

LexEVS 6.0 CTS2 Query 1 - Association Query Operation API

Contents of this Page

- [Introduction](#)
- [Interface](#)
- [Query Functions](#)
 - [listAssociations](#)
 - [determineTransitiveConceptRelationship](#)
 - [computeSubsumptionRelationship](#)
 - [getAssociationDetails](#)

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:	• <i>java.lang.String codingSystemName</i> - The code system of the Associations to be returned. • <i>org.LexGrid.LexBIG.DataModel.Core.CodingSchemeVersionOrTag versionOrTag</i> - (Optional) Version identifier of the code system (defaults to 'PRODUCTION' tagged code systems) • <i>java.lang.String namespace</i> - The namespace of the Entity focus. • <i>java.lang.String code</i> - The code of the Entity focus. • <i>java.lang.String associationName</i> - (Optional) The Association to restrict to (if null, all Associations will be returned) • <i>boolean isBackward</i> - The code system of the Associations to be returned. • <i>integer depth</i> - The Association depth to traverse • <i>integer maxToReturn</i> - The max number of results to return (use '-1' for 'no limit')
Output:	<i>org.LexGrid.LexBIG.DataModel.Collections.ResolvedConceptReferenceList</i> - List of Associations

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

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:	• <i>java.lang.String codingSystemName</i> - The code system of the Associations to be returned. • <i>org.LexGrid.LexBIG.DataModel.Core.CodingSchemeVersionOrTag versionOrTag</i> - (Optional) Version identifier of the code system (defaults to 'PRODUCTION' tagged code systems) • <i>java.lang.String relationContainerName</i> - (Optional) The Relations Container name. • <i>java.lang.String associationName</i> - (Optional) The Association to restrict to (if null, all Associations will be used) • <i>java.lang.String sourceCode</i> - The source Entity code. • <i>java.lang.String sourceNamespace</i> - The source Entity namespace. • <i>java.lang.String targetCode</i> - The target Entity code. • <i>java.lang.String targetNamespace</i> - The target Entity namespace.
Output:	<i>org.LexGrid.LexBIG.DataModel.Core.ResolvedConceptReference</i> - The path between the two nodes.
Sample Call:	• <i>Step 1:</i> Instantiate CodeSystemQueryOperation if it is not done yet: <pre>org.lexevs.cts2.query.AssociationQueryOperation associationQueryOp = new org.lexevs.cts2.LexEvsCTS2Impl().getQueryOperation().getAssociationQueryOperation();</pre> • <i>Step 2:</i> To get all the Associations for a given code and namespace: <pre>org.LexGrid.LexBIG.DataModel.Collections.CodingSchemeRenderingList csrList = associationQueryOp.determineTransitiveConceptRelationship("CodeSystem", null, "r1", "hasSubtype", "sc", "sns", "tc", "tns");</pre>

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:	• <i>java.lang.String codingSystemName</i> - The code system of the Associations to be returned. • <i>org.LexGrid.LexBIG.DataModel.Core.CodingSchemeVersionOrTag versionOrTag</i> - (Optional) Version identifier of the code system (defaults to 'PRODUCTION' tagged code systems) • <i>java.lang.String associationType</i> - (Optional) The Association to restrict to (if null, all Associations will be used) • <i>org.LexGrid.LexBIG.DataModel.Core.ConceptReference sourceCode</i> - The source Entity. • <i>org.LexGrid.LexBIG.DataModel.Core.ConceptReference sourceNamespace</i> - The target Entity.
Output:	<i>boolean</i> - 'true' if the two nodes have a transitive closure path, 'false' if not.
Sample Call:	• <i>Step 1:</i> Instantiate CodeSystemQueryOperation if it is not done yet: <pre>org.lexevs.cts2.query.AssociationQueryOperation associationQueryOp = new org.lexevs.cts2.LexEvsCTS2Impl().getQueryOperation().getAssociationQueryOperation();</pre> • <i>Step 2a:</i> Create Concept References: <pre>ConceptReference source = new ConceptReference(); source.setCode = "sourceCode"; ConceptReference target= new ConceptReference(); target.setCode = "targetCode";</pre> • <i>Step 2b:</i> To check the ConceptReferences for subsumption: <pre>boolean isSubsumed = associationQueryOp.computeSubsumptionRelationship("CodeSystem", null, "hasSubtype", source, target);</pre>

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:	• <i>java.lang.String codingSchemeName</i> - (Mandatory) Name of the code system. • <i>org.LexGrid.LexBIG.DataModel.Core.CodingSchemeVersionOrTag versionOrTag</i> - (Optional) Version or tag (like 'dev', 'production' etc) of the code system. • <i>java.lang.String associationInstanceId</i> - (Mandatory) The unique id of the triple.
Output:	<i>org.lexevs.dao.database.service.association.AssociationService.AssociationTriple</i> - 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");
```