



Leading healthcare
terminology, worldwide

SNOMED CT Expression Constraint Language Specification and Guide

Version 1.5

Publication date: 2020-09-30

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)	15
5.3 Informative Comments	18
5.4 Operator Precedence	30
5.5 Character Collation for Term Filters.....	31
6. Examples	34
6.1 Simple Expression Constraints.....	34
6.2 Refinements.....	39
6.3 Cardinality	46
6.4 Conjunction and Disjunction	51
6.5 Exclusion and Not Equals	56
6.6 Constraint Comments.....	57
6.7 Nested Expression Constraints.....	58
6.8 Filter Constraints.....	62
7. Implementation Considerations	68
7.1 Authoring.....	68
7.2 Parsing	70
7.3 Validating.....	70
7.4 Executing	70
7.5 Storing	71
7.6 Displaying	71



7.7 Exchanging72

Appendix A – Examples Of Valid Expressions73

A.1 Simple Expression Constraints - Valid Expressions.....73

A.2 Refinements - Valid Expressions74

A.3 Cardinality - Valid Expressions77

A.4 Conjunction and Disjunction - Valid Expressions.....79

A.5 Exclusion and Not Equals - Valid Expressions81

A.6 Nested Expression Constraints - Valid Expressions.....82

Appendix B – Examples Of Invalid Expressions84

B.1 Simple Expression Constraints - Invalid Expressions84

B.2 Refinements - Invalid Expressions85

B.3 Cardinality - Invalid Expressions88

B.4 Conjunction and Disjunction - Invalid Expressions.....90

B.5 Exclusion and Not Equals - Invalid Expressions91

B.6 Nested Expression Constraints - Invalid Expressions93

Appendix C - Dialect Aliases.....95

References.....96



SNOMED
International

Leading healthcare
terminology, worldwide

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.

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

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

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

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

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

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

1. Introduction

Background

SNOMED CT is a clinical terminology with global scope covering a wide range of clinical specialties and requirements. The use of SNOMED CT expressions in Electronic Health Records (EHRs) provides a standardized way to represent clinical meanings captured by clinicians and enables the automatic interpretation of these meanings. SNOMED CT 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, which are either complete or under development 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.

New Features in this Version

This document defines and describes the current version of the Expression Constraint Language - ECL v1.5. This version of ECL has introduced the filter constraint. These constraints allow the result set to be filtered, by matching

only on concepts which have a description that satisfies the filter criteria. To support this significant new features, the following pages have been updated or added:

- [3.2 Expression Constraint and Query Requirements](#)
- [4. Logical Model](#)
- [5.1 Brief Syntax \(Normative\)](#)
- [5.2 Long Syntax \(Informative\)](#)
- [5.3 Informative Comments](#)
- [5.5 Character Collation for Term Filters - new](#)
- [6.8 Filter Constraints - new](#)
- [Appendix C - Dialect Aliases - new](#)

History

Expression constraints have been used in projects and programs around the world for a number of years – for example [HL7 Terminology](#), 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 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.

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. And finally, [Appendix C - Dialect Aliases](#) provides a list of example aliases that may be used to specify a particular dialect in an ECL filter constraint.

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 specifications](#).

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).
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.
Term filter	The ability to filter the result set, based on each concept having a matching description. Matching criteria for descriptions should be able to restrict the 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.

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 filters may be applied to an expression constraint, which can include term, language, type and dialect filters.

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.

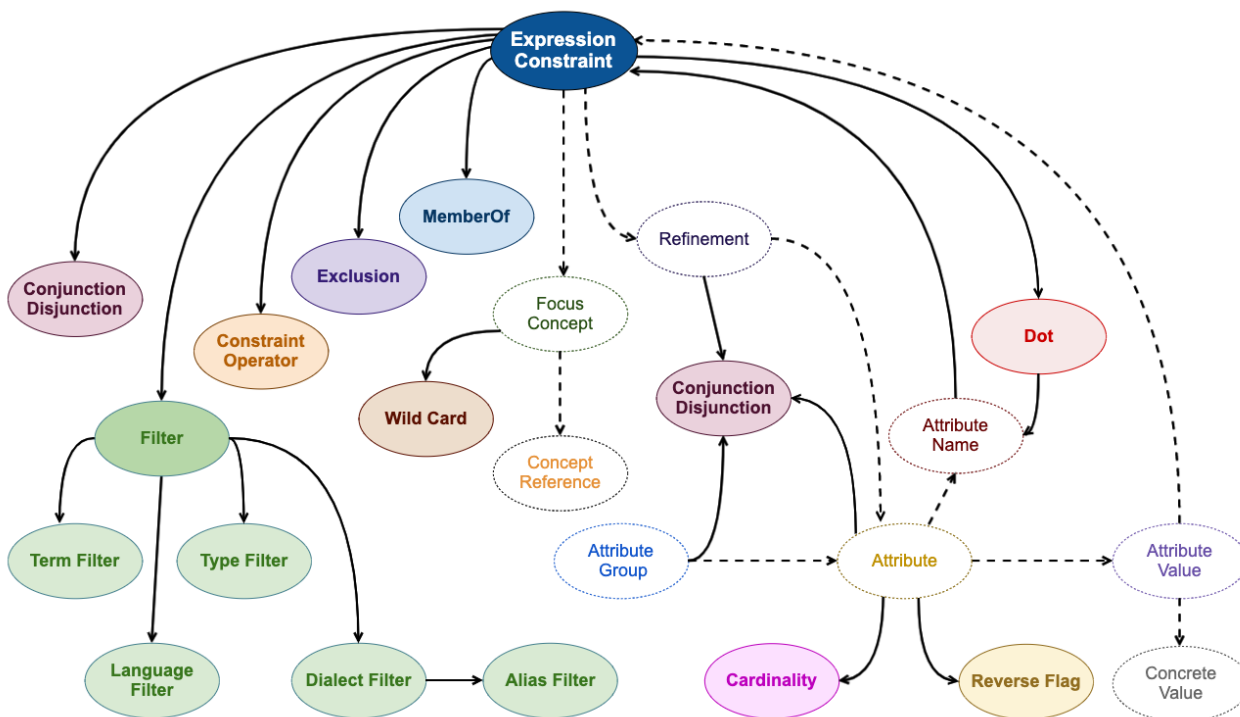


Figure 1: Abstract Model of a SNOMED CT Expression Constraint

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.

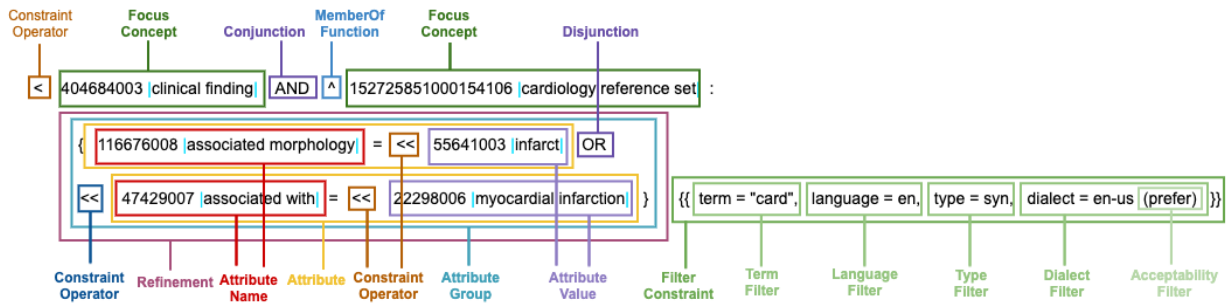


Figure 2: The main components of an example expression constraint

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.

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](#).

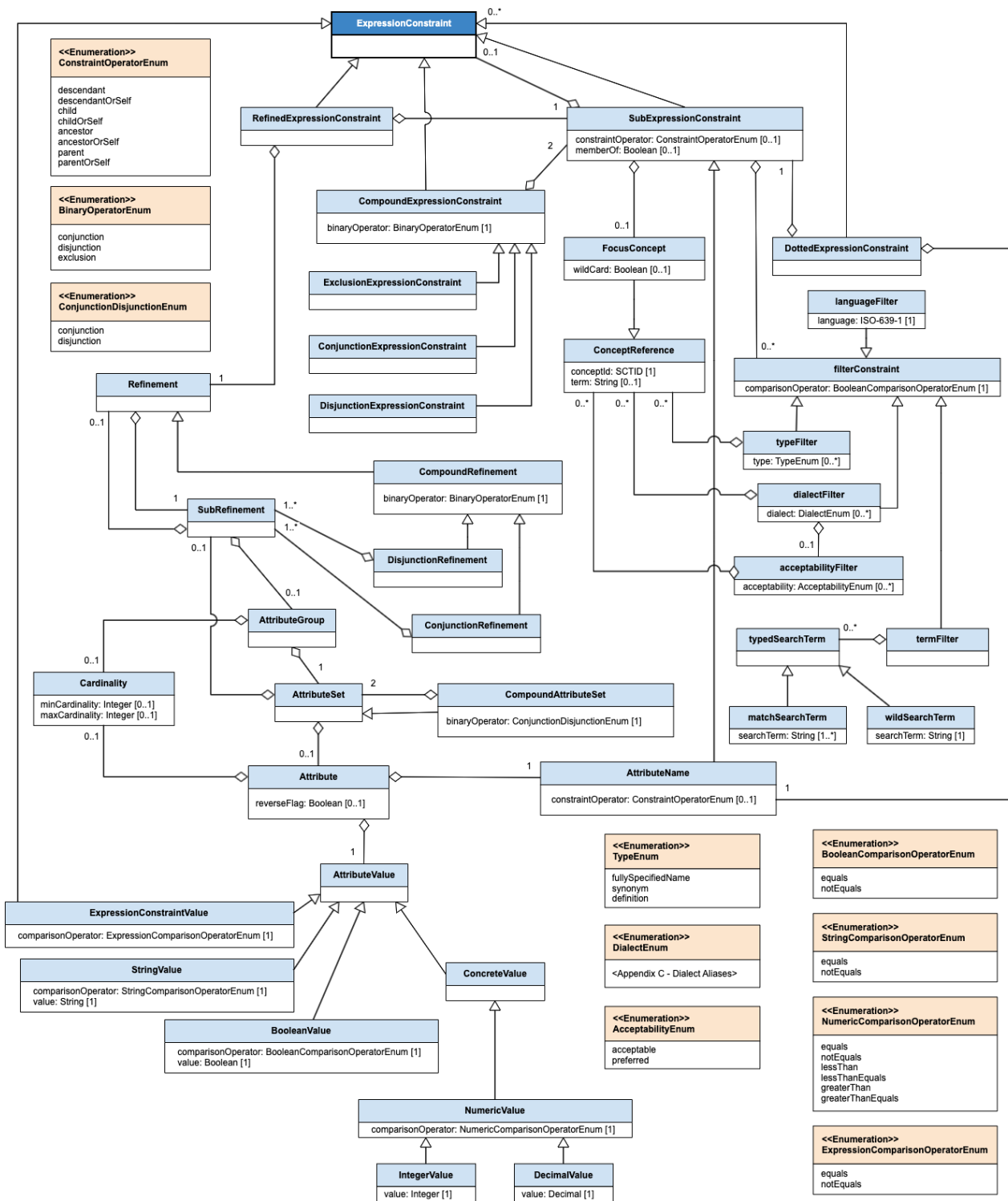


Figure 3: Logical Model of Expression Constraint Language

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 filterConstraint)
eclFocusConcept = eclConceptReference / wildCard
dot = "."
memberOf = "^"
eclConceptReference = conceptId [ws "|" ws term ws "|"]
eclConceptReferenceSet = "(" ws eclConceptReference *(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 QM
stringValue QM / booleanComparisonOperator ws booleanValue)
cardinality = minValue to maxValue
minValue = nonNegativeIntegerValue
to = ".."
maxValue = nonNegativeIntegerValue / many
many = "*"

reverseFlag = "R"
eclAttributeName = subExpressionConstraint
expressionComparisonOperator = "=" / "!="
numericComparisonOperator = "=" / "!=" / "<=" / "<" / ">=" / ">"
stringComparisonOperator = "=" / "!="
booleanComparisonOperator = "=" / "!="
filterConstraint = "{" ws filter *(ws " " ws filter) ws "
filter = termFilter / languageFilter / typeFilter / dialectFilter
termFilter = termKeyword ws booleanComparisonOperator 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 (eclConceptReference / 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 (eclConceptReference / 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 = acceptabilityIdSet / acceptabilityTokenSet
acceptabilityIdSet = eclConceptReferenceSet
acceptabilityTokenSet = "(" ws acceptabilityToken *(mws acceptabilityToken) ws ")"
acceptabilityToken = acceptable / preferred
acceptable = ("a"/"A") ("c"/"C") ("e"/"E") ("p"/"P") ("t"/"T")
preferred = ("p"/"P") ("r"/"R") ("e"/"E") ("f"/"F") ("e"/"E") ("r"/"R")
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 filterConstraint ws)

eclFocusConcept = eclConceptReference / wildcard

dot = "."

memberOf = "^" / ("m"/"M") ("e"/"E") ("m"/"M") ("b"/"B") ("e"/"E") ("r"/"R") ("o"/"O") ("f"/"F")

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

eclConceptReferenceSet = "(" ws eclConceptReference *(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 QM stringValue QM / 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 "=" / "<>" / "<=" / "<" / ">=" / ">"
stringComparisonOperator = "=" / "!=" / ("n"/"N") ("o"/"O") ("t"/"T") ws "=" / "<>"
booleanComparisonOperator = "=" / "!=" / ("n"/"N") ("o"/"O") ("t"/"T") ws "=" / "<>"
filterConstraint = "{ { ws filter *(ws " ws filter) ws } }"
filter = termFilter / languageFilter / typeFilter / dialectFilter
termFilter = termKeyword ws booleanComparisonOperator 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 (eclConceptReference / 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 (eclConceptReference / 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 = acceptabilityIdSet / acceptabilityTokenSet
acceptabilityIdSet = eclConceptReferenceSet
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")]
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 constraints 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 filterConstraint)	
	<p>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). A sub expression constraint may optionally include one or more filter constraints at the end.</p> <p>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 Operator Precedence. Therefore, if a constraintOperator is followed by a memberOf function, then the memberOf function is processed prior to the constraintOperator.</p>
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 = "^"	
LS: memberOf = "^" / ("m"/"M") ("e"/"E") ("m"/"M") ("b"/"B") ("e"/"E") ("r"/"R") ("o"/"O") ("f"/"F")	
	The 'memberOf' function returns the set of referenced components in the reference set whose concept identifier 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.
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 *(mws eclConceptReference) ws ")"	
	A concept reference set includes one 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 QM stringValue QM / 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 be equal to (or not equal to) the given string or 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: 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 = "{" ws filter *(ws " , " ws filter) ws "}"	
	A filterConstraint is a constraint used to filter the concepts in the result set, based on the given conditions.
BS/LS: filter = termFilter / languageFilter / typeFilter / dialectFilter	
	A filter is either a term filter, a language filter, a type filter or a dialect filter.
BS/LS: termFilter = termKeyword ws booleanComparisonOperator ws (typedSearchTerm / typedSearchTermSet)	
	A termFilter starts with the 'term' keyword, followed by a boolean 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 = typeldFilter / typeTokenFilter	
	A type filter specifies the description types that a matching description may have. A type filter is either a typeld filter or a typeToken filter.
BS/LS: typeldFilter = typeld ws booleanComparisonOperator ws (eclConceptReference / eclConceptReferenceSet)	
	A typeld filter starts with the 'typeld' keyword, followed by a boolean comparison operator, and either a single concept reference or a set of concept references.
BS/LS: typeld = ("t"/"T") ("y"/"Y") ("p"/"P") ("e"/"E") ("i"/"I") ("d"/"D")	
	The 'typeld' 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")	
BS: 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 (eclConceptReference / dialectIdSet)	
	A dialectId filter starts with the 'dialectId' keyword, followed by a boolean comparison operator, and either a single concept reference 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 = acceptabilityIdSet / 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 an acceptabilityId set or an acceptabilityToken set.
BS/LS: acceptabilityIdSet = eclConceptReferenceSet	
	An acceptabilityId set is a set of concept references.
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: 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: <ol style="list-style-type: none"> 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	

	<p>Digits 1 through 9, but excluding 0.</p> <p>The first character of a concept identifier is constrained to a digit other than zero.</p>
BS/LS: nonwsnonpipe = %x21-7B / %x7D-7E / UTF8-2 / UTF8-3 / UTF8-4	
	<p>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 "</p>
BS/LS: anyNonEscapedChar = SP / HTAB / CR / LF / %x20-21 / %x23-5B / %x5D-7E / UTF8-2 / UTF8-3 / UTF8-4	
	<p>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).</p>
BS/LS: escapedChar = BS QM / BS BS	
	<p>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.</p>
BS/LS: escapedWildChar = BS QM / BS BS / BS star	
	<p>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 *.</p>
BS/LS: nonwsNonEscapedChar = %x21 / %x23-5B / %x5D-7E / UTF8-2 / UTF8-3 / UTF8-4	
	<p>A nonwsNonEscapedChar is any printable ASCII, UTF8-2, UTF8-3 or UTF8-4 character, excluding double quotes ("), backslash (\), and space ().</p>
BS/LS: alpha = %x41-5A / %x61-7A	
	<p>An alpha is any uppercase or lowercase character from "A" to "Z" (and "a" to "z") inclusive.</p>
BS/LS: dash = %x2D	
	<p>A dash is a hyphen (i.e. "-").</p>
BS/LS: UTF8-2 = %xC2-DF UTF8-tail	
	<p>UTF8 characters encoded as 2-octet sequences.</p>
BS/LS: UTF8-3 = %xE0 %xA0-BF UTF8-tail / %xE1-EC 2(UTF8-tail) / %xED %x80-9F UTF8-tail / %xEE-EF 2(UTF8-tail)	
	<p>UTF8 characters encoded as 3-octet sequences.</p>
BS/LS: UTF8-4 = %xF0 %x90-BF 2(UTF8-tail) / %xF1-F3 3(UTF8-tail) / %xF4 %x80-8F 2(UTF8-tail)	
	<p>UTF8 characters encoded as 4-octet sequences.</p>
BS/LS: UTF8-tail = %x80-BF	
	<p>UTF8 characters encoded as 8-octet sequences.</p>

5.4 Operator Precedence

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.

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, rèsùmè, Résumé, RÉSUMÉ, ...	-
Resume	resume, Resume, RESUME, résumé, rèsùmè, 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, ...	ångstræm, Ångstræm, ÅNGSTRÆM, ...
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, ångstræm, Ångstræm, ÅNGSTRÆM, ...

- 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, ångstræm, Ångstræm, ÅNGSTRÆM, aangström, Aangstrøm, AANGSTRØM, ...
Ångström	ångström, Ångström, ÅNGSTRÖM, ångstrøm, Ångstrøm, ÅNGSTRØM, ångstræm, Ångstræm, ÅNGSTRÆM, ...	angstrom, Angstrom, ANGSTROM, aangström, Aangstrøm, AANGSTRØM, ...

Search Term	Target Matches	Target does NOT Match
Ångström	ångström, Ångström, ÅNGSTRÖM, ...	angstrom, Angstrom, ANGSTROM, ångström, Ångström, ÅNGSTRÖM, ångstrøem, Ångstrøem, ÅNGSTRØEM, ...
aangstrøm	aangstrøm, Aangstrøm, AANGSTRØM, ...	angstrom, Angstrom, ANGSTROM, ångström, Ångström, ÅNGSTRÖM, ångstrøm, Ångstrøm, ÅNGSTRØM, ångstrøem, Ångstrøem, ÅNGSTRØEM, ...

- 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, ångstrøem, Ångstrøem, ÅNGSTRØEM, aangstrøm, Aangstrøm, AANGSTRØM ...
Ångström	ångström, Ångström, ÅNGSTRÖM, aangström, Aangstrøm, AANGSTRØM, ...	angstrom, Angstrom, ANGSTROM, ångstrøm, Ångstrøm, ÅNGSTRØM, ångstrøem, Ångstrøem, ÅNGSTRØEM, ...
Ångstrøm	ångstrøm, Ångstrøm, ÅNGSTRØM, ångström, Ångström, ÅNGSTRÖM, aangstrøm, Aangstrøm, AANGSTRØM, ...	angstrom, Angstrom, ANGSTROM, ångstrøem, Ångstrøem, ÅNGSTRØEM, ...
aangstrøm	ångstrøm, Ångstrøm, ÅNGSTRØM, ångström, Ångström, ÅNGSTRÖM, aangstrøm, Aangstrøm, AANGSTRØM, ...	angstrom, Angstrom, ANGSTROM, ångstrøem, Ångstrøem, ÅNGSTRØEM, ...

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 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 `referencedComponents` 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|
```

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](#).

Any

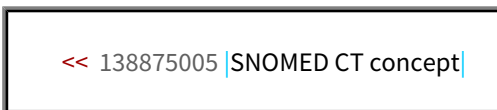
A single 'star' (i.e. "*") may be used in the place of a concept reference to represent any concept in the substrate. The expression constraint below evaluates to the set of all concepts in the given substrate.



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



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:



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:

```
^*
```

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 only by amoxicillin tablets, whose strength is greater than or equal to 200 mg .

```

< 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| )
  }
  
```

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 capsules that have a strength between 500 and 800 mg (inclusive), the following expression constraint may be used:

```

< 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| ) }
  
```

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 trade name equal to "PANADOL" ².

```

< 373873005 |Pharmaceutical / biologic product| :
  209999999104 |Has trade name| = "PANADOL"
  
```

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

```

< 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 represent 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
```

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

- 2 Concrete values of type string are case sensitive and compared using the Unicode Collation Algorithm (<http://www.unicode.org/reports/tr10/>).
- 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 view¹. 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-redundant² 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-redundant² 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. \geq) 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. \leq) 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 Filter Constraints

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

Overview

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 constraint will be included in the set of matching concepts. Descriptions can be filtered based on their term, type, language, and acceptability in a given dialect.

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. * `{{ 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"}}
```

Please note that filters apply only to the subexpression 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 subexpression, 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" }}
```

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 "typeld" 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 'typeld' keyword with the corresponding concept reference.

Type Keyword		Typeld
Brief Syntax	Full Syntax	Concept Reference
fsn	fullySpecifiedName	90000000000003001 Fully specified name
syn	synonym	90000000000013009 Synonym
def	definition	90000000000055004 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, typeld = 90000000000013009 |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 `< 9000000000000511003 |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 Concept Reference
Brief Syntax	Full Syntax	
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 = 90000000000013009 |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 preceding 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) }}
```

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.

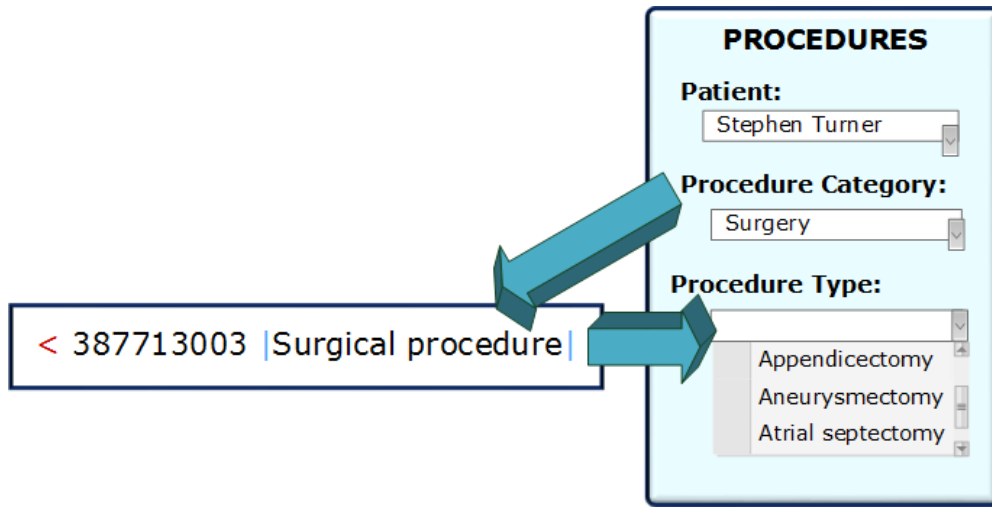


Figure 4: Authoring using expression constraint templates

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.

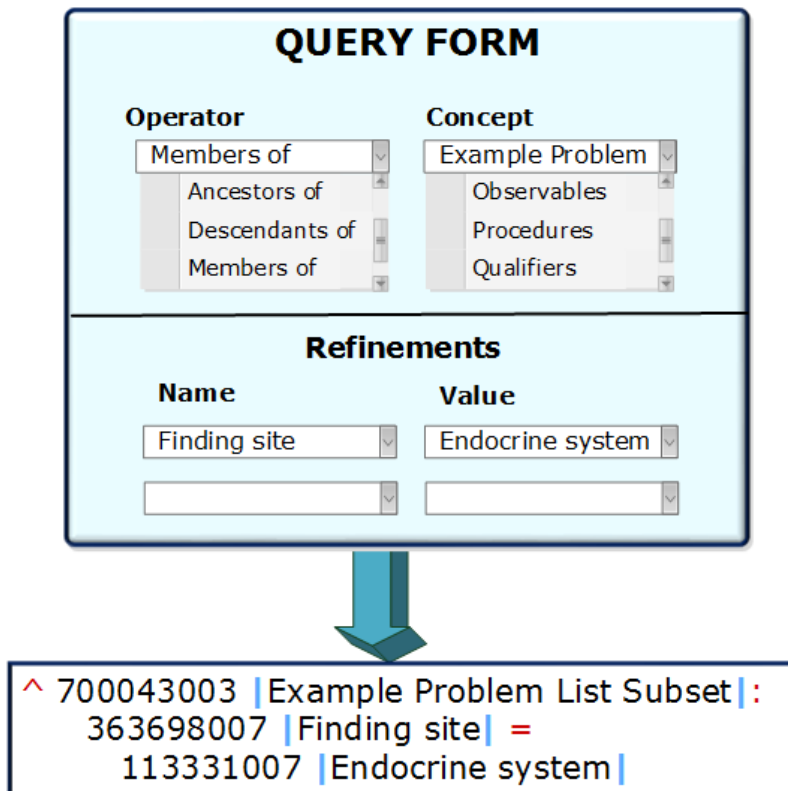


Figure 5: Authoring using a form-driven query tool

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
- abnfggen

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++.

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 9000000000000455006 |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 276637009 Hemorrhagic pulmonary edema	210051003 Injury to heart and lung : 116676008 Associated morphology = 79654002 Edema
< 19829001 Disorder of lung : 116676008 Associated morphology = << 79654002 Edema	233709006 Toxic pulmonary edema 233711002 Oxygen-induced pulmonary edema	275504005 Lung cyst : 116676008 Associated morphology = 103619005 Inflammatory edema 19829001 Disorder of lung : 116676008 Associated morphology = 40829002 Acute edema

<p>< 404684003 Clinical finding : 363698007 Finding site = << 39057004 Pulmonary valve structure , 116676008 Associated morphology = << 415582006 Stenosis </p>	<p>56786000 Pulmonic valve stenosis </p>	<p>56786000 Pulmonic valve stenosis : 363698007 Finding site = 90318009 Structure of anulus fibrosus of pulmonary artery , 116676008 Associated morphology = 88015002 Partial stenosis </p>
	<p>86299006 Tetralogy of Fallot </p>	<p>404684003 Clinical finding : 363698007 Finding site = 39057004 Pulmonary valve structure , 116676008 Associated morphology = 415582006 Stenosis </p>
<p>* : 246075003 Causative agent = 387517004 Paracetamol </p>	<p>295124009 Paracetamol overdose </p> <p>292042007 Adverse reaction to paracetamol </p>	<p>404684003 Clinical finding : 246075003 Causative agent = 387517004 Paracetamol </p>
<p>< 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 }</p>	<p>86299006 Tetralogy of Fallot </p> <p>204351007 Fallot's trilogy </p>	<p>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 }</p>
<p><< 404684003 Clinical finding : << 47429007 Associated with = << 267038008 Edema </p>	<p>230580009 Myxedema neuropathy </p>	<p>95356008 Mucosal ulcer : 42752001 Due to = 19242006 Pulmonary edema </p>
<p>< 27658006 Amoxicillin : 411116001 Has dose form = << 385055001 Tablet dose form , { 17999999100 Has basis of strength = (21999999102 Amoxicillin only : 18999999103 Has strength magnitude >= #200, 19999999101 Has strength unit = 258684004 mg) } }}</p>	<p>374644001 Amoxicillin trihydrate 200 mg tablet </p>	<p>27658006 Amoxicillin : 411116001 Has dose form = 421026006 Oral tablet , { 127489000 Has active ingredient = 96068000 Amoxicillin trihydrate , 17999999100 Has basis of strength = (21999999102 Amoxicillin only : 18999999103 Has strength magnitude = #500, 19999999101 Has strength unit = 258684004 mg) }</p>

<pre>< 27658006 Amoxicillin : 411116001 Has dose form = << 385055001 Tablet dose form , { 17999999100 Has basis of strength = (21999999102 Amoxicillin only : 18999999103 Has strength magnitude >= #500, 18999999103 Has strength magnitude <= #800, 19999999101 Has strength unit = 258684004 mg) }</pre>	<pre>374646004 Amoxicillin 500 mg tablet </pre>	<pre>27658006 Amoxicillin : 411116001 Has dose form = 421026006 Oral tablet , { 17999999100 Has basis of strength = (21999999102 Amoxicillin only : 18999999103 Has strength magnitude = #750, 19999999101 Has strength unit = 258684004 mg) }</pre>
<pre>< 373873005 Pharmaceutical / biologic product : 20999999104 Has trade name = "PANADOL"</pre>	<pre>25999999103 PANADOL [paracetamol] tablet </pre>	<pre>373873005 Pharmaceutical / biologic product : { 127489000 Has active ingredient = 412031009 Paracetamol or derivative }, 20999999104 Has trade name = "PANADOL"</pre>
<pre>< 91723000 Anatomical structure : R 363698007 Finding site = < 125605004 Fracture of bone </pre>	<pre>85050009 Humerus </pre>	<pre>85050009 Humerus : 272741003 Laterality = 7771000 Left </pre>
	<pre>71341001 Femur </pre>	<pre>71341001 Femur : 272741003 Laterality = 24028007 Right </pre>
<pre>< 125605004 Fracture of bone . 363698007 Finding site </pre>	<pre>85050009 Humerus </pre>	<pre>85050009 Humerus : 272741003 Laterality = 7771000 Left </pre>
	<pre>71341001 Femur </pre>	<pre>71341001 Femur : 272741003 Laterality = 24028007 Right </pre>
<pre>< 105590001 Substance : R << 127489000 Has active ingredient = < 27658006 Product containing amoxicillin </pre>	<pre>395938000 Clavulanate potassium 387137007 Omeprazole </pre>	<pre>-</pre>
<pre>< 27658006 Product containing amoxicillin . << 127489000 Has active ingredient </pre>	<pre>395938000 Clavulanate potassium 387137007 Omeprazole </pre>	<pre>-</pre>
<pre>< 404684003 Clinical finding : * = 79654002 Edema </pre>	<pre>19242006 Pulmonary edema 97341000119105 Prolifera tive retinopathy with retin al edema due to type</pre>	<pre>404684003 Clinical finding : 116676008 Associated morphology = 79654002 Edema </pre>

< 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	27999999108 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	

<p>< 404684003 Clinical finding : [1..1] 363698007 Finding site = < 91723000 Anatomical structure </p>	<p>125596004 Injury of elbow </p>	<p>404684003 Clinical finding : { 116676008 Associated morphology = 72704001 Fracture , 363698007 Finding site = 299701004 Bone of forearm , 363698007 Finding site = 62413002 Bone structure of radius }²</p>
<p>< 404684003 Clinical finding : [2..*] 363698007 Finding site = < 91723000 Anatomical structure </p>	<p>86299006 Tetralogy of Fallot </p>	<p>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 }</p>
<p>< 404684003 Clinical finding : { [2..*] 363698007 finding site = < 91723000 Anatomical structure }</p>	<p>-</p>	<p>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 }</p>
<p>< 373873005 Pharmaceutical / biologic product : [1..3] { [1..*] 127489000 Has active ingredient = < 105590001 Substance }</p>	<p>322236009 Paracetamol 500mg tablet </p>	<p>373873005 Pharmaceutical / biologic product : { 127489000 Has active ingredient = 412031009 Paracetamol or derivative }</p>
	<p>404826002 Benzocaine + butamben + tetracaine hydrochloride </p>	<p>373873005 Pharmaceutical / biologic product : { 127489000 Has active ingredient = 412031009 Paracetamol or derivative }, { 127489000 Has active ingredient = 387494007 Codeine }</p>
<p>< 373873005 Pharmaceutical / biologic product : [0..1] { 127489000 Has active ingredient = < 105590001 Substance }</p>	<p>11111527999999108 Inert tablet </p>	<p>373873005 Pharmaceutical / biologic product : { 127489000 Has active ingredient = 412031009 Paracetamol or derivative }</p>
	<p>370166004 Aspirin 325mg tablet </p>	
<p>< 373873005 Pharmaceutical / biologic product : [1..*] { 127489000 Has active ingredient = < 105590001 Substance }</p>	<p>370166004 Aspirin 325mg tablet </p>	<p>373873005 Pharmaceutical / biologic product : { 127489000 Has active ingredient = 412031009 Paracetamol or derivative }, { 127489000 Has active ingredient = 387494007 Codeine }</p>
<p>< 404684003 Clinical finding : [1..1] { 363698007 Finding site = < 91723000 Anatomical structure }</p>	<p>125596004 Injury of elbow </p>	<p>404684003 Clinical finding : { 363698007 Finding site = 299701004 Bone of forearm }, { 363698007 Finding site = 62413002 Bone structure of radius }</p>


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

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

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


 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 wound }
	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	



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

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

-	-
-	-

- 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	

<pre>*: 246075003 Causative agent = 387517004 Paracetamol </pre>	<pre>46093004 Paracetamol measurement </pre>	<pre>404684003 Clinical finding : 246075003 Causative agent = 372687004 Amoxicillin </pre>
<pre>< 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 }</pre>	<pre>404684003 Clinical finding 56786000 Pulmonary valve stenosis </pre>	<pre>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 }</pre>
<pre><< 404684003 Clinical finding : << 47429007 Associated with = << 267038008 Edema </pre>	<pre>404684003 Clinical finding </pre>	<pre>95356008 Mucosal ulcer : 42752001 Due to = 59901004 Cheek biting </pre>
<pre>< 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) }</pre>	<pre>269999999100 Amoxicillin capsule 374233002 Amoxicillin trihydrate 125 mg chewable tablet </pre>	<pre>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) }</pre>

<pre>< 27658006 Amoxicillin : 411116001 Has dose form =<< 385055001 Tablet dose form , { 17999999100 Has basis of strength = (21999999102 Amoxicillin only : 18999999103 Has strength magnitude >= #500, 18999999103 Has strength magnitude <= #800, 19999999101 Has strength unit = 258684004 mg) }</pre>	<pre>26999999100 Amoxicillin capsule 374647008 Amoxicillin 875 mg tablet </pre>	<pre>27658006 Amoxicillin : 411116001 Has dose form = 421026006 Oral tablet , { 17999999100 Has basis of strength = (21999999102 Amoxicillin only : 18999999103 Has strength magnitude = #850, 19999999101 Has strength unit = 258684004 mg) }</pre>
<pre>< 373873005 Pharmaceutical / biologic product : 20999999104 Has trade name = "PANADOL"</pre>	<pre>373873005 Pharmaceutical / biologic product 322236009 Paracetamol 500mg tablet </pre>	<pre>373873005 Pharmaceutical / biologic product : { 127489000 Has active ingredient = 412031009 Paracetamol or derivative , 20999999104 Has trade name = "PANADEINE" }</pre>
<pre>< 91723000 Anatomical structure : R 363698007 Finding site =< 125605004 Fracture of bone </pre>	<pre>34080009 Malleus structure 10200004 Liver structure </pre>	<pre>34080009 Malleus structure : 272741003 Laterality = 7771000 Left 10200004 Liver structure : 272741003 Laterality = 24028007 Right </pre>
<pre>< 125605004 Fracture of bone . 363698007 Finding site </pre>	<pre>34080009 Malleus structure 10200004 Liver structure </pre>	<pre>34080009 Malleus structure : 272741003 Laterality = 7771000 Left 10200004 Liver structure : 272741003 Laterality = 24028007 Right </pre>
<pre>< 105590001 Substance : R << 127489000 Has active ingredient = < 27658006 Product containing amoxicillin </pre>	<pre>105590001 Substance </pre>	<pre>373873005 Pharmaceutical / biologic product : 127489000 Has active ingredient = 372687004 Amoxicillin </pre>

	387517004 Paracetamol	
24999999101 TRIPHASIL tablet . 127489000 Has active ingredient	105590001 Substance 387517004 Paracetamol	373873005 Pharmaceutical / biologic product : 127489000 Has active ingredient = 126109000 Levonorgestrel
< 404684003 Clinical finding : * = 79654002 Edema	263225007 Hip fracture 385933006 Edema control education	404684003 Clinical finding : 116676008 Associated morphology = 72704001 Fracture
< 404684003 Clinical finding : 116676008 Associated morphology = *	195967001 Asthma 73211009 Diabetes mellitus	404684003 Clinical finding : 363698007 Finding site = 80891009 Heart structure 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 '999999' 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	27999999108 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 : [1..1] 127489000 Has active ingredient = < 105590001 Substance	27999999108 Inert tablet 412556009 Paracetamol + codeine	373873005 Pharmaceutical / biologic product : { 127489000 Has active ingredient = 412031009 Paracetamol or derivative }, { 127489000 Has active ingredient = 387494007 Codeine }

<p>< 373873005 Pharmaceutical / biologic product :</p> <p>[0..1] 127489000 Has active ingredient =</p> <p>< 105590001 Substance </p>	412556009 Paracetamol + codeine	373873005 Pharmaceutical / biologic product :
		{ 127489000 Has active ingredient = 412031009 Paracetamol or derivative },
		{ 127489000 Has active ingredient = 387494007 Codeine }
<p>< 373873005 Pharmaceutical / biologic product :</p> <p>[1..*] 127489000 Has active ingredient =</p> <p>< 105590001 Substance </p>	27999999108 Inert tablet	373873005 Pharmaceutical / biologic product :
		411116001 Has dose form =
		385055001 Tablet
<p>< 404684003 Clinical finding :</p> <p>[1..1] 363698007 Finding site =</p> <p>< 91723000 Anatomical structure </p>	75857000 Fracture of radius and ulna	404684003 Clinical finding :
	40733004 Infectious disease	{ 116676008 Associated morphology = 72704001 Fracture ,
		363698007 Finding site = 62413002 Bone structure of radius ,
		363698007 Finding site = 23416004 Bone structure of ulna }
<p>< 404684003 Clinical finding :</p> <p>[2..*] 363698007 Finding site =</p> <p>< 91723000 Anatomical structure </p>	23406007 Arm fracture	404684003 Clinical finding :
	40733004 Infectious disease	{ 116676008 Associated morphology = 72704001 Fracture ,
		363698007 Finding site = 702468001 Bone structure of lower leg }
<p>< 404684003 Clinical finding :</p> <p>{ [2..*] 363698007 Finding site =</p> <p>< 91723000 Anatomical structure }</p>	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 }
<p>< 373873005 Pharmaceutical / biologic product :</p> <p>[1..3] { [1..*] 127489000 Has active ingredient =</p> <p>< 105590001 Substance }</p>	27999999108 Inert tablet	373873005 Pharmaceutical / biologic product :
	437867004 Chlorphenamine + dextromethorphan + paracetamol + pseudoephedrine	{ 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 }
<p>< 373873005 Pharmaceutical / biologic product :</p> <p>[0..1] { 127489000 Has active ingredient =</p> <p>< 105590001 Substance }</p>	412556009 Paracetamol + codeine	373873005 Pharmaceutical / biologic product :
		{ 127489000 Has active ingredient = 412031009 Paracetamol or derivative },
		{ 127489000 Has active ingredient = 387494007 Codeine }
<p>< 373873005 Pharmaceutical / biologic product :</p> <p>[1..*] { 127489000 Has active ingredient =</p> <p>< 105590001 Substance }</p>	27999999108 Inert tablet	373873005 Pharmaceutical / biologic product :
		411116001 Has dose form =
		385055001 Tablet


<p>< 404684003 Clinical finding : [1..1] { 363698007 Finding site = < 91723000 Anatomical structure }</p>	<p>75857000 Fracture of radius and ulna 40733004 Infectious disease </p>	<p>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 }</p>
<p>< 404684003 Clinical finding : [0..0] { [2..*] 363698007 Finding site = < 91723000 Anatomical structure }</p>	<p>-</p>	<p>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 }</p>

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

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


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

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

B.6 Nested Expression Constraints - Invalid Expressions

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

 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 Filter Constraints](#) 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	90000000000508004 Great Britain English language reference set
en-ie	21000220103 Irish language reference set
en-nz	271000210107 New Zealand English language reference set
en-us	90000000000509007 United States of America English language reference set
en-int-gmdn	608771002 GMDN language reference set
en-nhs-clinical	99900126100000100 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	99900125100000103 United Kingdom Extension Great Britain English language reference set
es	448879004 Spanish language reference set
es-ar	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
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
zh	722128001 Chinese language reference set

References

1. *HL7 Version 3 Implementation Guide: Terminology – 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>