LexEVS 6.0 CTS2 Query 1 - Association Query Operation API

Contents of this Page

CTS2 Links for LexEVS 6.0

- CTS2 API Main Page
- Programmer's Guide Main Page
- LexEVS 6.0 Main Page
- LexEVS Current Release

Introduction

LexEVS CTS2 Association Query Operation API provides capability to query Associations available in the system.

Interface

`org.lexevs.cts2.query.AssociationQueryOperation` is the main interface for all the queries against Associations. This interface can be accessed using main LexEVSCTS2 interface:

```
org.lexevs.cts2.query.AssociationQueryOperation associationQueryOp = new org.lexevs.cts2.LexEvsCTS2Impl().
getQueryOperation().getAssociationQueryOperation();
```

Query Functions

Here are the major query functions available using AssociationQueryOperation interface:

listAssociations

This function returns the resolved concept reference (which contains the associations) according to given node.

```
listAssociations(String codingSystemName CodingSchemeVersionOrTag versionOrTag String namespace String code
String associationName boolean isBackward int depth int maxToReturn)
```

Description:	Returns the resolved concept reference (which contains the associations) according to given node.
Input:	• ***java.lang.String codingSystemName*** - The code system of the Associations to be returned. • ***org.LexGrid.LexBIG.DataModel.Core.CodingSchemeVersionOrTag versionOrTag*** - (Optional) Version identifier of the code system (defaults to 'PRODUCTION' tagged code systems) • ***java.lang.String namespace*** - The namespace of the Entity focus. • ***java.lang.String code*** - The code of the Entity focus. • ***java.lang.String associationName*** - (Optional) The Association to restrict to (if null, all Associations will be returned) • ***boolean isBackward*** - The code system of the Associations to be returned. • ***integer depth*** - The Association depth to traverse • ***integer maxToReturn*** - The max number of results to return (use '-1' for 'no limit')
Output:	***org.LexGrid.LexBIG.DataModel.Collections.ResolvedConceptReferenceList*** - List of Associations

Sample Call:	• *Step 1:* Instantiate CodeSystemQueryOperation if it is not done yet:
	`org.lexevs.cts2.query.AssociationQueryOperation associationQueryOp = new org.lexevs.cts2.LexEvsCTS2Impl().getQueryOperation().getAssociationQueryOperation();`
	• *Step 2:* To get all the Associations for a given code and namespace:
	`org.LexGrid.LexBIG.DataModel.Collections.CodingSchemeRenderingList csrList = associationQueryOp.listAssociations("CodeSystem",null,"ns","code",null,false,1,-1);`

determineTransitiveConceptRelationship

Returns the path according to given two nodes.

`determineTransitiveConceptRelationship(String codingSystemName CodingSchemeVersionOrTag versionOrTag String relationContainerName String associationName String sourceCode String sourceNamespace String targetCode String targetNamespace)`

Description:	Returns the path according to given two nodes.
Input:	• ***java.lang.String codingSystemName*** - The code system of the Associations to be returned. • ***org.LexGrid.LexBIG.DataModel.Core.CodingSchemeVersionOrTag versionOrTag*** - (Optional) Version identifier of the code system (defaults to 'PRODUCTION' tagged code systems) • ***java.lang.String relationContainerName*** - (Optional) The Relations Container name. • ***java.lang.String associationName*** - (Optional) The Association to restrict to (if null, all Associations will be used) • ***java.lang.String sourceCode*** - The source Entity code. • ***java.lang.String sourceNamespace*** - The source Entity namespace. • ***java.lang.String targetCode*** - The target Entity code. • ***java.lang.String targetNamespace*** - The target Entity namespace.
Output:	***org.LexGrid.LexBIG.DataModel.Core.ResolvedConceptReference*** - The path between the two nodes.

Sample Call:	• *Step 1:* Instantiate CodeSystemQueryOperation if it is not done yet:
	`org.lexevs.cts2.query.AssociationQueryOperation associationQueryOp = new org.lexevs.cts2.LexEvsCTS2Impl().getQueryOperation().getAssociationQueryOperation();`
	• *Step 2:* To get all the Associations for a given code and namespace:
	`org.LexGrid.LexBIG.DataModel.Collections.CodingSchemeRenderingList csrList = associationQueryOp.determineTransitiveConceptRelationship("CodeSystem",null,"r1","hasSubtype","sc","sns","tc","tns");`

computeSubsumptionRelationship

Return whether the two nodes has a transitive closure path.

`determineTransitiveConceptRelationship(String codingSystemName CodingSchemeVersionOrTag versionOrTag String associationType ConceptReference sourceCode, ConceptReference targetCode)`

Description:	Return whether the two nodes has a transitive closure path.

Input:	• ***java.lang.String codingSystemName*** - The code system of the Associations to be returned. • ***org.LexGrid.LexBIG.DataModel.Core.CodingSchemeVersionOrTag versionOrTag*** - (Optional) Version identifier of the code system (defaults to 'PRODUCTION' tagged code systems) • ***java.lang.String associationType*** - (Optional) The Association to restrict to (if null, all Associations will be used) • ***org.LexGrid.LexBIG.DataModel.Core.ConceptReference sourceCode*** - The source Entity. • ***org.LexGrid.LexBIG.DataModel.Core.ConceptReference sourceNamespace*** - The target Entity.
Output:	***boolean*** - 'true' if the two nodes have a transitive closure path, 'false' if not.

Sample Call:	• *Step 1:* Instantiate CodeSystemQueryOperation if it is not done yet:
	`org.lexevs.cts2.query.AssociationQueryOperation associationQueryOp = new org.lexevs.cts2.LexEvsCTS2Impl().getQueryOperation().getAssociationQueryOperation();`
	• *Step 2a:* Create Concept References:
	`ConceptReference source = new ConceptReference();`
	`source.setCode = "sourceCode";`
	`ConceptReference target= new ConceptReference();`
	`target.setCode = "targetCode";`
	• *Step 2b:* To check the ConceptReferences for subsumption:
	`boolean isSubsumed = associationQueryOp.computeSubsumptionRelationship("CodeSystem",null,"hasSubtype",source,target");`

getAssociationDetails

Return association triple according to association instance id.

`getAssociationDetails(String codingSchemeName, CodingSchemeVersionOrTag versionOrTag, String associationInstanceId)`

Description:	Return association triple according to association instance id.
Input:	• ***java.lang.String codingSchemeName*** - (**Mandatory**) Name of the code system. • ***org.LexGrid.LexBIG.DataModel.Core.CodingSchemeVersionOrTag versionOrTag*** - (Optional) Version or tag (like 'dev', 'production' etc) of the code system. • ***java.lang.String associationInstanceId*** - (**Mandatory**) The unique id of the triple.
Output:	***org.lexevs.dao.database.service.association.AssociationService.AssociationTriple*** - Detailed Association Triple object

Sample Call:

- *Step 1:* Instantiate CodeSystemQueryOperation if it is not done yet:

```
org.lexevs.cts2.query.AssociationQueryOperation associationQueryOp = new org.lexevs.cts2.
LexEvsCTS2Impl().getQueryOperation().getAssociationQueryOperation();
```

- *Step 2:* Populate CodeSystemVersionOrTag object:

```
CodingSchemeVersionOrTag versionOrTag = new CodingSchemeVersionOrTag();
versionOrTag.setVersion("1.0");
```

- *Step 3:* Call getAssociationDetails method by providing code system version and association instance id:

```
AssociationTriple triple = csQueryOp.getAssociationDetails ("Automobiles", versionOrTag,
"id12345");
```