LexEVS 6.0 CTS2 Administration 1 - Load 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 Load API provides capability to load complete or incremental updates of Code System, Value Sets and Association contents. It also provides capability to activate and deactivate loaded contents.

Load Interfaces

There are three major load interfaces proved, each for loading specific content:

- Code System loader - Provides capability to load complete or partial contents of Code System plus functions to activate and deactivate loaded Code System.
- Value Sets loader - Provides capability to load Value Sets.
- Association loader - Provides capability to load Associations.

Each of these interfaces can be accessed using:

```
org.lexevs.cts2.admin.load.CodeSystemLoadOperation csLoadOp = new org.lexevs.cts2.LexEvsCTS2Impl().
getAdminOperation().getCodeSystemLoadOperation();
org.lexevs.cts2.admin.load.ValueSetLoadOperation vsLoadOp = new org.lexevs.cts2.LexEvsCTS2Impl().
getAdminOperation().getValueSetLoadOperation();
org.lexevs.cts2.admin.load.AssociationLoadOperation assnLoadOp = new org.lexevs.cts2.LexEvsCTS2Impl().
getAdminOperation().getAssociationLoadOperation();
```

Code System Loader

`org.lexevs.cts2.admin.load.CodeSystemLoadOperation` is the main interface which can be used to load, activate and deactivate Code System. This interface can be accessed using main LexEVSCTS2 interface, like:

```
org.lexevs.cts2.admin.load.CodeSystemLoadOperation csLoadOp = new org.lexevs.cts2.LexEvsCTS2Impl().
getAdminOperation().getCodeSystemLoadOperation();
```

There are two different methods available to load Code System:

- Loading Code System from a file - This method provides capability to load complete Code System contents that are present in a file system.
- Loading Code System Object - This method provides capability to load supplied Code System Object.

Loading Code System from a File

This function provides the capability to load Code System found in the file using the loader specified.

```
load(URI source, URI metadata, URI manifest, String loaderName, Boolean stopOnErrors, Boolean async, Boolean overwriteMetadata, String versionTag, Boolean activate)
```

Description:	Loads Code System found in source file using the loader specified.
Input:	• ***java.net.URI source*** - (**Mandatory**) URI corresponding to the code system file. • ***java.net.URI metadata*** - (Optional) URI of the XML file containing custom code system meta data. Loads additional data to be maintained and queried as terminology meta-information within the system. All tags and values are interpreted as simple text-based key/value pairs. • ***java.net.URI manifest*** - (Optional) URI corresponding to the manifest file. The LexGrid Manifest accommodates the need to supplement or override default information provided by the source. More specifically, the manifest provides a means to customize the same code system metadata defined by the LexGrid model, since each element of the manifest extends directly from an element used to define the LexGrid coding scheme(aka code system) object. Each extended element allows for the administrator to specify whether the manifest definition replaces or supplements original values provided in the terminology source. Like the LexGrid Terminology model, the manifest is defined by a formal model mastered as XML Schema. • ***java.lang.String loaderName*** - (**Mandatory**) Loader to use for loading the code system. Use getSupportedLoaderNames() method to get all the loaders supported by the service. For example, 'OBOLoader' could be used to load code system source that is in OBO format, 'OWLLoader' for code system source in OWL format, 'LexGrid_Loader' for Code System in LexGrid XML format, etc. • ***boolean stopOnErrors*** - (Optional) default is false. True means stop if any load error is detected. False means attempt to load what can be loaded if recoverable errors are encountered. • ***boolean async*** - (Optional) Flag controlling whether load occurs in the calling thread. If true, the load will occur in a separate asynchronous process. If false, this method blocks until the load operation completes or fails. Regardless of setting, the getStatus and getLog calls are used to fetch results. • ***boolean_overwriteMetadata*** - (Optional) If true, existing meta data for the code system will be erased. If false, new meta data will be appended to existing meta data. • ***java.lang.String versionTag*** - (Optional) The tag (e.g "devel", "production", ...) to be set for the this code system. • ***boolean activate*** - (Optional) True: activates the code system after the load.
Output:	Array of ***edu.mayo.informatics.lexgrid.convert.utility.URNVersionPair*** - URN and Version of the loaded code system
Exception:	***org.LexGrid.LexBIG.Exceptions.LBException***

Sample Call:	• *Step 1:* Instantiate CodeSystemLoadOperation if it is not done yet:
	`org.lexevs.cts2.admin.load.CodeSystemLoadOperation csLoadOp = new org.lexevs.cts2.LexEvsCTS2Impl().getAdminOperation().getCodeSystemLoadOperation();`
	• *Step 2:* Call load method by passing the inputfile location and other parameter values:
	`URNVersionPair[] csloaded = csLoad.load(new File("resources/testData/fungal_anatomy.obo").toURI(), null, null, "OBOLoader", true, true, true, "DEV", true);`

Loading Code System Object

This function provides the capability to load supplied Code System object.

```
load(CodingScheme codeSystem, URI metadata, Boolean stopOnErrors, Boolean async, Boolean overwriteMetadata, String versionTag, Boolean activate)
```

Description:	Loads supplied Code System Object.
Input:	• ***org.LexGrid.codingSchemes.CodingScheme codeSystem*** - (**Mandatory**) Code System Object to be loaded into the terminology service. • ***java.net.URI metadata*** - (Optional) URI of the XML file containing custom code system meta data. Loads additional data to be maintained and queried as terminology meta-information within the system. All tags and values are interpreted as simple text-based key/value pairs. • ***boolean stopOnErrors*** - (Optional) default is false. True means stop if any load error is detected. False means attempt to load what can be loaded if recoverable errors are encountered. • ***boolean async*** - (Optional) Flag controlling whether load occurs in the calling thread. If true, the load will occur in a separate asynchronous process. If false, this method blocks until the load operation completes or fails. Regardless of setting, the getStatus and getLog calls are used to fetch results. • ***boolean* overwriteMetadata**_ - (Optional) If true, existing meta data for the code system will be erased. If false, new meta data will be appended to existing meta data. • ***java.lang.String versionTag*** - (Optional) The tag (e.g "devel", "production", ...) to be set for the this code system. • ***boolean activate*** - (Optional) True: activates the code system after the load.
Output:	Array of ***edu.mayo.informatics.lexgrid.convert.utility.URNVersionPair*** - URN and Version of the loaded code system
Exception:	***org.LexGrid.LexBIG.Exceptions.LBException***

Sample Call:

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

```
org.lexevs.cts2.admin.load.CodeSystemLoadOperation csLoadOp = new org.lexevs.cts2.
LexEvsCTS2Impl().getAdminOperation().getCodeSystemLoadOperation();
```

- *Step 2:* Populate Code System Object:

```
org.LexGrid.codingSchemes.CodingScheme cs = new org.LexGrid.codingSchemes.CodingScheme();
cs.setApproxNumConcepts(new Long(4));
cs.setCodingSchemeName("miniautomobiles");
cs.setFormalName("miniautomobiles");
cs.setCodingSchemeURI("11.22.33.44");
org.LexGrid.commonTypes.Text copyright = new org.LexGrid.commonTypes.Text();
copyright.setContent("Copyright Mayo Clinic.");
cs.setCopyright(copyright);
cs.setDefaultLanguage("en");
cs.setMappings(new org.LexGrid.naming.Mappings());
cs.setRepresentsVersion("1.1");

org.LexGrid.naming.SupportedAssociation saHasSubType = edu.mayo.informatics.lexgrid.convert.
exporters.xml.lgxml.factory.SupportedAssociationFactory.createSupportedAssociationHasSubType();

cs.getMappings().addSupportedAssociation(saHasSubType);
org.LexGrid.naming.SupportedAssociation saUses = edu.mayo.informatics.lexgrid.convert.exporters.
xml.lgxml.factory.SupportedAssociationFactory.createSupportedAssociationUses();
cs.getMappings().addSupportedAssociation(saUses);

org.LexGrid.relations.Relations rels = new org.LexGrid.relations.Relations();
rels.setContainerName("asD");

org.LexGrid.relations.AssociationPredicate ap = new org.LexGrid.relations.
AssociationPredicate();
ap.setAssociationName("hasSubtype");
rels.addAssociationPredicate(ap);

ap = new org.LexGrid.relations.AssociationPredicate();

ap.setAssociationName("uses");

rels.addAssociationPredicate(ap);

cs.addRelations(rels);
```

- *Step 3:* Call load method by passing the Code System Object and other parameter values:

```
URNVersionPair[] csloaded = csLoad.load(cs, null, true, false, true, "DEV", true);
```

Value Set Loader

Value Set is stored in repository in terms of definitions, in LexEVS, it is known as 'Value Set Definition'. As name indicates, it is a definition of a value set contents, NOT the expanded value set contents that will be loaded. During the runtime, these definitions are resolved against the supplied Code System Version to return expanded Value Set contents. Visit LexEVS 6.0 Value Set and Pick List Definition Guide for detailed information about LexEVS Value Set Definition.

`org.lexevs.cts2.admin.load.ValueSetLoadOperation` is the main interface which can be used to load Value Set Definition. This interface can be accessed using main LexEVSCTS2 interface, like:

```
org.lexevs.cts2.admin.load.ValueSetLoadOperation vsLoadOp = new org.lexevs.cts2.LexEvsCTS2Impl().
getAdminOperation().getValueSetLoadOperation();
```

Similar to Code System, there are two different methods available to load Value Sets:

- Loading Value Set Definition from a file - This method provides capability to load Value Set Definition(s) that are present in a file system.
- Loading Value Set Definition Object - This method provides capability to load supplied Value Set Definition Object.

Loading Value Set Definition from a File

This function provides the capability to load Value Set Definition(s) found in the file using the loader specified.

```
load(URI source, URI releaseURI, String loaderName, Boolean stopOnErrors)
```

Description:	Loads Value Set Definition(s) found in source file using the loader specified.
Input:	• ***java.net.URI source*** - (**Mandatory**) URI corresponding to the Value Set Definition(s) file. • ***java.net.URI releaseURI*** - (Optional) Release URI the loaded contents belong to. • ***java.lang.String loaderName*** - (**Mandatory**) Loader to use for loading the Value Set Definition. Use getSupportedLoaderNames() method to get all the loaders supported by the service. For example, use 'LexGrid_Loader' to load file containing Value Set Definition(s) in LexGrid XML format, etc. • ***boolean stopOnErrors*** - (Optional) default is false. True means stop if any load error is detected. False means attempt to load what can be loaded if recoverable errors are encountered.
Output:	Array of ***edu.mayo.informatics.lexgrid.convert.utility.URNVersionPair*** - URI and Version of the loaded Value Set Definition
Exception:	***org.LexGrid.LexBIG.Exceptions.LBException***

Sample Call:	• *Step 1:* Instantiate ValueSetLoadOperation if it is not done yet:
	`org.lexevs.cts2.admin.load.ValueSetLoadOperation vsLoadOp = new org.lexevs.cts2.LexEvsCTS2Impl().getAdminOperation().getValueSetLoadOperation();`
	• *Step 2:* Call load method by passing the inputfile location and other parameter values:
	`URNVersionPair[] vsloaded = vsLoad.load(new File("resources/testData/vsTestData.xml").toURI(), null, "LexGrid_Loader", true);`

Loading Value Set Definition Object

This function provides the capability to load supplied Value Set Definition object.

```
load(ValueSetDefinition valueSetDefinition, URI releaseURI, Boolean stopOnErrors)
```

Description:	Loads supplied Value Set Definition Object.
Input:	• ***org.LexGrid.valueSets.ValueSetDefinition valueSetDefinition*** - (**Mandatory**) Value Set Definition Object to be loaded into the terminology service. • ***java.net.URI releaseURI*** - (Optional) Release URI the loaded contents belong to. • ***boolean stopOnErrors*** - (Optional) default is false. True means stop if any load error is detected. False means attempt to load what can be loaded if recoverable errors are encountered.
Output:	***java.lang.String*** - URI of Value Set Definition loaded
Exception:	***org.LexGrid.LexBIG.Exceptions.LBException***

Sample Call:	• *Step 1:* Instantiate ValueSetLoadOperation if it is not done yet:
	`org.lexevs.cts2.admin.load.ValueSetLoadOperation vsLoadOp = new org.lexevs.cts2.LexEvsCTS2Impl().getAdminOperation().getValueSetLoadOperation();_`
	• *Step 2:* Populate Code System Object:
	`org.LexGrid.valueSets.ValueSetDefinition vsd = new org.LexGrid.valueSets.ValueSetDefinition();`
	`vsd.setValueSetDefinitionURI("SRITEST:JUNIT:TEST:VSD1");`
	`vsd.setValueSetDefinitionName("JUnit test vsd 1");`
	`vsd.setConceptDomain("cd");`
	`vsd.addRepresentsRealmOrContext("context");`
	`vsd.setDefaultCodingScheme("Automobiles");`
	`vsd.setIsActive(false);`
	`vsd.setOwner("cts2");`
	`java.util.List<org.LexGrid.commonTypes.Source> srcList = new ArrayList<org.LexGrid.commonTypes.Source>();`
	`org.LexGrid.commonTypes.Source src = new org.LexGrid.commonTypes.Source();`
	`src.setContent("lexevs");`
	`srcList.add(src);`
	`src = new org.LexGrid.commonTypes.Source();`
	`src.setContent("cts2");`
	`srcList.add(src);`
	`vsd.setSource(srcList);`
	
	`org.LexGrid.valueSets.DefinitionEntry de = new org.LexGrid.valueSets.DefinitionEntry();`
	`de.setRuleOrder(1L);`
	`de.setOperator(org.LexGrid.valueSets.types.DefinitionOperator.OR);`
	`org.LexGrid.valueSets.CodingSchemeReference csr = new org.LexGrid.valueSets.CodingSchemeReference();`
	`csr.setCodingScheme("Automobiles");`
	`de.setCodingSchemeReference(csr);`
	
	`vsd.addDefinitionEntry(de);`
	• *Step 3:* Call load method by passing the Value Set Definition Object and other parameter values:
	`String vsdURI = vsLoadOp.load(vsd, null, true);;`

Association Loader

`org.lexevs.cts2.admin.load.AssociationLoadOperation` is the main interface which can be used to load Associations with in a Code System. This interface can be accessed using main LexEVSCTS2 interface, like:

```
org.lexevs.cts2.admin.load.AssociationLoadOperation assnLoadOp = new org.lexevs.cts2.LexEvsCTS2Impl().getAdminOperation().getAssociationLoadOperation();
```

There are two different methods available to load Associations:

- Load Associations from a file - This method provides capability to load Associations from a file system.
- Load Associations from an Object - This method provides capability to load Associations supplied with in a Code System Object.

Loading Associations from a File

This function provides the capability to load Associations found in the file using the loader specified.

`importAssociationVersion(URI source, URI metadata, URI manifest, String loaderName, Boolean stopOnErrors, Boolean async, Boolean overwriteMetadata, String versionTag, Boolean activate)`

Description:	Loads Associations found in source file using the loader specified.

Input:	• ***java.net.URI source*** - (**Mandatory**) URI corresponding to the file that contains Associations. • ***java.net.URI metadata*** - (Optional) URI of the XML file containing custom meta data for the code system that contains this associations. Loads additional data to be maintained and queried as terminology meta-information within the system. All tags and values are interpreted as simple text-based key/value pairs. • ***java.net.URI manifest*** - (Optional) URI corresponding to the manifest file. The LexGrid Manifest accommodates the need to supplement or override default information provided by the source. More specifically, the manifest provides a means to customize the same code system metadata defined by the LexGrid model, since each element of the manifest extends directly from an element used to define the LexGrid coding scheme(aka code system) object. Each extended element allows for the administrator to specify whether the manifest definition replaces or supplements original values provided in the terminology source. Like the LexGrid Terminology model, the manifest is defined by a formal model mastered as XML Schema. • ***java.lang.String loaderName*** - (**Mandatory**) Loader to use for loading the associations. Use getSupportedLoaderNames() method to get all the loaders supported by the service. For example, 'OBOLoader' could be used to load code system source that is in OBO format, 'OWLLoader' for code system source in OWL format, 'LexGrid_Loader' for Code System in LexGrid XML format, etc. • ***boolean stopOnErrors*** - (Optional) default is false. True means stop if any load error is detected. False means attempt to load what can be loaded if recoverable errors are encountered. • ***boolean async*** - (Optional) Flag controlling whether load occurs in the calling thread. If true, the load will occur in a separate asynchronous process. If false, this method blocks until the load operation completes or fails. Regardless of setting, the getStatus and getLog calls are used to fetch results. • ***boolean overwriteMetadata_*** - (Optional) If true, existing meta data for the code system will be erased. If false, new meta data will be appended to existing meta data. • ***java.lang.String versionTag*** - (Optional) The tag (e.g "devel", "production", ...) to be set for the the code system that contains this associations. • ***boolean activate*** - (Optional) True: activates the code system that contains this association after the load.
Output:	None
Exception:	***org.LexGrid.LexBIG.Exceptions.LBException***

Sample Call:	• *Step 1:* Instantiate AssociationLoadOperation if it is not done yet:
	`org.lexevs.cts2.admin.load.AssociationLoadOperation assnLoadOp = new org.lexevs.cts2.LexEvsCTS2Impl().getAdminOperation().getAssociationLoadOperation();`
	• *Step 2:* Call load method by passing the inputfile location and other parameter values:
	`assnLoad.importAssociationVersion(new File("resources/testData/fungal_anatomy.obo").toURI(), null, null, "OBOLoader", true, true, true, "DEV", true);`

Loading Associations from an Object

This function provides the capability to load Associations supplied with in a Code System object.

`importAssociationVersion(CodingScheme codeSystem, URI metadata, Boolean stopOnErrors, Boolean async, Boolean overwriteMetadata, String versionTag, Boolean activate)`

Description:	Loads Associations supplied with in Code System Object.
Input:	• ***org.LexGrid.codingSchemes.CodingScheme codeSystem*** - (**Mandatory**) Code System Object that contains Associations to be loaded into the terminology service. • ***java.net.URI metadata*** - (Optional) URI of the XML file containing custom meta data of code system that contains this associations. Loads additional data to be maintained and queried as terminology meta-information within the system. All tags and values are interpreted as simple text-based key/value pairs. • ***boolean stopOnErrors*** - (Optional) default is false. True means stop if any load error is detected. False means attempt to load what can be loaded if recoverable errors are encountered. • ***boolean async*** - (Optional) Flag controlling whether load occurs in the calling thread. If true, the load will occur in a separate asynchronous process. If false, this method blocks until the load operation completes or fails. Regardless of setting, the getStatus and getLog calls are used to fetch results. • **boolean* overwriteMetadata - (Optional) If true, existing meta data for the code system will be erased. If false, new meta data will be appended to existing meta data. • ***java.lang.String versionTag*** - (Optional) The tag (e.g "devel", "production", ...) to be set for the code system that contains this associations. • ***boolean activate*** - (Optional) True: activates the code system that contains this associations after the load.
Output:	None
Exception:	***org.LexGrid.LexBIG.Exceptions.LBException***

Sample Call:

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

```
org.lexevs.cts2.admin.load.AssociationLoadOperation assnLoadOp = new org.lexevs.cts2.
LexEvsCTS2Impl().getAdminOperation().getAssociationLoadOperation();
```

- *Step 2:* Populate Code System Object:

```
org.LexGrid.codingSchemes.CodingScheme cs = new org.LexGrid.codingSchemes.CodingScheme();
cs.setApproxNumConcepts(new Long(4));
cs.setCodingSchemeName("miniautomobiles");
cs.setFormalName("miniautomobiles");
cs.setCodingSchemeURI("11.22.33.44");
org.LexGrid.commonTypes.Text copyright = new org.LexGrid.commonTypes.Text();
copyright.setContent("Copyright Mayo Clinic.");
cs.setCopyright(copyright);
cs.setDefaultLanguage("en");
cs.setMappings(new org.LexGrid.naming.Mappings());
cs.setRepresentsVersion("1.1");

org.LexGrid.naming.SupportedAssociation saHasSubType = edu.mayo.informatics.lexgrid.convert.
exporters.xml.lgxml.factory.SupportedAssociationFactory.createSupportedAssociationHasSubType();

cs.getMappings().addSupportedAssociation(saHasSubType);
org.LexGrid.naming.SupportedAssociation saUses = edu.mayo.informatics.lexgrid.convert.exporters.
xml.lgxml.factory.SupportedAssociationFactory.createSupportedAssociationUses();
cs.getMappings().addSupportedAssociation(saUses);

org.LexGrid.relations.Relations rels = new org.LexGrid.relations.Relations();
rels.setContainerName("asD");

org.LexGrid.relations.AssociationPredicate ap = new org.LexGrid.relations.
AssociationPredicate();
ap.setAssociationName("hasSubtype");
rels.addAssociationPredicate(ap);

ap = new org.LexGrid.relations.AssociationPredicate();

ap.setAssociationName("uses");

rels.addAssociationPredicate(ap);

cs.addRelations(rels);
```

- *Step 3:* Call load method by passing the Code System Object and other parameter values:

```
assnLoad.importAssociationVersion(cs, null, true, false, true, "DEV", true);
```