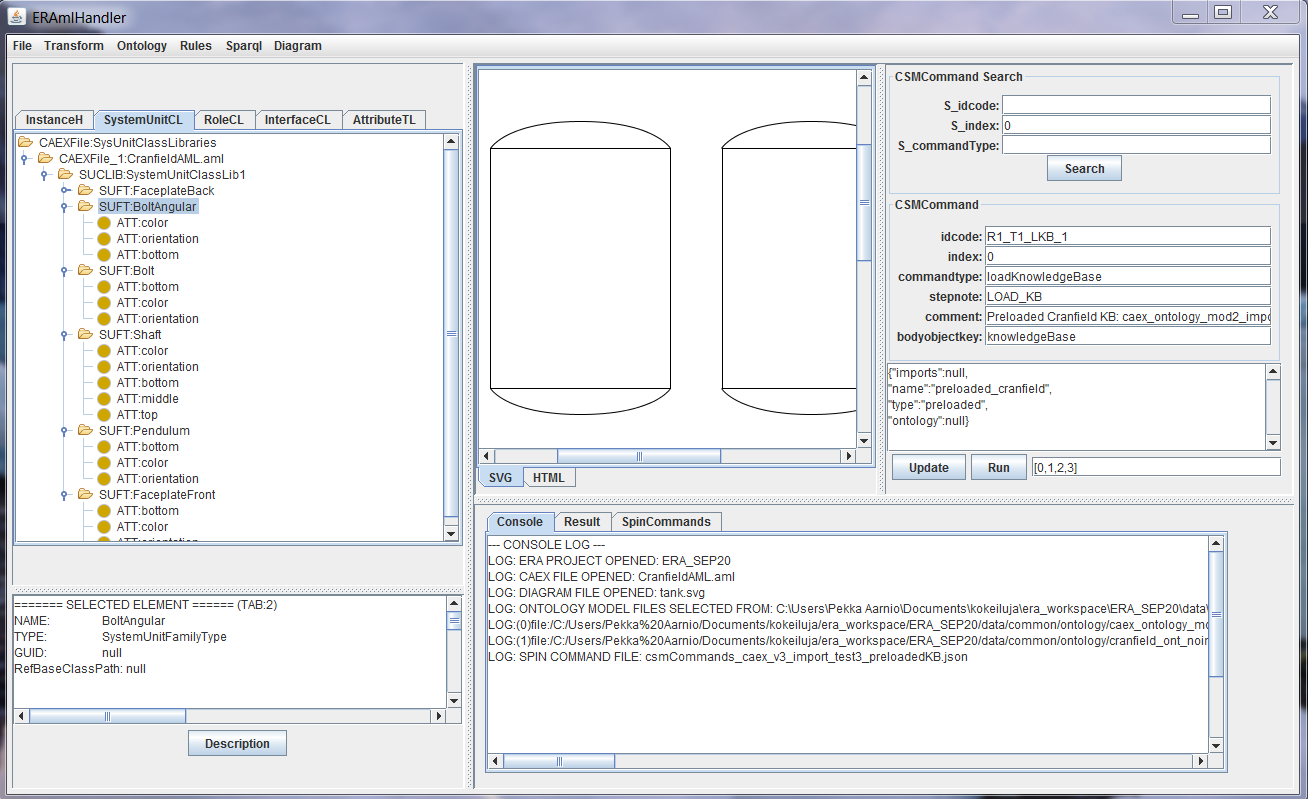
ERAmlHandlerUserGuide\_v2.4.docx

P. Aarnio

2019-05-14;

ERAmlHandler User Guide 2.4

2019-05-14



Version dates:

2019-05-14;

2018-05-24;

Contents

[Introduction 5](#_Toc8749807)

[Creating a new ERA Project 6](#_Toc8749808)

[Loading CAEX Files 7](#_Toc8749809)

[Generating CAEX Ontology Models 8](#_Toc8749810)

[RDFContainer Class 8](#_Toc8749811)

[Merging RDF partial models 9](#_Toc8749812)

[Saving ontology models 10](#_Toc8749813)

[SPARQL and SPIN CSMCommands and Knowledge Base loading 11](#_Toc8749814)

[Example Case1: Loading one ontology file into the knowledge base 12](#_Toc8749815)

[STEP 1: Loading Sparql&Spin Commands file 12](#_Toc8749816)

[STEP2 Searching and displaying the description of the load knowledge base command with index 0 13](#_Toc8749817)

[Example Case2: Loading the main ontology with imports and editing the commands CAEXCranfield 14](#_Toc8749818)

[A: Editing and running a command sequence 14](#_Toc8749819)

[B: Copying a command into CSMStore as a new command example 16](#_Toc8749820)

[C: Changing the order of commands in a workflow 17](#_Toc8749821)

[Example Case3: Preloading the KB from ontology files 19](#_Toc8749822)

[STEP 1: Preload Knowledge Base using: Menu/Sparql/Preload Ontology Files into KB… 19](#_Toc8749823)

[STEP 2: Navigate to Ontology folder and select two ontology files (using +ctrl key): 19](#_Toc8749824)

[STEP 3: Load Spin Command File (.json) 20](#_Toc8749825)

[Example Case 4: Preloading directly from Merged CAEX ontology 22](#_Toc8749826)

[XSL Transforms 23](#_Toc8749827)

[Generic XSL transforms 23](#_Toc8749828)

[Pre-specified XSL transforms 24](#_Toc8749829)

[Caex to JMonkey transformation 24](#_Toc8749830)

[Caex to ASP facts transformation 25](#_Toc8749831)

[ASP Rules 27](#_Toc8749832)

[Generating ASP fact file 27](#_Toc8749833)

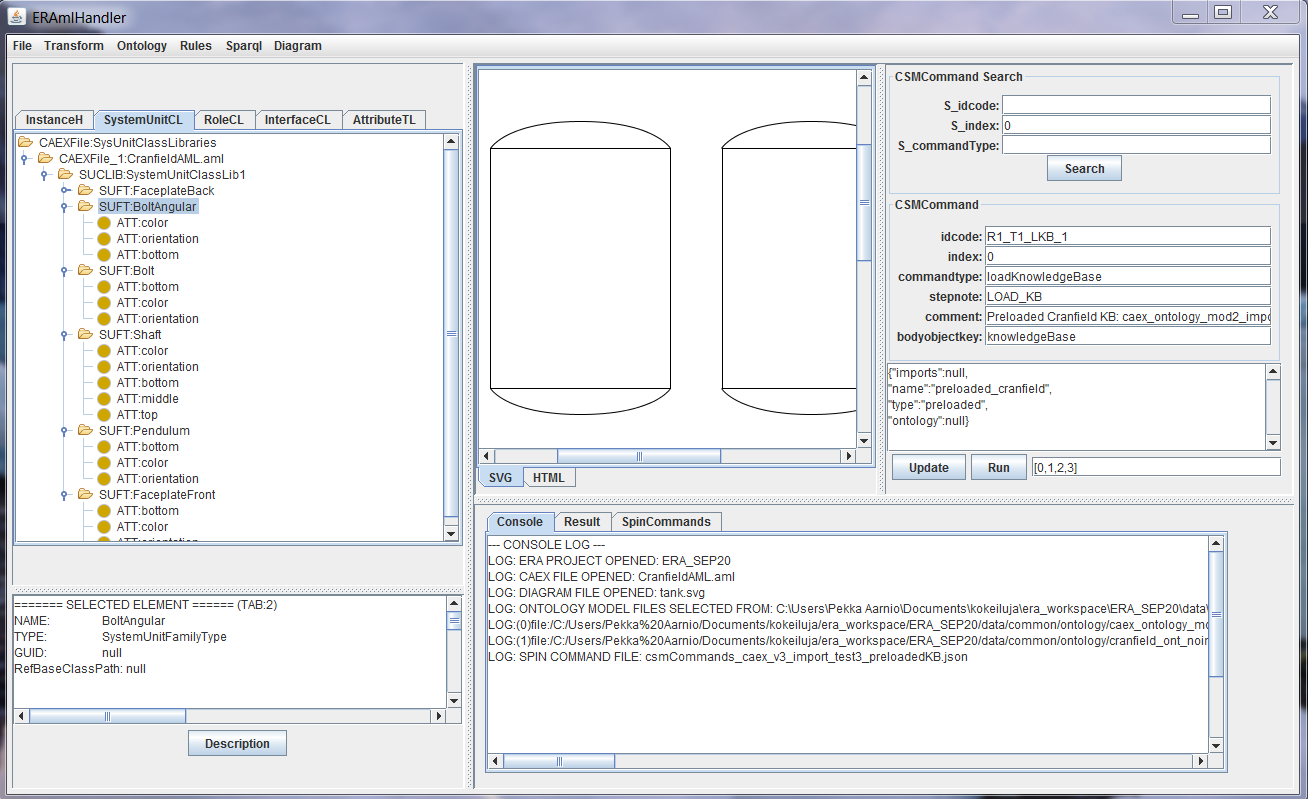
[Loading Rules and Facts 28](#_Toc8749834)

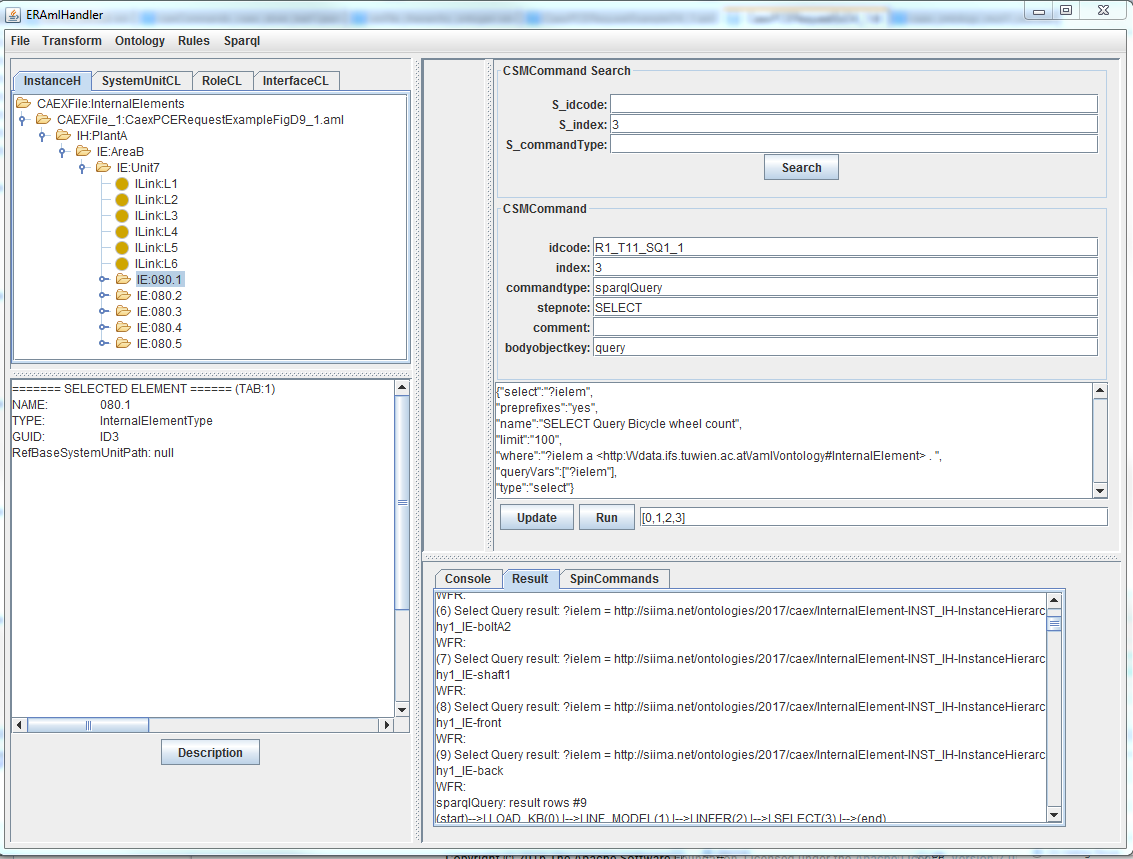
[Running ASP reasoning and saving the result model 28](#_Toc8749835)

[ANNEX A: SPARQL & SPIN CSMCommands 30](#_Toc8749836)

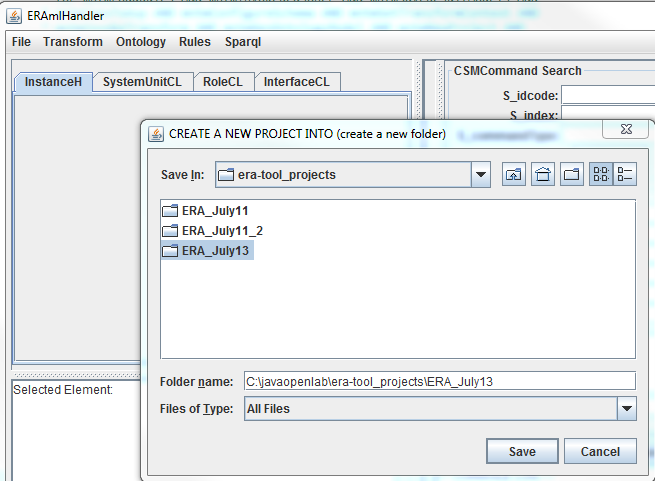
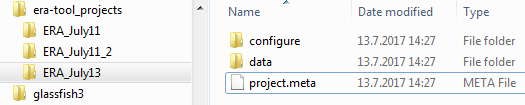
[Command type: ‘*addNsPrefixes’* 30](#_Toc8749837)

# Introduction



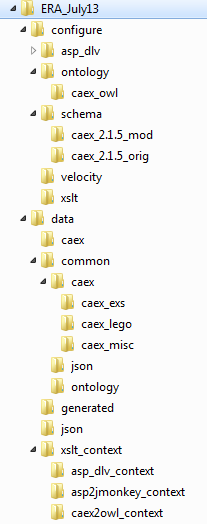


# Creating a new ERA Project

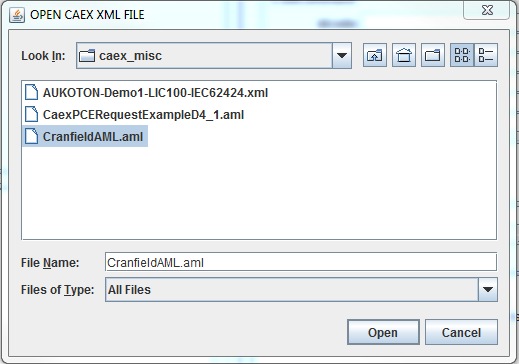
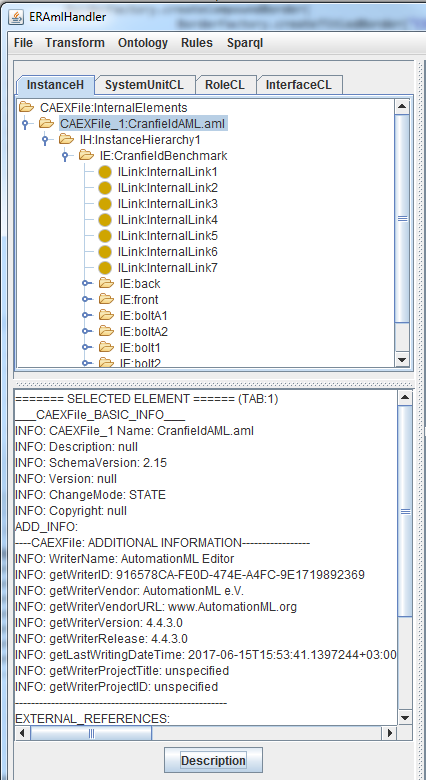


*project.meta* content:

PROJECT NAME: ERA\_July13

CREATED: 2017-07-13 14:27

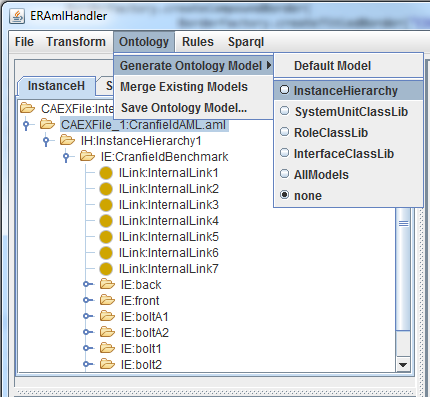
# Loading CAEX Files



# Generating CAEX Ontology Models

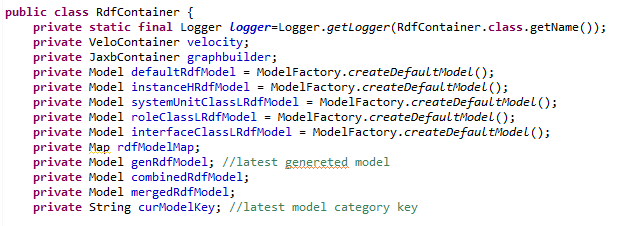
CAEX ontology models are generated from the last loaded CAEX JAXB object model. Four different partial models can be generated.

PARTIAL RDF MODELS:

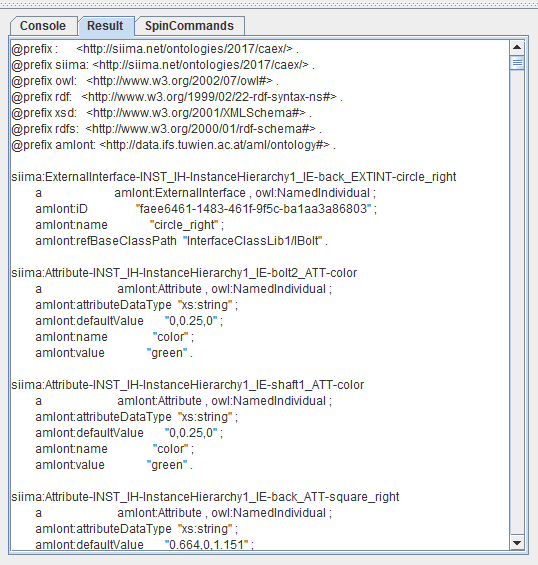
1. IHO: InstanceHIerarchy ontology model
2. SUCLO: SystemUnitClassLib ontology model
3. RCLO: RoleClassLib ontology model
4. ICLO: InterfaceClassLib ontology model

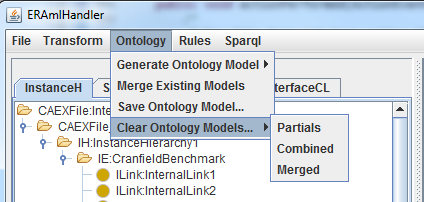
For instance, loaded *CranfieldAML.aml* file contains only an IH. By selecting *InstanceHIerarchy* RadioButton from the *Menu/Ontology/Generated Ontology Model/ InstanceHIerarchy* IHO model is generated. If the loaded CAEX file contains more partial xml models, ontology models can be generated from all of them by selecting *AllModels* from the menu list.

### RDFContainer Class

RDFContainer class provides the methods for ontology generation and management of all Jena RDF Models.

Java Class *RdfContainer* maintains eight (8) different jena rdf models. RDF Models are generated by Velocity engine into a temporary intermediate model *genRdfModel*, which is then loaded into one of the selected partial models (overdriving previously generated ones). Default option is for tool testing purposes and it currently generates an IHO model (same as InstanceHierarchy option). If *AllModels* option is selected, all the partial CAEX xml models contained in the latest loaded CAEX file are transformed (generated) as partial RDFmodels and combined into a RDF model called *combinedRdfModel*.

1. *genRdfModel* temporary model contains the latest generated partial model
2. *combinedRdfModel* contains all the models generated from last loaded CAEX XML file
3. *mergedRdfModel* merges all of the partial models and the combined models, when menu Merge Existing Models is selected



### Merging RDF partial models

All the existing partial models and the combined model can be merged into *mergedRdfModel*. After the first merge operation and clearing the existing partial models, a new CAEX file can be loaded, new partial models generated and finally merged to the existing *mergedRdfModel*. Using these operations any combination of different partial XML models from several CAEX files can be transformed into ontology format and saved as a single owl ontology model (*mergedRdfModel*). Notice that this merged model can be later directly loaded into the Knowledge base for SPARQL queries. This can be done from *MENU:Sparql/Preload Merged Ontology into KB*.

### Saving ontology models

The generated target RDF model of the last ontology operation can be saved into a file. Note: this means that if the merged model needs to be saved ‘*Merge Existing Models’* has to be the last Ontology Operation.

Ontology menu contains *Save Ontology Model* operation, which opens a SAVE FILE dialog. The ontology serialization format is defined based on the file name extension. If the extension is **.owl** *RDF/XML* serialization is used; if it is ***.ttl*** TURTLE serialization is selected.

1. RDF/XML serialization with *.owl* suffix
2. TURTLE serialization with *.ttl* suffix

# SPARQL and SPIN CSMCommands and Knowledge Base loading

SPARQL queries and SPIN rules can be executed against ontology models in system knowledge base (KB). A set of operations needs to be run before this is possible. For instance, first ontology models need to be loaded into KB and SPIN rules initialized before SPARQL queries and SPIN rules can be executed. This workflow of operations and queries is defined in a specific *CSMCommand* script file (json) written for the target ontology model. After loading this SPIN commands workflow script file (MENU: Sparql/Load Spin Commands…) its full content is displayed in SpinCommands tab of the bottom right window. CSMCommand window on the right enables to search and edit commands. A workflow consists of a sequence of commands specified as a sequence of command index numbers in the sequence edit field (beside Run button).

(SPARQL and SPIN operations are managed by *CommandFileSpinMng* java class implemented by classes in siima.spin package; modified from *ContextMngBySpin* project)

1. Example Case1: Loading one ontology file into the knowledge base
2. Example Case2: Loading the main ontology with imports and editing the commands CAEXCranfield
3. Example Case3: Preloading the KB from ontology files
4. Example Case 4: Preloading directly from Merged CAEX ontology (See era\_demo\_steps\_2018.zip)

## Example Case1: Loading one ontology file into the knowledge base

In this example, the CSMCommand file defines a Knowledge base to be uploaded with one ontology model only without any imports.

### STEP 1: Loading Sparql&Spin Commands file

Loading Spin Commands file: *csmCommands\_caex\_store\_new\_v3.json*

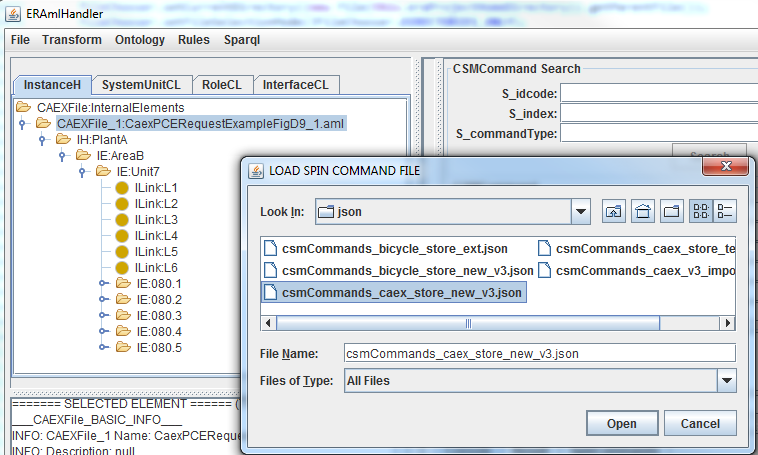


Figure. Loading Spin Commands file: csmCommands\_caex\_store\_new\_v3.json

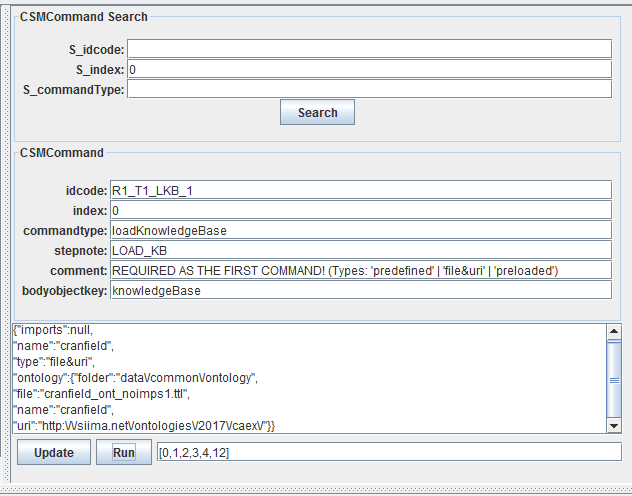


Figure. Search results of the first (index=0) CSMCommand with commandtype: “loadKnowledgeBase”

The knowledge base loading command (type loadKnowledgeBase) contains a knowledgebase object with type “file&uri”, which requires that the “ontology” object contains the path to the folder (relative to the eclipse project) containing the main ontology file. In this case it is *cranfield\_ont\_noimps1.ttl* with uri: *http://siima.net/ontologies/2017/caex/*. In this case, the imports object is *null* which means that no other files are loaded.

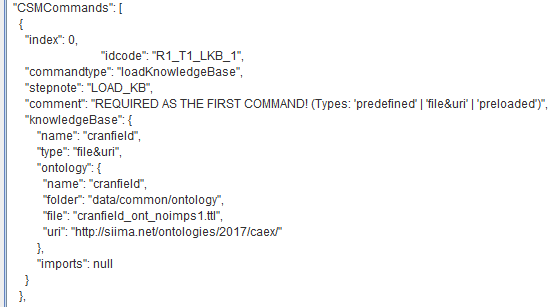


Figure. The description of the Knowledge base loading command.

### STEP2 Searching and displaying the description of the load knowledge base command with index 0

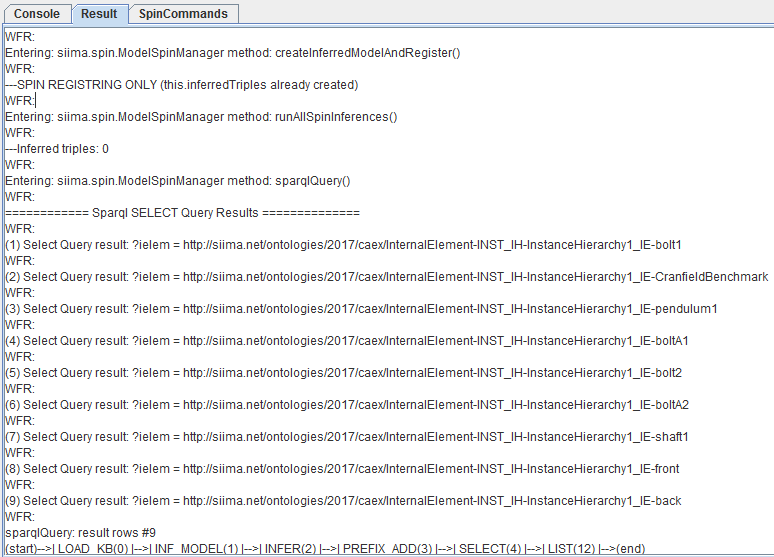
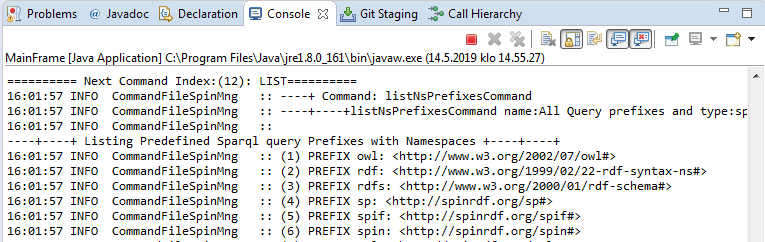
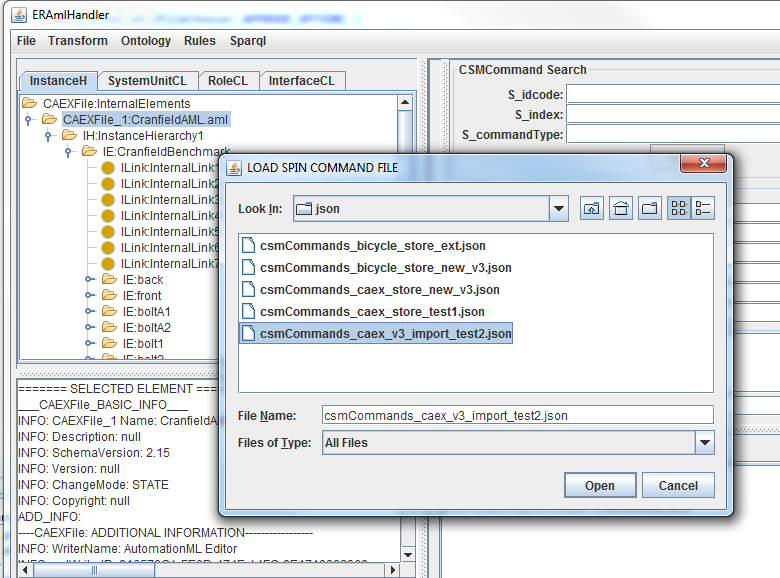


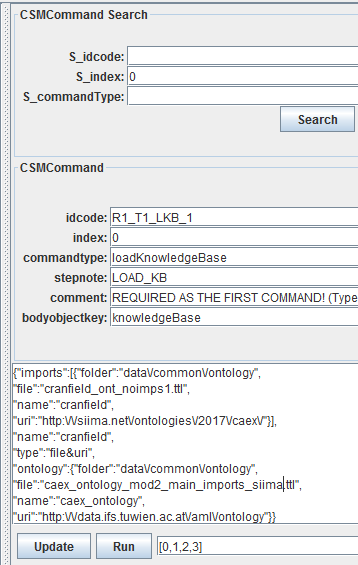
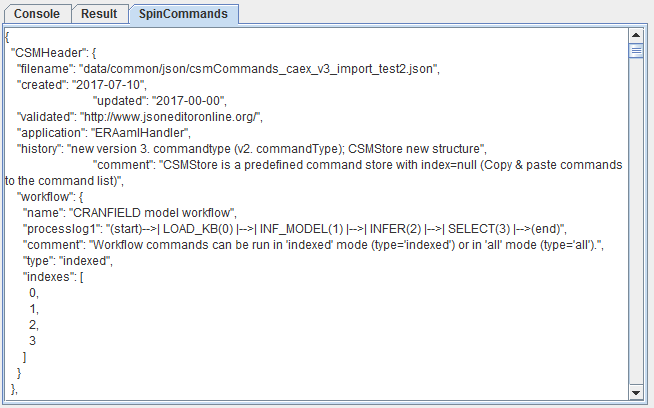
Figure. Result display after running the command sequence [0,1,2,3,4,12]

Figure. The results of the command ‘*listNsPrefixesCommand’* are listed on Eclipse console only.

## Example Case2: Loading the main ontology with imports and editing the commands CAEXCranfield

### A: Editing and running a command sequence

1. Loading CAEX XML file *CranfieldAML.aml* just to see its *InstanceHierarchy* content
2. Loading Spin Commands file: *csmCommands\_caex\_v3\_import\_test2.json*
3. Searching CSMCommand with *s\_index=0* into the command edit area (CSMCommand fields). This command will load an ontology from a common ontology folder of the project (previously generated from CranfieldAML.aml XML model) into the knowledge base (KB)
4. Editing the command workflow, i.e. command sequence shown in the textfield beside the RUN button by removing the last index 3 from the array [0,1,2,3] -> [0,1,2]
5. Running the command workflow by pushing RUN button.
6. Selecting the command 4 (Searching with *s\_index = 4*) to the edit fields and examining it (SPARQL Select query)
7. Editing the sequence by extending it [0,1,2] -> [0,1,2,3,4] and running the workflow again (RUN)
8. The Select query returns nine (9) individuals of the queried #*InternalElement* class ("where":"?ielem a *<http:\/\/data.ifs.tuwien.ac.at\/aml\/ontology#InternalElement*>)



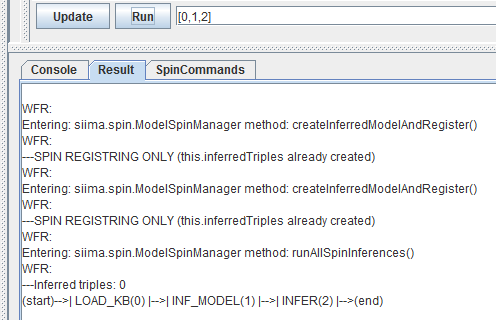
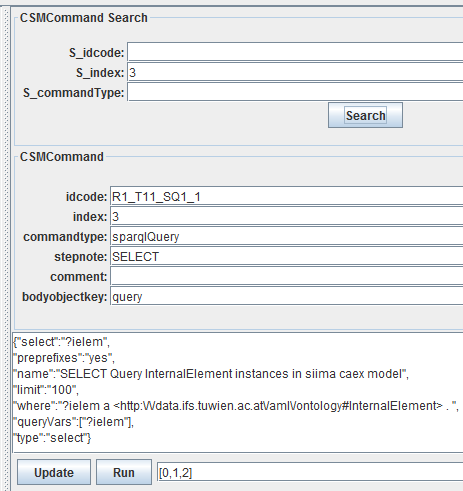


Figure. Steps A3-A5

4

Figure. Step A6



4

4

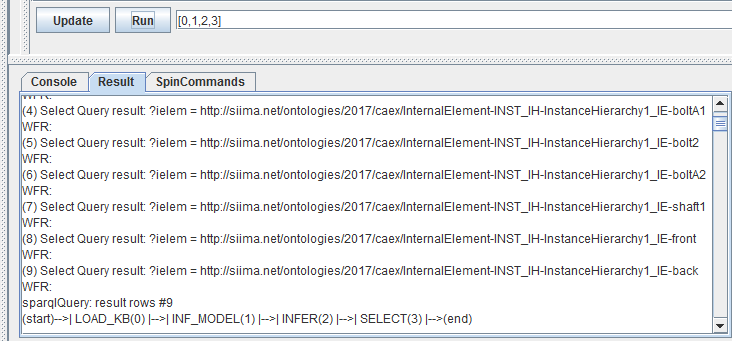


Figure. Steps A7-A8

### B: Copying a command into CSMStore as a new command example

1. Editing the *CSMCommand* by setting its *index=null* and *idcode=SR1\_T11\_SQ1\_555* (note S prefix and 555)
2. Pushing the *UPDATE* Button.
3. CSMCommands with **index=null** are copied to *CSMStore* list (i.e. *CSMCommand* with index=3 is not changed or updated)
4. Now it can be found from the updated *SpinCommands Tab* below as the last command in *CSMStore* array.

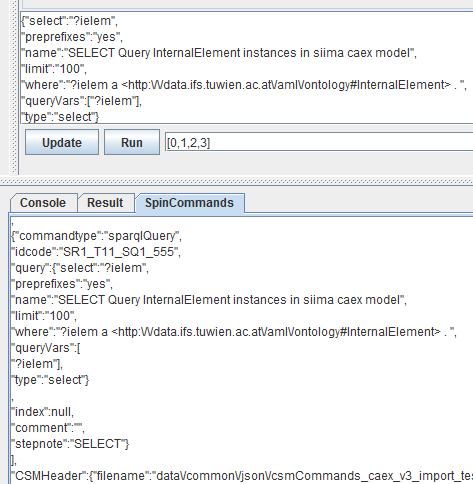


Figure. Step B4

### C: Changing the order of commands in a workflow

1. Searching the original command with index=3 reviels that the original has not been changed in the previous update (with index set to null)
2. Again editing the selected *CSMCommand* by setting its *index=4* and *idcode=R1\_T11\_SQ1\_444* (note 444)
3. Pushing the *UPDATE* Button.
4. CSMCommands with index changed to some new integer value (**index=4)** will be inserted into *CSMCommands Array* into that index position and the following command indexes will be increased by one.
5. Now it can be found from the updated *SpinCommands Tab* below as the fifth (indexes begin with number 0) command in *CSMCommands* array.

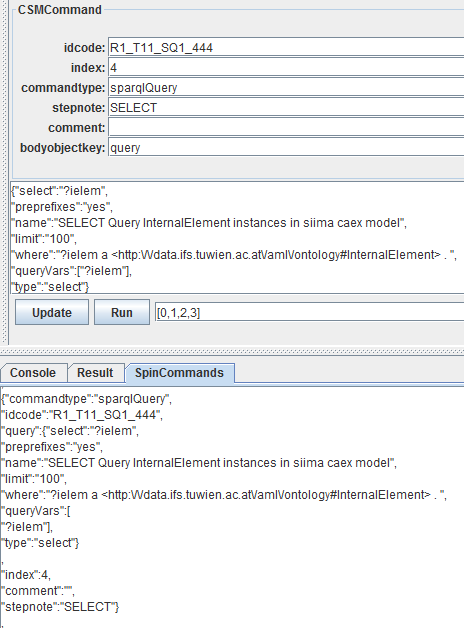
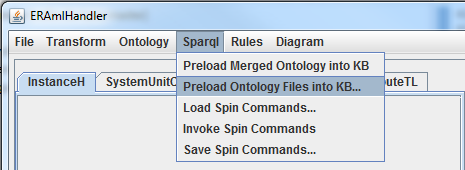


Figure. Steps C1-C5

## Example Case3: Preloading the KB from ontology files

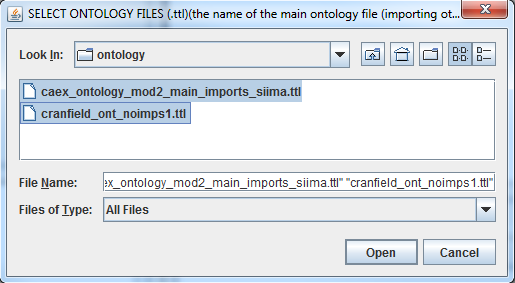
### STEP 1: Preload Knowledge Base using: Menu/Sparql/Preload Ontology Files into KB…

Loading CSM knowledge base (KB) from ontology files (.ttl) in advance, before running CSMCommands workflow (.json). In this case, the first CSMCommand (index 0) *loadKnowledgeBase* must set its *knowledgeBase* type as ‘*preloaded’*.

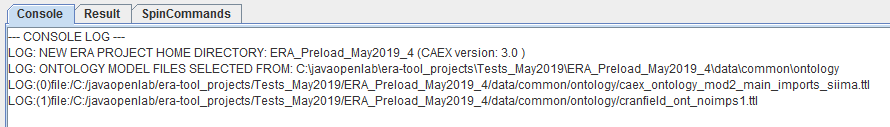
*Menu/Sparql/Preload Ontology Files into KB…*

### STEP 2: Navigate to Ontology folder and select two ontology files (using +ctrl key):

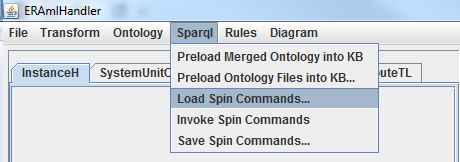
The name of the main ontology file (importing others) should contain a substring '\_main' OR '\_imports' (or be the first in the defined folder file ordering, e.g. alphabetical or date). Note: press +ctrl to select multiple files.

1. The main ontology file with import statements: *caex\_ontology\_mod2\_main\_imports\_siima.ttl*,
2. The imported ontologies: *cranfield\_ont\_noimps1.ttl*

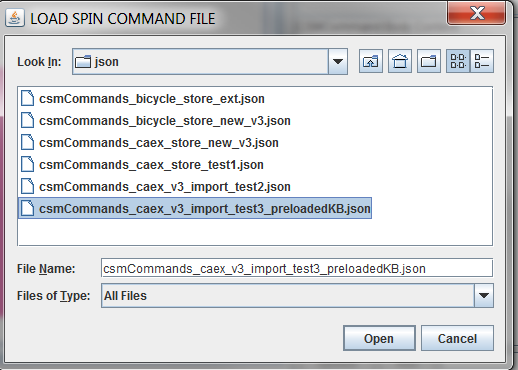
CONSOLE TAB:

Console view displays the URLs of the loaded ontologies in URI format (Notice %20 for space char *file:/C:/ Users/Pekka%20Aarnio/*)

### STEP 3: Load Spin Command File (.json)



*Figure. Menu/Sparql/Load Spin Commands File*



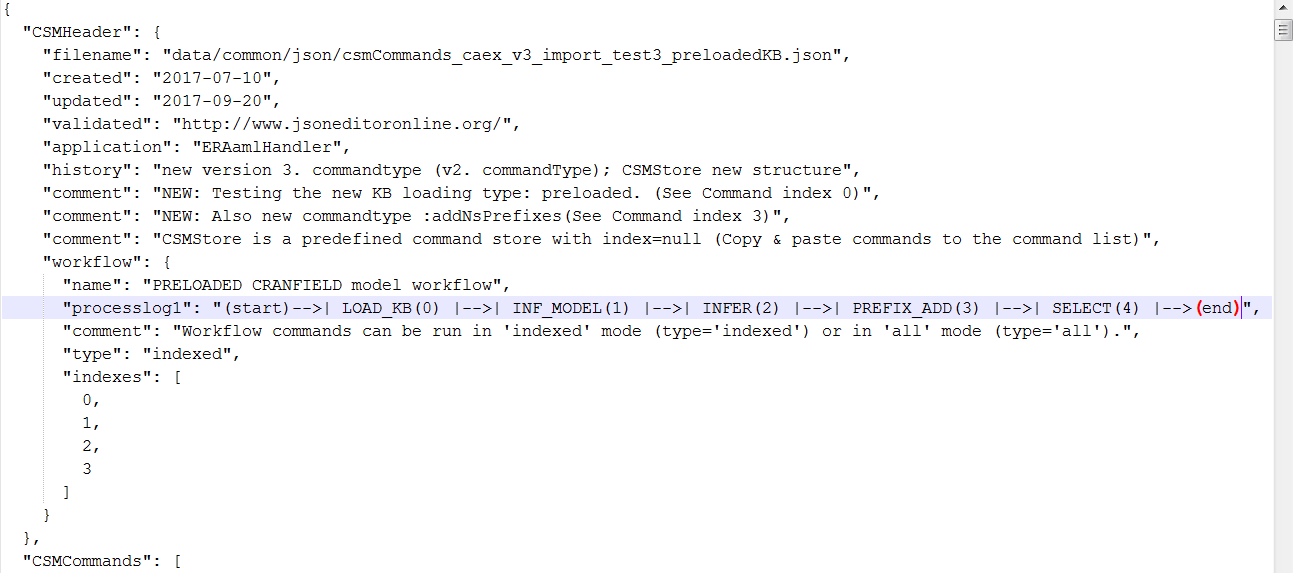
Figure *csmCommands\_caex\_v3\_import\_test3\_preloadedKB.json*

Figure. Header info of a CSMCommand file

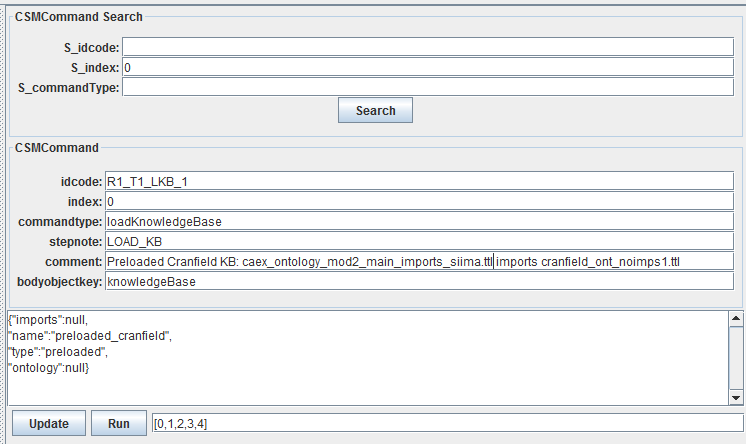


Figure. Notice: *knowledgeBase ‘type’:’preloaded’*

------------------

"CSMCommands": [

{

"index": 0,

"idcode": "R1\_T1\_LKB\_1",

"commandtype": "loadKnowledgeBase",

"stepnote": "LOAD\_KB",

"comment": "REQUIRED AS THE FIRST COMMAND! (Types: 'predefined' | 'file&uri' | 'preloaded')",

"comment": "Preloaded Cranfield KB: caex\_ontology\_mod2\_main\_imports\_siima.ttl imports cranfield\_ont\_noimps1.ttl",

"knowledgeBase": {

"name": "preloaded\_cranfield",

"type": "preloaded",

"ontology": null,

"imports": null

}

},

-------------------

Listing. CSMCommand with Knowledge Base loading type: ‘*preloaded’*:

## Example Case 4: Preloading directly from Merged CAEX ontology

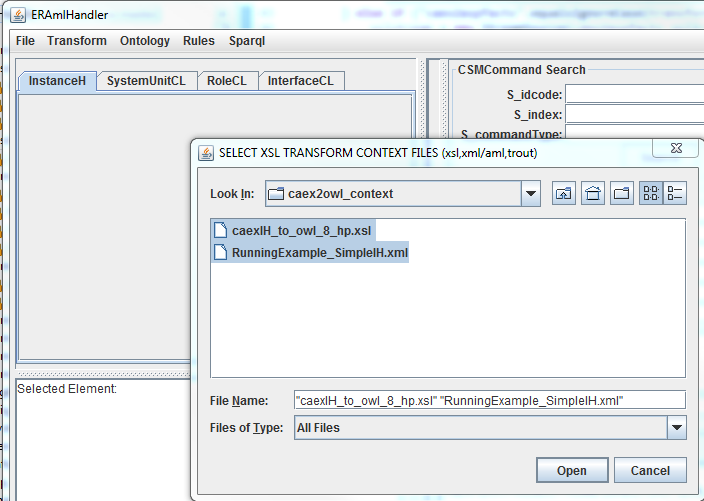
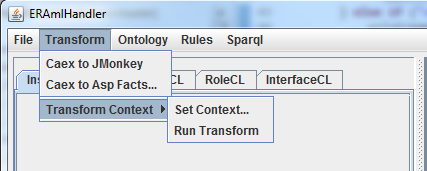
(See era\_demo\_steps\_2018.zip)

# XSL Transforms

## Generic XSL transforms

First we need to define the transformation context files, i.e. the source xml file to be transformed (.xml/aml) and the xsl stylesheet file (.xml) and optionally also the file, where the transformation result/output should be saved (.trout) (if this output file is not defined, the name of the output file is generated from the source file name with a suffix \_*trout.trout*)

1. Click ***Set Context..*** from the menu *Transform/Transform Context/Set Context* …
2. When the Select Context dialog opens, browse the project directories and select the context files
3. All context files can be selected at once or by three different *Set Context* steps
4. When at least source xml file and xsl file has been defined for the context the actual transformation operation can be invoked from the menu: *Transform/Transform Context/****Run Transform***



LOG:

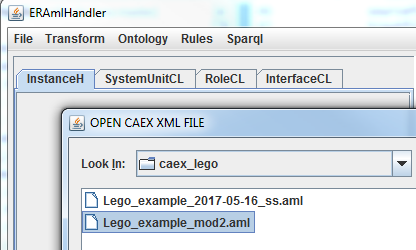
*12:01:41 INFO XSLTransform :: doSpecificTransform: Transformation result file (.trout) saved: C:\javaopenlab\era-tool\_projects\ERA\_Aug28\data\xslt\_context\caex2owl\_context\****RunningExample\_SimpleIH\_trout.trout***

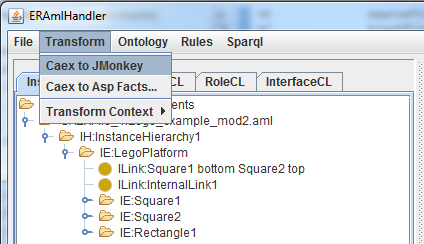
## Pre-specified XSL transforms

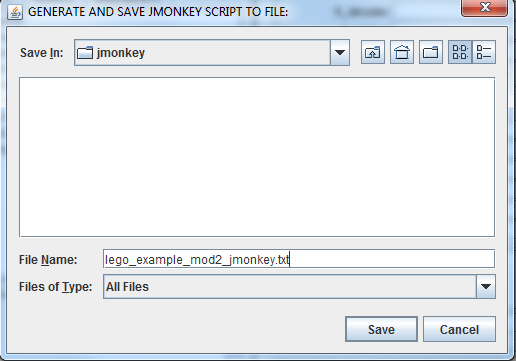
1. Menu/Transform/Caex to JMonkey
2. Menu/Transform/Caex to ASP facts

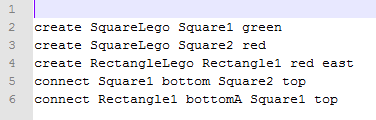
### Caex to JMonkey transformation

This pre-specified xsl transformation parses the opened Caex model and generates a command script file for SS-jmonkey 3D simulation application.



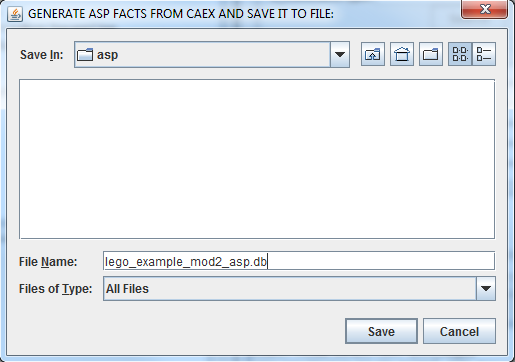


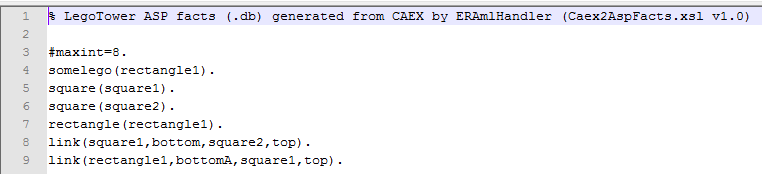


lego\_example\_mod2\_jmonkey.txt

### Caex to ASP facts transformation

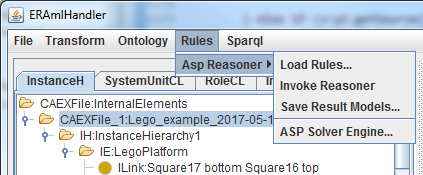
This pre-specified xsl transformation parses the opened Caex model and generates a ASP (Answer Set Programming) fact file (.db) for rule based reasoning ASP application (.dlv).



lego\_example\_mod2\_asp.db

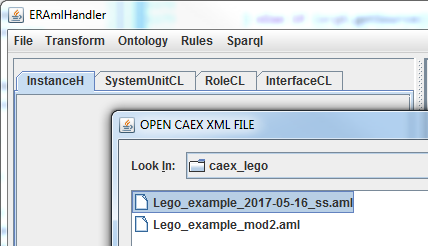
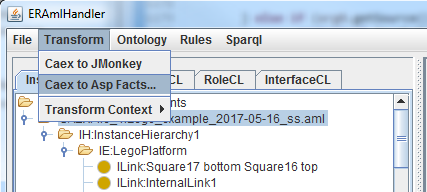
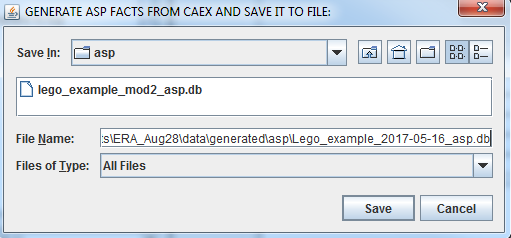
# ASP Rules

Answer Set Programming (ASP) rules can be invoked via Rules menu. (See ERA project: C:\javaopenlab\era-tool\_projects\ERA\_Aug28)

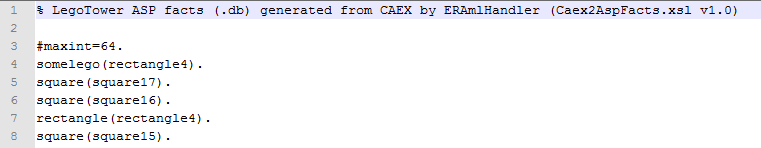


### Generating ASP fact file

1. Open CAEX file*: Lego\_example\_2017-05-16\_ss.aml*
2. Transform CAEX model to ASP facts file: *Lego\_example\_2017-05-16\_asp.db*

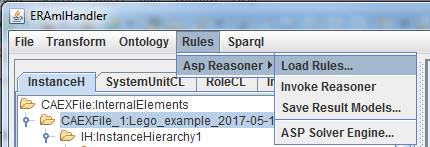


*data/generated/asp/Lego\_example\_2017-05-16\_asp.db*



### Loading Rules and Facts

1. First loading rules file: *Menu/Rules/ASP Reasoner/Load Rules…*(.dlv file: configure/asp\_dlv/legotower/legotower\_rules\_mod7.dlv)
2. Next loading the generated facts file: *Menu/Rules/ASP Reasoner/Load Rules…* (.db file: data/generated/asp/Lego\_example\_2017-05-16\_asp.db)



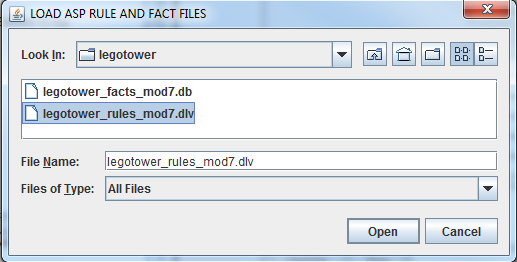
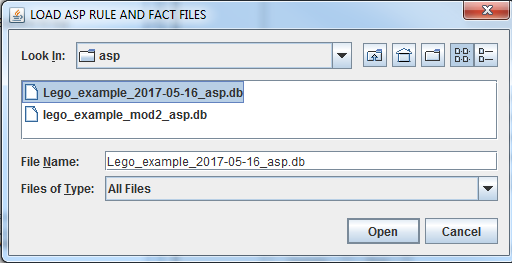


Figure. Loading rules and facts files

### Running ASP reasoning and saving the result model

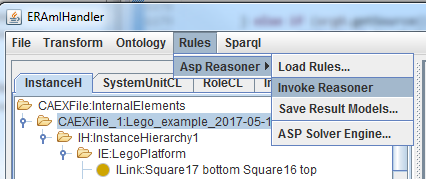
1. Invoking/running ASP reasoning: reasoning results displayed in *Results-tab*
2. Saving reasoning result model into a file in a specific xml format.

Figure. Invoking ASP reasoner

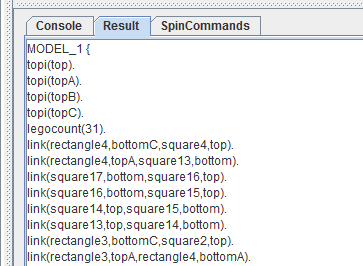
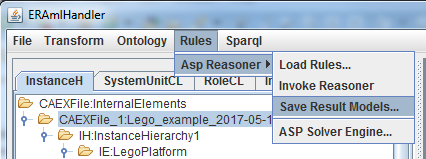


Figure. ASP reasoning results displayed in *Results-tab* in ASP format.



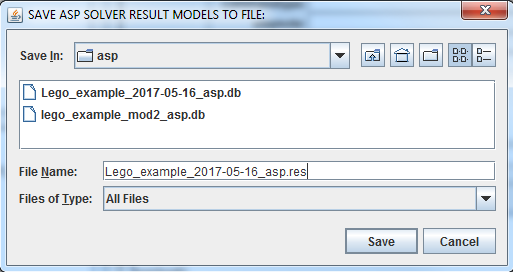


Figure. Saving the results (.res) in XML format

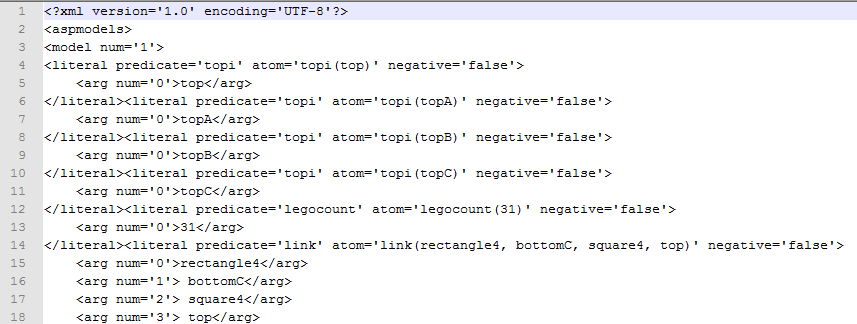


Figure. Reasoning results file (.res) in XML format *Lego\_example\_2017-05-16\_asp.res*

# ANNEX A: SPARQL & SPIN CSMCommands

### Command type: ‘*addNsPrefixes’*

------

{

"index": 3,

"idcode": "R1\_T66\_QP1\_1",

"commandtype": "addNsPrefixes",

"stepnote": "PREFIX\_ADD",

"comment": "Adding NS prefix declarations (separated by space or newline) in format: PREFIX owl: <http://www.w3.org/2002/07/owl#> ",

"comment": "Note: preprefixes field is currently ignored, i.e. hard coded preprefixes are always loaded also",

"prefix": {

"name": "NS PREFIX Addition ",

"type": "add",

"preprefixes": "default",

"prefixlist": "PREFIX caex: <http://data.ifs.tuwien.ac.at/aml/ontology#> PREFIX foo: <http://foo.net/ontology#> PREFIX bar: <http://bar.net/ontology#> "

}

},

------

