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") ("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