```
// a test
/* a b c */
@namespace(uri="http://www.eclipse.org/ocl/1.1.0/OCL", prefix="ocl")
package ocl;

@namespace(uri="http://www.eclipse.org/ocl/1.1.0/OCL/Types", prefix="ocl.types")
package types {
  class AnyType <O> extends utilities.PredefinedType<O> {
  }

  class BagType <C, O> extends CollectionType<C, O> {
  }

  class CollectionType <C, O> extends utilities.PredefinedType<O>, utilities.TypedASTNode {
    op O[*] oclIterators();
    ref C elementType;
    readonly volatile transient attr expressions.CollectionKind[1] kind;
  }

  class ElementType {
  }

  class InvalidType <O> extends utilities.PredefinedType<O> {
  }

  class MessageType <C, O, P> extends utilities.PredefinedType<O> {
    op P[*] oclProperties();
    ref O referredOperation;
    ref C referredSignal;
  }

  class OrderedSetType <C, O> extends CollectionType<C, O> {
  }

  class PrimitiveType <O> extends utilities.PredefinedType<O> {
  }

  class SequenceType <C, O> extends CollectionType<C, O> {
  }

  class SetType <C, O> extends CollectionType<C, O> {
  }

  class TupleType <O, P> extends utilities.PredefinedType<O> {
```

Listing 1: A floating example (float=bp causes bottom page)

```
for i:=maxint to 0 do
begin
{ do nothing }
end;
Write( Case insensitive );
WritE( Pascal keywords. );
```

```
45      op P[*] oclProperties();
46    }
47
48    class TypeType <C, O> extends utilities.PredefinedType<O> {
49      readonly transient ref C[1] referredType;
50    }
51
52    class VoidType <O> extends utilities.PredefinedType<O> {
53    }
54
55  }
56
57  @namespace(uri="http://www.eclipse.org/ocl/1.1.0/OCL/Expressions", prefix="ocl.expr")
58  package expressions {
59    class AssociationClassCallExp <C, P> extends NavigationCallExp<C, P> {
60      ref C referredAssociationClass;
61    }
62
63    class BooleanLiteralExp <C> extends PrimitiveLiteralExp<C> {
64      attr Boolean booleanSymbol;
65    }
66
67    abstract class CallExp <C> extends OCLExpression<C>, utilities.CallingASTNode {
68      val OCLExpression<C> source;
69    }
70
71    class CollectionItem <C> extends CollectionLiteralPart<C> {
72      val OCLExpression<C>[1] item;
73    }
74
75    enum CollectionKind {
76      set = 0;
77      orderedSet = 1;
78      bag = 2;
79      sequence = 3;
80      collection = 4;
81    }
82
83    class CollectionLiteralExp <C> extends LiteralExp<C> {
84
85      @ExtendedMetaData(name="kind")
86      attr CollectionKind kind;
87      val CollectionLiteralPart<C>[*] part;
88      readonly volatile transient attr boolean simpleRange;
89    }
90
91    class CollectionLiteralPart <C> extends utilities.TypedElement<C>, utilities.Visitable {
92    }
93
94    class CollectionRange <C> extends CollectionLiteralPart<C> {
95      val OCLExpression<C>[1] first;
96      val OCLExpression<C>[1] last;
97    }
98
99    class EnumLiteralExp <C, EL> extends LiteralExp<C> {
100     ref EL referredEnumLiteral;
101   }
```

```
102
103   abstract class FeatureCallExp <C> extends CallExp<C> {
104     attr boolean markedPre;
105   }
106
107   class IfExp <C> extends OCLExpression<C> {
108     val OCLExpression<C> condition;
109     val OCLExpression<C> thenExpression;
110     val OCLExpression<C> elseExpression;
111   }
112
113   class IntegerLiteralExp <C> extends NumericLiteralExp<C> {
114     attr Integer integerSymbol;
115   }
116
117   class UnlimitedNaturalLiteralExp <C> extends NumericLiteralExp<C> {
118     attr Integer integerSymbol;
119     readonly volatile transient derived attr boolean[1] unlimited;
120   }
121
122   class InvalidLiteralExp <C> extends LiteralExp<C> {
123   }
124
125   class IterateExp <C, PM> extends LoopExp<C, PM> {
126     val Variable<C, PM> result;
127   }
128
129   class IteratorExp <C, PM> extends LoopExp<C, PM> {
130   }
131
132   class LetExp <C, PM> extends OCLExpression<C> {
133     val OCLExpression<C> in;
134     val Variable<C, PM> variable;
135   }
136
137   abstract class LiteralExp <C> extends OCLExpression<C> {
138   }
139
140   class LoopExp <C, PM> extends CallExp<C> {
141     val OCLExpression<C> body;
142     val Variable<C, PM>[*] iterator;
143   }
144
145   class MessageExp <C, COA, SSA> extends OCLExpression<C>, utilities.CallingASTNode {
146     val OCLExpression<C> target;
147     val OCLExpression<C>[*] argument;
148     val COA calledOperation;
149     val SSA sentSignal;
150   }
151
152   abstract class NavigationCallExp <C, P> extends FeatureCallExp<C> {
153     val OCLExpression<C>[*] qualifier;
154     ref P navigationSource;
155   }
156
157   class NullLiteralExp <C> extends LiteralExp<C> {
158   }
```

```
159
160   abstract class NumericLiteralExp <C> extends PrimitiveLiteralExp<C> {
161   }
162
163   @ExtendedMetaData(name="OclExpression")
164   abstract class OCLExpression <C> extends utilities.TypedElement<C>, utilities.Visitable, utilities.ASTNode {
165   }
166
167   class OperationCallExp <C, O> extends FeatureCallExp<C> {
168     val OCLExpression<C>[*] argument;
169     ref O referredOperation;
170     volatile transient attr int operationCode;
171   }
172
173   abstract class PrimitiveLiteralExp <C> extends LiteralExp<C> {
174   }
175
176   class PropertyCallExp <C, P> extends NavigationCallExp<C, P> {
177     ref P referredProperty;
178   }
179
180   class RealLiteralExp <C> extends NumericLiteralExp<C> {
181     attr Double realSymbol;
182   }
183
184   class StateExp <C, S> extends OCLExpression<C> {
185     ref S referredState;
186   }
187
188   class StringLiteralExp <C> extends PrimitiveLiteralExp<C> {
189     attr String stringSymbol;
190   }
191
192   class TupleLiteralExp <C, P> extends LiteralExp<C> {
193     val TupleLiteralPart<C, P>[*] part;
194   }
195
196   class TupleLiteralPart <C, P> extends utilities.Visitable, utilities.TypedASTNode, utilities.TypedElement<C> {
197     val OCLExpression<C> value;
198     ref P attribute;
199   }
200
201   class TypeExp <C> extends OCLExpression<C> {
202     ref C referredType;
203   }
204
205   class UnspecifiedValueExp <C> extends OCLExpression<C>, utilities.TypedASTNode {
206   }
207
208   class Variable <C, PM> extends utilities.TypedElement<C>, utilities.Visitable, utilities.TypedASTNode {
209     val OCLExpression<C> initExpression;
210     ref PM representedParameter;
211   }
212
213   class VariableExp <C, PM> extends OCLExpression<C> {
214     ref Variable<C, PM> referredVariable;
215   }
```

```
}

@namespace(uri="http://www.eclipse.org/ocl/1.1.0/OCL/Utilities", prefix="ocl.util")
package utilities {
  abstract interface ASTNode {
    transient attr int startPosition = -1;
    transient attr int endPosition = -1;
  }

  abstract interface CallingASTNode extends ASTNode {
    transient attr int propertyStartPosition = -1;
    transient attr int propertyEndPosition = -1;
  }

  abstract interface PredefinedType <O> {
    op String[1] getName();
    op O[*] oclOperations();
  }

  abstract interface TypedASTNode extends ASTNode {
    transient attr int typeStartPosition = -1;
    transient attr int typeEndPosition = -1;
  }

  abstract interface Visitable {
    op  <T, U extends Visitor<T, ?, ?, ?, ?, ?, ?, ?, ?, ?>> T accept(U v);
  }

  abstract interface TypedElement <C> {
    attr String name;
    ref C[1] type;
  }

  abstract interface Visitor <T, C, O, P, EL, PM, S, COA, SSA, CT> {
    op T visitVariableExp(expressions.VariableExp<C, PM>[1] variableExp);
    op T visitLetExp(expressions.LetExp<C, PM>[1] letExp);
    op T visitIfExp(expressions.IfExp<C>[1] ifExp);
    op T visitTypeExp(expressions.TypeExp<C>[1] typeExp);
    op T visitPropertyCallExp(expressions.PropertyCallExp<C, P>[1] callExp);
    op T visitOperationCallExp(expressions.OperationCallExp<C, O>[1] callExp);
    op T visitAssociationClassCallExp(expressions.AssociationClassCallExp<C, P>[1] callExp);
    op T visitIteratorExp(expressions.IteratorExp<C, PM>[1] callExp);
    op T visitIterateExp(expressions.IterateExp<C, PM>[1] callExp);
    op T visitUnspecifiedValueExp(expressions.UnspecifiedValueExp<C>[1] unspecExp);
    op T visitIntegerLiteralExp(expressions.IntegerLiteralExp<C>[1] literalExp);
    op T visitUnlimitedNaturalLiteralExp(expressions.UnlimitedNaturalLiteralExp<C>[1] literalExp);
    op T visitRealLiteralExp(expressions.RealLiteralExp<C>[1] literalExp);
    op T visitStringLiteralExp(expressions.StringLiteralExp<C>[1] literalExp);
    op T visitBooleanLiteralExp(expressions.BooleanLiteralExp<C>[1] literalExp);
    op T visitEnumLiteralExp(expressions.EnumLiteralExp<C, EL>[1] literalExp);
    op T visitCollectionLiteralExp(expressions.CollectionLiteralExp<C>[1] literalExp);
    op T visitCollectionItem(expressions.CollectionItem<C>[1] item);
    op T visitCollectionRange(expressions.CollectionRange<C>[1] range);
    op T visitTupleLiteralExp(expressions.TupleLiteralExp<C, P>[1] literalExp);
    op T visitTupleLiteralPart(expressions.TupleLiteralPart<C, P>[1] part);
    op T visitInvalidLiteralExp(expressions.InvalidLiteralExp<C>[1] literalExp);
```

```
273     op T visitNullLiteralExp(expressions.NullLiteralExp<C>[1] literalExp);
274     op T visitStateExp(expressions.StateExp<C, S>[1] stateExp);
275     op T visitMessageExp(expressions.MessageExp<C, COA, SSA>[1] messageExp);
276     op T visitVariable(expressions.Variable<C, PM>[1] variable);
277     op T visitExpressionInOCL(ExpressionInOCL<C, PM>[1] expression);
278     op T visitConstraint(CT[1] constraint);
279   }
280
281   @ExtendedMetaData(name="ExpressionInOcl")
282   abstract interface ExpressionInOCL <C, PM> extends Visitable {
283     val expressions.OCLExpression<C>[1] bodyExpression;
284     val expressions.Variable<C, PM>[1] contextVariable;
285     val expressions.Variable<C, PM> resultVariable;
286     val expressions.Variable<C, PM>[*] parameterVariable;
287   }
288
289 }
```