OCL Grammar

This section describes the grammar for OCL expressions.

The grammar description uses the EBNF syntax, where “|” means a choice, “?” optionality, and “*” means zero or more times, + means one or more times. In the description of the *name, number* and *string,* the syntax for lexical tokens from the JavaCC parser generator is used. (See http://www.metamata.com.)

```
oclFile                := ( "package" packageName
                             oclExpressions
                             "endpackage"
                           )+
packageName            := pathName
oclExpressions         := ( constraint )*
constraint             := contextDeclaration
                           ( stereotype name? ":"
                             oclExpression
                           )+
contextDeclaration     := "context"
                           ( operationContext | classifierContext )
classifierContext      := (  name ":" name )
                           | name
operationContext       := name "::" operationName
                           "(" formalParameterList ")"
                           ( ":" returnType )?
stereotype             := ( "pre" | "post" | "inv" )
operationName          := name | "=" | "+" | "-" | "<"  | "<=" |
                           ">="   | ">" | "/" | "*" | "<>" |
                           "implies" | "not" | "or" | "xor" | "and"
formalParameterList    := ( name ":" typeSpecifier
                             ("," name ":" typeSpecifier )*
                           )?
typeSpecifier          := simpleTypeSpecifier
                           | collectionType
collectionType         := collectionKind
                           "(" simpleTypeSpecifier ")"
oclExpression          := ( letExpression )* expression
returnType             := typeSpecifier
expression             := logicalExpression
letExpression          := "let" name
                           ( "(" formalParameterList ")" )?
                           ( ":" typeSpecifier )?
```

```
                                 "=" expression ";"
ifExpression                 := "if" expression
                                "then" expression
                                "else" expression
                                "endif"
logicalExpression            := relationalExpression
                                ( logicalOperator
                                  relationalExpression
                                )*
relationalExpression         := additiveExpression
                                ( relationalOperator
                                  additiveExpression
                                )?
additiveExpression           := multiplicativeExpression
                                ( addOperator
                                  multiplicativeExpression
                                )*
multiplicativeExpression:= unaryExpression
                                 ( multiplyOperator
                                   unaryExpression
                                 )*
unaryExpression              := ( unaryOperator
                                  postfixExpression
                                )
                                | postfixExpression
postfixExpression            := primaryExpression
                                ( ("." | "->")propertyCall )*
primaryExpression            := literalCollection
                                | literal
                                | propertyCall
                                | "("  expression  ")"
                                | ifExpression
propertyCallParameters := "(" ( declarator )?
                                ( actualParameterList )? ")"
literal                      := string
                                | number
                                | enumLiteral
enumLiteral                  := name "::" name ( "::" name )*
simpleTypeSpecifier          := pathName
literalCollection            := collectionKind "{"
                                ( collectionItem
                         ( "," collectionItem )*
                       )?
                       "}"
collectionItem        := expression (".." expression )?
propertyCall          := pathName
                       ( timeExpression )?
                       ( qualifiers )?
                       ( propertyCallParameters )?
qualifiers            := "[" actualParameterList "]"
declarator            := name ( "," name )*
                       ( ":" simpleTypeSpecifier )?
                       ( ";" name ":" typeSpecifier "="
                         expression
                       )?
                       "|"
pathName              := name ( "::" name )*
timeExpression        := "@" "pre"
actualParameterList   := expression ("," expression)*
logicalOperator       := "and" | "or" | "xor" | "implies"
collectionKind        := "Set" | "Bag" | "Sequence" | "Collection"
relationalOperator    := "=" | ">" | "<" | ">=" | "<=" | "<>"
addOperator           := "+" |  "-"
multiplyOperator      := "*" | "/"
unaryOperator         := "-" | "not"
name                  := ["a"-"z", "A"-"Z", "_"]
                       ( ["a"-"z", "A"-"Z", "0"-"9", "_" ] )*
number                := ["0"-"9"] (["0"-"9"])*
                       ( "." ["0"-"9"] (["0"-"9"])* )?
                       ( ("e" | "E") ( "+" | "-" )? ["0"-"9"]
                         (["0"-"9"])*
                       )?
string                := "'"
                       (( ~["′","\\","\n","\r"] )
                        |("\\"
                           ( ["n","t","b","r","f","\\","′","\""]
                             | ["0"-"7"]
                               ( ["0"-"7"] ( ["0"-"7"] )? )?
                           )
                         )
                       )*
                       "'"
```