set|
sv-se	46011000052107 |Swedish language reference set|

dialect	dialectId
zh	722128001 |Chinese language reference set|

Appendix D - ECL Quick Reference

This section provides a quick reference to the key syntax features of the Expression Constraint Language.

Syntax Overview

The following table summarises the key symbols used in the Expression Constraint Language's brief syntax, with the ECL version in which each symbol was introduced. For more information about the version history of ECL, please refer to the 'History' section in 1. Introduction.

Symbol	**Name**	Version	Notes
|	Pipe	1.0	Used on either side of a concept's term for human readability
*	Any	1.0	Retrieves all concepts in the substrate
^	Member of	1.0	Retrieves the referencedComponentId of all (active) members of a reference set (or set of reference sets)
^ [A, B]	Member of (with field selection)	2.0	Retrieves the values of fields A and B of all (active) members of a reference set (or set of reference sets) that match the included Member filters (if applicable)
<	Descendant of	1.0	Retrieves all descendants (subtypes) of the specified concept *excluding* the concept itself
<<	Descendant or self of	1.0	Retrieves all descendants (subtypes) of the specified concept *including* the concept itself
<!	Child of	1.1	Retrieves all children (immediate subtypes) of the specified concept *excluding* the concept itself
<<!	Child or self of	1.4	Retrieves all children (immediate subtypes) of the specified concept *including* the concept itself
>	Ancestor of	1.0	Retrieves all ancestors (supertypes) of the specified concept *excluding* the concept itself
>>	Ancestor or self of	1.0	Retrieves all ancestors (supertypes) of the specified concept *including* the concept itself
>!	Parent of	1.1	Retrieves all parents (immediate supertypes) of the specified concept *excluding* the concept itself
>>!	Parent or self of	1.4	Retrieves all parents (immediate supertypes) of the specified concept *including* the concept itself
AND	Conjunction	1.0	Retrieves the intersection of the results of each sub-expressions
OR	Disjunction	1.0	Retrieves the union of the results of each sub-expressions
MINUS	Exclusion	1.0	Retrieves the members of the first expression and excludes the members returned by the second expression
:	Refinement	1.0	Used before one or more attribute-value pairs to refine the set of concepts retrieved
[1..3]	Cardinality	1.0	Used to indicate the minimum and maximum number of occurrences of attributes or relationship groups
R	Reverse flag	1.0	Retrieves the set of attribute values (i.e. destination concepts) of a specified attribute for a specified set of concepts
.	Dot notation	1.1	Retrieves the set of attribute values (i.e. destination concepts) of a specified attribute for a specified set of concepts

Symbol	Name	Version	Notes
/* */	Comment	1.1	Allows comments to be added within the text of an expression constraint
{{ }}	Description filter	1.5	Filters the result set, by matching only on concepts which have a description with a matching term, language, type, dialect and/or acceptability
{{ D }}	Description filter	1.6	Filters the result set, by matching only on concepts which have a description with a matching term, language, type, dialect and/or acceptability
{{ C }}	Concept filter	1.6	Filters the result set based on the definition status, module, effectiveTime and active status of each concept
{{ M }}	Member filter	2.0	Filters the result set based on the value of specific fields in a reference set.
{{+ HISTORY}}	History supplement	2.0	Supplements the results with relevant inactive concepts

Examples

The following table provides some examples of each of the key syntax features of the Expression Constraint Language.

Notes:

1. In the table above:
   - '**id**' represents a single SNOMED CT concept identifier,
   - '**term**' represents a term associated with the concept identified by '**id**',
   - '**x**', '**y**' and '**v**' each represent either a single concept or a set of concepts defined using an expression constraint,
   - '**z**' represents either a single concept or a set of concepts that are a subtype of 900000000000455006 |Reference set|,
   - '**a**' and '**b**' each represent either a single concept or a set of concepts that are a subtype of 410662002 |Concept model attribute|, and
   - '**min**' and '**max**' are two numeric values that represent the minimum and maximum cardinality allowed.
2. The default substrate, to which expression constraints are applied, includes all concepts, active relationships, active descriptions and active reference set members of a chosen SNOMED CT versioned edition.

Simple expression constraints			
Syntax	Evaluation Notes	Example	Example Expansion Concepts
id | term |	Only the concept with the identifier 'id'	128477000 |Abscess|	128477000 |Abscess|
*	All concepts in the given substrate	*	*Any concept in the given substrate*
^ z	The set of concepts which are members of the reference sets in z	^ 723264001 |Lateralizable body structure reference set|	181216001 |Entire lung| 65784005 |Structure of fundus of eye|

< x	The set of all descendants (both direct and indirect) of x	< 73211009 |Diabetes mellitus| < 73211009 |Diabetes mellitus)|	46635009 |Diabetes mellitus type 1| 8801005 |Secondary diabetes mellitus|
<< x	The set of all descendants (both direct and indirect) of x, plus x itself	<< 73211009 |Diabetes mellitus|	73211009 |Diabetes mellitus| 46635009 |Diabetes mellitus type 1| 8801005 |Secondary diabetes mellitus|
<! x	The set of all immediate children of x	<! 362965005 |Disorder of body system|	49601007 |Disorder of cardiovascular system| 362969004 |Disorder of endocrine system|
<<! x	The set of all immediate children of x, plus x itself	<<! 362965005 |Disorder of body system|	362965005 |Disorder of body system| 49601007 |Disorder of cardiovascular system| 362969004 |Disorder of endocrine system|
> x	The set of all ancestors (both direct and indirect) of x	> 279420009 |Hematoma of skin|	106076001 |Skin finding| 297968009 |Bleeding skin|
>> x	The set of all ancestors (both direct and indirect) of x, plus x itself	>> 279420009 |Hematoma of skin|	106076001 |Skin finding| 297968009 |Bleeding skin| 279420009 |Hematoma of skin|
>! x	The set of all immediate parents of x	>! 22298006 |Myocardial infarction|	57809008 |Myocardial disease| 251061000 |Myocardial necrosis|
>>! x	The set of all immediate parents of x, plus x itself	>>! 22298006 |Myocardial infarction|	22298006 |Myocardial infarction| 57809008 |Myocardial disease| 251061000 |Myocardial necrosis|
Conjunction, Disjunction and Exclusion			
Syntax	Evaluation Notes	Example	Example Expansion Concepts

x **AND** y	The set of concepts that are both in x and in y (i.e. the intersection of x and y)	< 19829001 |Disorder of lung| AND < 87628006 |Bacterial infectious disease|	430395005 |Pneumonia caused by Gram negative bacteria| 154283005 |Pulmonary tuberculosis|
x **OR** y	The set of concepts that are either in x or in y (i.e. the union of x and y)	< 73452002 |Abscess of lung| OR < 275504005 |Cyst of lung|	446543007 |Tuberculous abscess of lung| 87119009 |Congenital cystic lung|
x **MINUS** y	The set of concepts that are in x but are not in y (i.e. x excluding concepts in y)	< 29303009 |Electrocardiographic procedure| MINUS < 75444003 |Fetal electrocardiogram|	447114004 |12 lead electrocardiogram during exercise| 252417001 |24 Hour electrocardiogram|
Refinement			
Syntax	Evaluation Notes	Example	Example Expansion Concepts
x **:** a **=** y	The set of concepts in **x**, which have a necessary relationship with an attribute in **a** and a value in **y**	< 385494008 |Hematoma| : << 370135005 |Pathological process| = << 441862004 |Infectious process|	698573001 |Infected hematoma| 444109008 |Infection of wound hematoma|
x **:** a **=** y**,** b **=** v	The set of concepts in **x**, which have both a necessary relationship with an attribute in **a** and a value in **y**, and also have a necessary relationship (either the same one or a different one) with an attribute in **b** and a value in **v**	< 71388002 |Procedure| : << 363704007 |Procedure site| = << 69695003 |Stomach structure| , << 405815000 |Procedure device| =<< 86174004 |Laparoscope|	708987006 |Laparoscopic total gastrectomy| 57922004 |Laparoscopic pyloromyotomy|
x **:{** a **=** y**,** b **=** v **}**	The set of concepts in **x**, which have a role group that contains both a necessary relationship with an attribute in **a** and a value in **y**, and also have a necessary relationship (either the same one or a different one) with an attribute in **b** and a value in **v**	< 71388002 |Procedure (procedure)| :{ 405813007 |Procedure site - Direct| =<< 10200004 |Liver structure| , 260686004 |Method| = << 129433002 |Inspection - action| }	773252007 |Diagnostic laparoscopy of liver| 20933000 |Endoscopy of liver|
Cardinality			
Syntax	Evaluation Notes	Example	Example Expansion Concepts
x **: [min .. max]** a **=** y	The set of concepts in **x**, which have between **min** and **max** necessary relationships with an attribute in **a** and a value in **y**	< 373873005 |Pharmaceutical / biologic product| : [3..*] 127489000 |Has active ingredient| =< 105590001 |Substance|	786732006 |Product containing only brompheniramine and codeine and phenylpropanolamine| 787979009 |Product containing cyanocobalamin and folic acid and pyridoxine|

x **: [min .. max] {** a **=** y **}**	The set of concepts in **x**, which have between **min** and **max** role groups that contain a necessary relationship with an attribute in **a** and a value in **y**	< 404684003 |Clinical finding| : [2..3]{ 363698007 |Finding site| = *, 116676008 |Associated morphology| = 72704001 |Fracture| }	271577005 |Fracture of shaft of tibia and fibula| 75857000 |Fracture of radius AND ulna|
Reversed Attributes			
Syntax	Evaluation Notes	Example	Example Expansion Concepts
y **: R** a **=** x	The set of concepts in y, which are the destination (ie attribute value) of a necessary relationship on a source concept in **x** with an attribute in **a**	< 91723000 |Anatomical structure| : R 363698007 |Finding site| = < 445945000 |Infectious disease associated with acquired immune deficiency syndrome|	280369009 |Brain tissue structure| 39607008 |Lung structure| 395939008 |Clavulanic acid (substance)|
x **.** a	The set of attribute values (ie destination concepts) of all necessary relationships on a source concept in **x** with an attribute in **a**	< 27658006 |Product containing amoxicillin| . 127489000 |Has active ingredient|	372687004 |Amoxicillin| 395939008 |Clavulanic acid|

Appendix E - Reference Set Fields

In the SNOMED CT Release File Specification (http://snomed.org/rfs), SNOMED International specifies a set of reference set types with their own specific properties (e.g. an attribute value type reference set). Each reference set that is developed to conform to a specified type is defined as a subtype of the associated reference set type concept (e.g. 900000000000480006 |Attribute value type reference set|). All reference sets of a given type are populated with members using the same data structure - with the same set of field names in the same order. SNOMED International uses these reference set type data structures (as defined in the Release File Specification) as the release file format for all reference sets of that type.

All reference set type concepts are a subtype of 900000000000455006 |Reference set|, and have an associated set of reference set descriptors in the |Reference set descriptor reference set|. Some reference set type concepts are organised under one or more reference set groups (e.g. 723564002 |MRCM reference set|), which represent a group of reference set types (often with different data structures).

In the Expression Constraint Language (v2.0+) reference set field names are used to indicate which field values to return, and to filter reference set members based on specific field criteria. The first (non-metadata) field in every reference set (in order '0') must always be 'referencedComponentId'. For reference sets, which are a subtype of an international reference set type, the additional field names defined in the SNOMED CT Release File Specification must be used. In all other cases, the additional field names may use any latin-script alphabetic character (a-z or A-Z) defined by the owner of the corresponding reference set type concept. Owners of a reference set type are encouraged to explicitly document these field names, keep them unchanged and publish a machine readable representation of these (following the format used below). In the absence of this, the column name from the corresponding RF2 file (with all whitespace removed) will be used.

The international reference set types and their corresponding list of field names to be used in ECL v2.0+ are shown in the table below (for information only). A normative, computable representation of this table is attached below the table. Please note that this file may be extended by implementers with national or local reference set types.

Content Reference Set Types	
Reference Set Type	**Field Names**
446609009 |Simple type reference set|	referencedComponentId
733619002 |Ordered component type reference set|	referencedComponentId,order
900000000000480006 |Attribute value type reference set|	referencedComponentId,valueId
900000000000521006 |Association type reference set|	referencedComponentId,targetComponentId
733618005 |Ordered association type reference set|	referencedComponentId,targetComponentId,order
900000000000516008 |Annotation type reference set|	referencedComponentId,annotation
900000000000512005 |Query specification type reference set|	referencedComponentId,query
447258008 |Ordered type reference set|	referencedComponentId,order,linkedToId
762676003 |OWL expression type reference set|	referencedComponentId,owlExpression
1119417006 |Postcoordinated expression type reference set|	referencedComponentId,expression,substrate

Language Reference Set Types	
Reference Set Type	**Field Names**
900000000000506000 |Language type reference set|	referencedComponentId,acceptabilityId
Map Reference Set Types	
Reference Set Type	**Field Names**
900000000000496009 |Simple map from SNOMED CT type reference set|	referencedComponentId,mapTarget
1187636009 |Simple map to SNOMED CT type reference set|	referencedComponentId,mapSource
447250001 |Complex map from SNOMED CT type reference set|	referencedComponentId,mapGroup,mapPriority,mapRule,mapAdvice,mapTarget,correlationId
609331003 |Extended map from SNOMED CT type reference set|	referencedComponentId,mapGroup,mapPriority,mapRule,mapAdvice,mapTarget,correlationId,mapCategoryId
705111002 |Map to SNOMED CT with correlation and origin type reference set|	referencedComponentId,mapSource,attributeId,correlationId,contentOriginId
705109006 |Code to expression type reference set type reference set|	referencedComponentId,mapSource,expression,definitionStatusId,correlationId,contentOriginId
1193542003 |Simple map with correlation from SNOMED CT type reference set|	referencedComponentId,mapTarget,correlationId
1193543008 |Simple map with correlation to SNOMED CT type reference set|	referencedComponentId,mapSource,correlationId
1193544002 |Simple map with correlation from SNOMED CT to SNOMED CT type reference set|	referencedComponentId,mapTarget,correlationId
Metadata Reference Set Types	
Reference Set Type	**Field Names**
900000000000456007 |Reference set descriptor type reference set|	referencedComponentId,attributeDescription,attributeType,attributeOrder
900000000000534007 |Module dependency type reference set|	referencedComponentId,sourceEffectiveTime,targetEffectiveTime
900000000000538005 |Description format type reference set|	referencedComponentId,descriptionFormat,descriptionLength
723589008 |MRCM domain type reference set|	referencedComponentId,domainConstraint,parentDomain,proximalPrimitiveConstraint,proximalPrimitiveRefinement,domainTemplateForPrecoordination,domainTemplateForPostcoordination,guideURL
723604009 |MRCM attribute domain type reference set|	referencedComponentId,domainId,grouped,attributeCardinality,attributeInGroupCardinality,ruleStrengthId,contentTypeId
723592007 |MRCM attribute range type reference set|	referencedComponentId,rangeConstraint,attributeRule,ruleStrengthId,contentTypeId

723563008 |MRCM module scope type reference set|	referencedComponentId,mrcmRuleRefsetId

References

1. *HL7 Version 3 Implementation Guide: TermInfo – Using SNOMED CT in CDA R2 Models, Release 1*, HL7 5th DSTU Ballot, January 2014, http://wiki.hl7.org/index.php?title=File:V3_IG_SNOMED_R1_D5_2014JAN.docx
2. *SNOMED International APG Syntax Parsers*, IHTSDO, 2016, http://apg.ihtsdotools.org/
3. *NHS Logical Record Architecture for Health and Social Care*, UK Terminology Centre, November 2013, https://isd.hscic.gov.uk/trud3/user/guest/group/0/pack/12
4. *SNOMED CT Compositional Grammar – Specification and Guide*, IHTSDO, July 2015, http://snomed.org/compgrammar
5. SNOMED International Glossary, Draft version July 2014, http://snomed.org/gl
6. SNOMED CT Languages Github Repository, https://github.com/IHTSDO/SNOMEDCT-Languages
7. *SNOMED CT Starter Guide*, IHTSDO, February 2014, http://snomed.org/sg
8. *SNOMED CT Technical Implementation Guide*, IHTSDO, July 2014, http://snomed.org/tig