5.1. Normative Specification

The following ABNF definition specifies the SNOMED CT template syntax (v1.1.1). This syntax incorporates the Expression Constraint Language version 1.4 (ECL v1.4), with some adaptations to support slot references. It also incorporates 3 rules from Compositional Grammar (SCG v2.4) to ensure that the definitionStatus rule is defined.

; Template Syntax v1.1

templateSlot = templateReplacementSlot / templateInformationSlot

templateReplacementSlot = conceptReplacementSlot / expressionReplacementSlot / tokenReplacementSlot / concreteValueReplacementSlot

conceptReplacementSlot = "[[" ws "+" ws conceptReplacement [slotName ws] "]]"

expressionReplacementSlot = "[[" ws "+" ws expressionReplacement [slotName ws] "]]"

tokenReplacementSlot = "[[" ws "+" ws tokenReplacement [slotName ws] "]]"

concreteValueReplacementSlot = "[[" ws "+" ws concreteValueReplacement [slotName ws] "]]"

conceptReplacement = "id" ws ["(" ws slotExpressionConstraint ws ")" ws]

expressionReplacement = ["scg" ws] ["(" ws slotExpressionConstraint ws ")" ws]

tokenReplacement = "tok" ws ["(" ws slotTokenSet ws ")" ws]

concreteValueReplacement = stringReplacement / integerReplacement / decimalReplacement / booleanReplacement

stringReplacement = "str" ws ["(" ws slotStringSet ws ")" ws]

integerReplacement = "int" ws ["(" ws slotIntegerSet ws ")" ws]

decimalReplacement = "dec" ws ["(" ws slotDecimalSet ws ")" ws]

booleanReplacement = "bool" ws ["(" ws slotBooleanSet ws ")" ws]

slotTokenSet = slotToken *(mws slotToken)

slotStringSet = slotStringValue *(mws slotStringValue)

slotStringValue = QM stringValue QM

slotIntegerSet = (slotIntegerValue / slotIntegerRange) *(mws (slotIntegerValue / slotIntegerRange))

slotDecimalSet = (slotDecimalValue / slotDecimalRange) *(mws (slotDecimalValue / slotDecimalRange))

slotBooleanSet = slotBooleanValue *(mws slotBooleanValue)

slotBooleanValue = booleanValue

slotIntegerRange = (slotIntegerMinimum to [slotIntegerMaximum]) / (to slotIntegerMaximum)

slotIntegerMinimum = [exclusiveMinimum] slotIntegerValue

slotIntegerMaximum = [exclusiveMaximum] slotIntegerValue

slotIntegerValue = "#" ["-"/"+"] integerValue

slotDecimalRange = (slotDecimalMinimum to [slotDecimalMaximum]) / (to slotDecimalMaximum)

slotDecimalMinimum = [exclusiveMinimum] slotDecimalValue

slotDecimalMaximum = [exclusiveMaximum] slotDecimalValue

slotDecimalValue = "#" ["-"/"+"] decimalValue

exclusiveMinimum = ">"

exclusiveMaximum = "<"

slotName = "@" (slotStringValue / nonQuoteStringValue)

slotToken = definitionStatus / memberOf / constraintOperator / conjunction / disjunction / exclusion / reverseFlag / expressionComparisonOperator / numericComparisonOperator / stringComparisonOperator /booleanComparisonOperator

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

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

slotInformation = [cardinality ws] [slotName ws]

; Expression Constraint Language v1.4

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

slotRefinedExpressionConstraint = slotSubExpressionConstraint ws ":" ws slotEclRefinement

slotCompoundExpressionConstraint = slotConjunctionExpressionConstraint / slotDisjunctionExpressionConstraint / slotExclusionExpressionConstraint

slotConjunctionExpressionConstraint = slotSubExpressionConstraint 1*(ws conjunction ws slotSubExpressionConstraint)

slotDisjunctionExpressionConstraint = slotSubExpressionConstraint 1*(ws disjunction ws slotSubExpressionConstraint)

slotExclusionExpressionConstraint = slotSubExpressionConstraint ws exclusion ws slotSubExpressionConstraint

slotDottedExpressionConstraint = slotSubExpressionConstraint 1*(ws slotDottedExpressionAttribute)

slotDottedExpressionAttribute = dot ws slotEclAttributeName

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

slotEclFocusConcept = slotEclConceptReference / wildCard

dot = "."

memberOf = "^"

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

conceptId = sctId

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

wildcard = "*"

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

descendantOf = "<"

descendantOrSelfOf = "<<"

childOf = "<!"

ancestorOf = ">"

ancestorOrSelfOf = ">>"

parentOf = ">!"

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

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

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

slotEclRefinement = slotSubRefinement ws [slotConjunctionRefinementSet / slotDisjunctionRefinementSet]

slotConjunctionRefinementSet = 1*(ws conjunction ws slotSubRefinement)

slotDisjunctionRefinementSet = 1*(ws disjunction ws slotSubRefinement)

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

slotEclAttributeSet = slotSubAttributeSet ws [slotConjunctionAttributeSet / slotDisjunctionAttributeSet]

slotConjunctionAttributeSet = 1*(ws conjunction ws slotSubAttributeSet)

slotDisjunctionAttributeSet = 1*(ws disjunction ws slotSubAttributeSet)

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

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

slotEclAttribute = ["[" cardinality "]" ws] [reverseFlag ws] slotEclAttributeName ws (expressionComparisonOperator ws slotSubExpressionConstraint / numericComparisonOperator ws (slotIntegerValue /slotDecimalValue) / stringComparisonOperator ws slotStringValue / booleanComparisonOperator ws slotBooleanValue)

cardinality = minValue to maxValue

minValue = nonNegativeIntegerValue

to = ".."

maxValue = nonNegativeIntegerValue / many

many = "*"

reverseFlag = "R"

slotEclAttributeName = slotSubExpressionConstraint

expressionComparisonOperator = "=" / "!="

numericComparisonOperator = "=" / "!=" / "<=" / "<" / ">=" / ">"

stringComparisonOperator = "=" / "!="

booleanComparisonOperator = "=" / "!="

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

digit = %x30-39

zero = %x30

digitNonZero = %x31-39

nonwsNonPipe = %x21-7B / %x7D-7E / UTF8-2 / UTF8-3 / UTF8-4

anyNonEscapedChar = SP / HTAB / CR / LF / %x20-21 / %x23-5B / %x5D-7E / UTF8-2 / UTF8-3 / UTF8-4

escapedChar = BS QM / BS BS

UTF8-2 = %xC2-DF UTF8-tail

UTF8-3 = %xE0 %xA0-BF UTF8-tail / %xE1-EC 2(UTF8-tail) / %xED %x80-9F UTF8-tail / %xEE-EF 2(UTF8-tail)

UTF8-4 = %xF0 %x90-BF 2(UTF8-tail) / %xF1-F3 3(UTF8-tail) / %xF4 %x80-8F 2(UTF8-tail)

UTF8-tail = %x80-BF

; Additional rules from Compositional Grammar v2.4

definitionStatus = equivalentTo / subtypeOf

equivalentTo = "==="

subtypeOf = "<<<"