Tonto EBNF

In this section, we delineate the Extended Backus-Naur Form (EBNF) declaration pertinent to Tonto.

```
grammar Tonto

entry Model:
    imports+=Import*
    packageDeclaration=PackageDeclaration
;

/**
 * Package declaration
 */

PackageDeclaration:
    (isGlobal?='global')? 'package' (name=QualifiedName | name=S

    declarations+=Declaration*
;

Import:
    'import' referencedModel=[PackageDeclaration:QualifiedName]
;

// <--- Declarations --->

Declaration:
    ClassDeclarationRule | AuxiliaryDeclaration
;

AuxiliaryDeclaration:
```

```
    DataType | Enum | GeneralizationSetImpl | GeneralizationSetS
;

/**
* Class Declaration
*/

ClassDeclarationRule returns ClassDeclaration:
    classElementType=OntologicalCategory
    name=QualifiedName
    ontologicalNatures=ElementOntologicalNature?
    ('(' 'instanceOf' instanceOf=[ClassDeclaration:QualifiedName
    ('specializes' specializationEndurants+=[ClassDeclaration:Qu
        (',' specializationEndurants+=[ClassDeclaration:Qualifie
        )?
        ('{'
            (attributes+=Attribute | references+=InternalRelatio
    '}')?
;

type DataTypeOrClass = DataType | ClassDeclaration;
type DataTypeOrClassOrRelation = DataType | ClassDeclaration | R

/**
* Ontological Category
*/

OntologicalCategory:
    ontologicalCategory=(UnspecifiedType | NonEndurantType | End
;

UnspecifiedType returns string:
    'class'
;

NonEndurantType returns string:
```

```
    'event' | 'situation' | 'process'
;

EndurantType returns string:
    NonSortal | UltimateSortal | Sortal;


NonSortal returns string:
    'category' | 'mixin' | 'phaseMixin' | 'roleMixin' | 'histori
;

UltimateSortal returns string:
    'kind' | 'collective' | 'quantity' | 'quality' | 'mode' | 'i
;

Sortal returns string:
    'subkind' | 'phase' | 'role' | 'historicalRole'

/**
* Ontological Nature
*/

ElementOntologicalNature:
    'of' natures+=OntologicalNature (',' natures+=OntologicalNat
;

OntologicalNature returns string:
    'objects' | 'functional-complexes' | 'collectives'  | 'quant
    'relators' | 'intrinsic-modes' | 'extrinsic-modes' | 'qualit
    'events' | 'situations' |  'types' | 'abstract-individuals'
;

/**
* Attributes
*/
```

```
Attribute:
    name=ID ':' attributeTypeRef=[DataType:QualifiedName]
    cardinality=Cardinality?
    ('{'((isOrdered?='ordered') & (isConst?='const') & (isDerive
;

Cardinality:
    '[' lowerBound=(INT | '*')
    ('..' upperBound=(INT | '*'))? ']'
;

/**
* Relations
*/

ElementRelation:
    InternalRelation | ExternalRelation
;

InternalRelation infers ElementRelation:
    ('@'relationType=RelationStereotype)?
    RelationData
;

ExternalRelation infers ElementRelation:
    ('@'relationType=RelationStereotype)?
    'relation' firstEnd=[ClassDeclaration:QualifiedName]
    RelationData
;

fragment RelationData:
    firstEndMetaAttributes=RelationMetaAttributes?
    firstCardinality=Cardinality?
    RelationName
    secondCardinality=Cardinality?
    secondEnd=[DataTypeOrClassOrRelation:ID]
```

```
    secondEndMetaAttributes=RelationMetaAttributes?
    ('specializes' specializeRelation=[ElementRelation:Qualifie
    (hasInverse='inverseOf' inverseEnd=[ElementRelation:Qualifi
;

fragment RelationName:
    (RelationType ((name=QualifiedName | name=STRING) '--')?) |
    (('--' name=QualifiedName | name=STRING)? RelationInvertedTy
;

fragment RelationType:
 (isAssociation?='--' | isAggregation?='<>--' | isComposition?=
;

fragment RelationInvertedType:
    isAggregationInverted?='--<>' | isCompositionInverted?='--<
;

RelationMetaAttributes:
    '('
    ('{' endMetaAttributes+=RelationMetaAttribute
    (',' endMetaAttributes+=RelationMetaAttribute )* '}')?
    (endName=ID)?
    ')'
;

RelationMetaAttribute:
    isOrdered?='ordered' | isConst?='const' | isDerived?='derive
    ('subsets' subsetRelation=[ElementRelation:QualifiedName] )
    ('redefines' redefinesRelation=[ElementRelation:QualifiedNam
;

RelationStereotype returns string:
    'material' |
    'derivation' |
```

```
    'comparative' |
    'mediation' |
    'characterization' |
    'externalDependence' |
    'componentOf' |
    'memberOf' |
    'subCollectionOf' |
    'subQuantityOf' |
    'instantiation' |
    'termination' |
    'participational' |
    'participation' |
    'historicalDependence' |
    'creation' |
    'manifestation' |
    'bringsAbout' |
    'triggers' |
    'composition' |
    'aggregation' |
    'inherence' |
    'value' |
    'formal' |
    'manifestation' |
    'constitution' |
    ID |
    STRING
;

/**
* GenSets
*/

type ClassDeclarationOrRelation = ClassDeclaration | ElementRela

GeneralizationSet:
    (disjoint?='disjoint')? (complete?='complete')?
```

```
    'genset' name=ID '{'
            (
                'general' generalItem=[ClassDeclarationOrRelatio
                ('categorizer' categorizerItems+=[ClassDeclarati
                'specifics' specificItems+=[ClassDeclarationOrRe
                (',' specificItems+=[ClassDeclarationOrRelation
            )
    '}'
;

GeneralizationSetShort returns GeneralizationSet:
    (disjoint?='disjoint')? (complete?='complete')?
    'genset' name=ID 'where'
    specificItems+=[ClassDeclarationOrRelation:QualifiedName]
    'specializes' generalItem=[ClassDeclarationOrRelation:Qualif
;

/**
* DataTypes
*/

DataType:
    'datatype' name=ID ontologicalNature=ElementOntologicalNatur
    ('specializes' specializationEndurants+=[DataTypeOrClass:Qua
    (',' specializationEndurants+=[DataTypeOrClass:QualifiedName
    )?
    ('{'
        (attributes+=Attribute)*
    '}')?
;

Enum infers DataType:
    isEnum?='enum' name=ID
    ('specializes' specializationEndurants+=[DataTypeOrClass:Qua
    (',' specializationEndurants+=[DataTypeOrClass:QualifiedName
    )?
```

```
    '{'
        (elements+=EnumElement
        ((',') elements+=EnumElement)*)?
    '}'
;

EnumElement:
    name=ID
;

/**
* Terminals
*/

hidden terminal WS: /\s+/;
terminal ID: /[_a-zA-Z][\w_]*/;
terminal INT returns number: /[0-9]+/;
terminal STRING: /"[^"]*"|'[^']*'/;

hidden terminal ML_COMMENT: /\/\*[\s\S]*?\*\//;
hidden terminal SL_COMMENT: /\/\/[^\n\r]*/;

QualifiedName returns string:
    ID ('.' ID)*
;
```