IFC-to-RDF Converter User manual (version 1.5)

Document history

Version	Date	Action	Modifier
1.0	22.10.2013	Created document	Nam Vu Hoang
1.5	13.03.2015	Updated	Nam Vu Hoang

Table of Contents

1 General information

1.1 The program purpose

IFC-to-RDF software is a configurable Java program with open API used for:

- Parsing IFC schema defined in EXPRESS format (ISO 10303-11:2004[1]);
- Parsing IFC model defined in STEP-File format (ISO_10303-21:2002[2]);
- Exporting IFC schema and model to Jena models[3] which can be later exported to triple stores or text files in different RDF formats[4] (e.g. RDF/XML, N3 Turtle);
- Importing IFC models from Jena models.

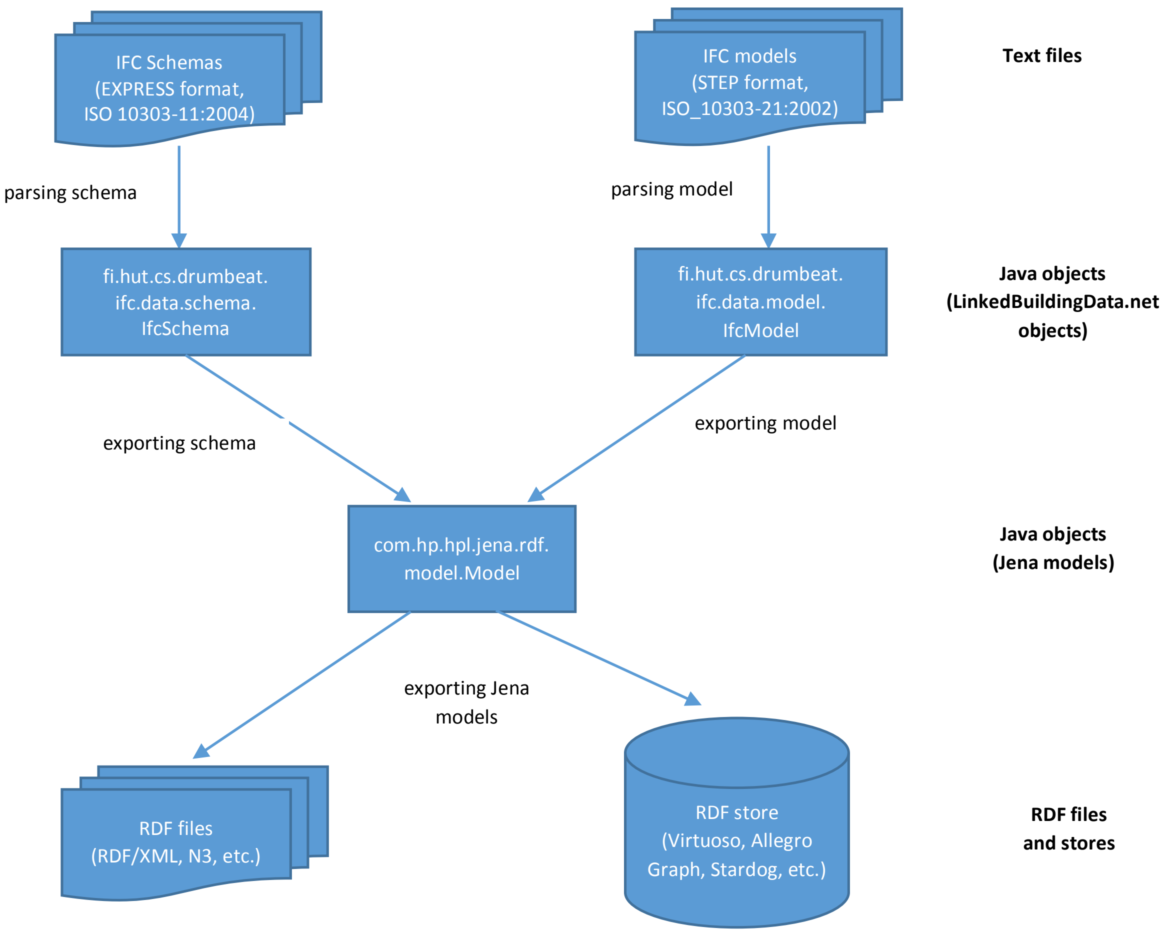
IFC Schemas (EXPRESS format, ISO 10303-11:2004)
IFC models (STEP format, ISO_10303-21:2002)
Text files
parsing schema
parsing model
fi.hut.cs.drumbeat. ifc.data.schema. IfcSchema
fi.hut.cs.drumbeat. ifc.data.model. IfcModel
Java objects (LinkedBuildingData.net objects)
exporting schema
exporting model
com.hp.hpl.jena.rdf. model.Model
Java objects (Jena models)
exporting Jena models
RDF files (RDF/XML, N3, etc.)
RDF store (Virtuoso, Allegro Graph, Stardog, etc.)
RDF files and stores

Figure 1. IFC-to-RDF software functionalities

1.2 System requirements

- Windows or Unix-like OS
- Java Platform, Second Edition, version 7 (Java SE 7)[5]: JRE is needed for using the program in command line, JDK is required for using the program API.

[1] http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=38047
[2] http://en.wikipedia.org/wiki/ISO_10303-21
[3] http://jena.apache.org/documentation/javadoc/jena/com/hp/hpl/jena/rdf/model/Model.html
[4] http://www.w3.org/2008/01/rdf-media-types
[5] http://www.oracle.com/technetwork/java/javase/downloads/index.html

- The release package already includes adapters for such triple stores as OpenLink Virtuoso[6], Allegro Graph[7], Stardog[8]. In case of exporting to another triple store, you have to implement your own adapter (see below).

1.3 The package content

The release package includes the following folders and subfolders:

• bin	executable script files for Unix and Windows
• config	IFC-2-RDF configuration and logging configuration files
• docs	user manuals
• lib	Java archive (JAR) files
○ agraph	JAR files for AllegroGraph API
○ virtuoso	JAR files for Virtuoso API
• resources	EXPRESS schemas of IFC2x3 TC1[9] and IFC4[10]
• samples	sample files
○ java	sample Java code
○ scripts	sample shell (for Unix) and batch files (for Windows)

1.4 Copyright and feedback

The IFC-to-RDF converter was created by Nam Vu Hoang at Department of Computer Science and Engineering of Aalto University, School of Science (Helsinki, Finland), under the guidance of D.Sc. (Tech.) Seppo Törmä in the DRUM research effort (Distributed Transactional Building Information Management, 2010-2014) belonging to the PRE project of RYM Oy and funded by Tekes.

The software package is freeware and is used “as is”. The authors do not take any responsibility for its correctness. For any suggestions, feedbacks or requests, please contact us:

- Seppo Törmä, seppo.torma@aalto.fi
- Nam Vu Hoang, nam.vuhoang@aalto.fi.

[6] http://virtuoso.openlinksw.com/
[7] http://www.franz.com/agraph/allegrograph/
[8] http://stardog.com/
[9] http://www.buildingsmart-tech.org/specifications/ifc-releases/ifc2x3-tc1-release
[10] http://www.buildingsmart-tech.org/specifications/ifc-releases/ifc4-release

2 Using IFC-to-RDF Converter as an executable program

2.1 Command line

In Windows:

```
bin/export-ifc2rdf.bat -lcf <file> -cf <file> -isf <file> [-imf <file>]
[-osn <name> | -osf <file>]  [-omn <name> | -omf <file>]  [-ofl <name>]
```

In Unix:

```
bash bin/export-ifc2rdf.sh -lcf <file> -cf <file> -isf <file> [-imf <file>]
[-osn <name> | -osf <file>]  [-omn <name> | -omf <file>]  [-ofl <name>]
```

Where:

-lcf,--log-config-file <file>	*Logger configuration file (.properties or .xml)*
-cf,--config <file>	*Configuration file*
-oln,--output-layer-name <file	*Layer name*
-isf,--input-schema-file <file>	*Source IFC schema file or folder*
-imf,--input-model-file <file>	*Source IFC model file*
-osn,--output-schema-name <name>	*Target RDF store for the ontology (optional)*
-osf,--output-schema-file <file>	*Target RDF file for the ontology (optional)*
-omn,--output-model-name <name>	*Target RDF store for the model (optional)*
-omf,--output-model-file <file>	*Target RDF file for the model (optional)*
-off,--output-file-format <name>	*RDF format (add suffix .GZ to gzip file), e.g.: TURTLE (default), TURTLE.GZ, NTRIPLES, NQUADS, RDFXML_ABBREV, RDFXML_PRETTY, JSONLD, etc. (see more: https://jena.apache.org/documentation/javadoc/arq/org/apache/jena/riot/RDFFormat.html)*

For example:

```
bash bin/export-ifc2rdf.sh -lcf config/log4j.xml -cf config/IfcAnalysis.xml
-isf resources/IFC2X3_TC1.exp -imf samples/sample.ifc -omf output/sample.rdf -off TURTLE
–oln Standard
```

2.2 Sample script files

In folder *samples/scripts* of the release package there are some sample script files*:*

- *export-help (.sh or .bat)*
- *export-model-to-file-001 (.sh or .bat)*
- *export-schema-2x3-to-agraph-on-allegrograph.net (.sh or .bat)*
- *export-schema-2x3-to-agraph-on-murskain (.sh or .bat)*
- *export-schema-2x3-to-file (.sh or .bat)*
- *export-schema-2x3-to-stardog-on-localhost (.sh or .bat)*

- *export-schema-2x3-to-virtuoso-on-localhost (.sh or .bat)*
- *export-schema-4-to-file (.sh or .bat)*

It is easy to understand what these scripts do from their names. Every script is available in two versions: batch file for the Windows-family OS and shell file for Unix-like systems.

3 Using IFC-to-RDF Converter in your Java program

The following JAR files must be included in the Java class path:

commons-codec-1.6.jar	Apache Commons Codec API (general encoding/decoding algorithms)
commons-lang3-3.1.jar	Apache Commons Lang API (extra functionality for classes in java.lang)
jena-arq-2.9.2.jar	Apache Jena ARQ API (A SPARQL processor for Jena)
jena-core-2.7.2.jar	Apache Jena Core API (Core RDF API)
jena-iri-0.9.2.jar	Apache Jena IRI API (supports for International Resource Identifiers in Jena)
log4j-1.2.16.jar	Apache Log4J API (supports logging)
fi.hut.cs.drumbeat.common-1.0.jar	LinkedBuildingData.net Common API (common utilities)
fi.hut.cs.drumbeat.ifc.common-1.0.jar	LinkedBuildingData.net IFC Common API (common utilities for IFC)
fi.hut.cs.drumbeat.ifc.convert.ifc2rdf-1.0.jar	LinkedBuildingData.net IFC-2-RDF API (conversion from in-memory IFC object model to Jena model)
fi.hut.cs.drumbeat.ifc.convert.step2ifc-1.0.jar	LinkedBuildingData.net STEP-2-IFC API (conversion from STEP-file to in-memory IFC object model)
fi.hut.cs.drumbeat.ifc.data-1.0.jar	LinkedBuildingData.net IFC Data API (in-memory IFC object schema and model structure)
fi.hut.cs.drumbeat.ifc.util-1.0.jar	LinkedBuildingData.net IFC Utils API (high-level specific IFC utilities)
fi.hut.cs.drumbeat.rdf-1.0.jar	LinkedBuildingData.net RDF API (RDF utilities)
fi.hut.cs.drumbeat.rdf.modelfactory-1.0.jar	LinkedBuildingData.net Jena Model Factory API (base class for Jena model factory)
fi.hut.cs.drumbeat.rdf.modelfactory. allegrograph-1.0.jar (optional)	LinkedBuildingData.net Jena Model Factory API for Allegro Graph (creating and exporting Jena model to Allegro Graph)
fi.hut.cs.drumbeat.rdf.modelfactory. virtuoso-1.0.jar (optional)	LinkedBuildingData.net Jena Model Factory API for Virtuoso (creating and exporting Jena model to Virtuoso)
fi.hut.cs.drumbeat.rdf.modelfactory. stardog-1.0.jar (optional)	LinkedBuildingData.net Jena Model Factory API for Stardog (creating and exporting Jena model to Stardog)
slf4j-api-1.6.4.jar	Simple Logging Facade for Java (SLF4J) API
slf4j-log4j12-1.6.4.jar	SLF4J – Log4J bridge API
xercesImpl-2.10.0.jar	Xerces2 Java Parser API
xml-apis-1.4.01.jar	XML APIs

For more information, see the JavaDoc of Ifc2RdfExporter software package (will be published later).

Please find a Java code sample in the release package:
samples/java/net/linkedbuildingdata/ifc/convert/ifc2rdf/cli/Ifc2RdfExporter.java

3.1 Loading IFC schema file

In order to import an IFC schema defined EXPRESS format file you must use the class *ParserUtil* as below (see method *parseSchema()* in the sample file *Ifc2RdfExporter.java*):

```
import fi.hut.cs.drumbeat.ifc.convert.text2ifc.util.ParserUtil;
import fi.hut.cs.drumbeat.ifc.data.schema.IfcSchema;
...
IfcSchema schema = ParserUtil.parseSchema(filePath);

// or
List<IfcSchema> schemas = ParserUtil.parseSchemas(folderPath);
```

IfcSchema is a Java class which contains all information about entity, select, enumeration, or defined data types of the EXPRESS[11] schema. Once the schema is imported successfully, it will be added to the schema pool automatically. To get a schema from that pool:

```
import fi.hut.cs.drumbeat.ifc.data.schema.IfcSchema;
import fi.hut.cs.drumbeat.ifc.data.schema.IfcSchemaPool;
...
IfcSchema schema = IfcSchemaPool.getSchema(schemaVersion);
```

3.2 Parsing IFC model file

In order to import an IFC model defined STEP file you must use the class ParserUtil (see method *parseModel()* in the sample file *Ifc2RdfExporter.java*):

```
import fi.hut.cs.drumbeat.ifc.convert.text2ifc.util.ParserUtil;
import fi.hut.cs.drumbeat.ifc.data.model.IfcModel;
...
IfcModel model = ParserUtil.parseModel(filePath);
```

An *IfcModel* contains all entities (*IfcEntity*) which has literal attributes, outgoing and incoming links with other entities.

3.3 Converting IFC schema to Jena model

Use the class *Ifc2RdfExportUtil* as below (see method *exportSchema()* in the sample file *Ifc2RdfExporter.java*):

```
import fi.hut.cs.drumbeat.ifc.convert.ifc2rdf.util.Ifc2RdfExportUtil;
import fi.hut.cs.drumbeat.ifc.convert.ifc2rdf.Ifc2RdfConversionContext;
import fi.hut.cs.drumbeat.ifc.data.schema.IfcSchema;
import fi.hut.cs.drumbeat.ifc.util.ifc2rdf
import com.hp.hpl.jena.rdf.model.Model;
...
```

[11] http://en.wikipedia.org/wiki/EXPRESS_(data_modeling_language)

```
Ifc2RdfExportUtil.exportSchemaToJenaModel(jenaModel, ifcSchema, conversionContext);
```

Parameter *jenaModel* has type *com.hp.hpl.jena.rdf.model.Model*[12] (see more in the documentation of Apache Jena[13]). The third parameter has type *Ifc2RdfConversionContext*. When it is null or skipped, the default context loaded from the configuration file is used (see paragraph 4).

3.4 Converting IFC model to Jena model

Use the class *Ifc2RdfExportUtil* as below (see method *exportModel()* in the sample file *Ifc2RdfExporter.java*):

```
import fi.hut.cs.drumbeat.ifc.convert.ifc2rdf.util.Ifc2RdfExportUtil;
import fi.hut.cs.drumbeat.ifc.convert.ifc2rdf.Ifc2RdfConversionContext;
import fi.hut.cs.drumbeat.ifc.data.model.IfcModel;
import fi.hut.cs.drumbeat.ifc.util.ifc2rdf
import com.hp.hpl.jena.rdf.model.Model;
...

Ifc2RdfExportUtil.exportModelToJenaModel(jenaModel, ifcModel, conversionContext);
```

The third parameter has type *Ifc2RdfConversionContext*. When it is null or skipped, the default context loaded from the configuration file is used (see paragraph 4).

3.5 Exporting Jena models to RDF-file or RDF store

You can create a *com.hp.hpl.jena.rdf.model.Model* instance either by your own way, or by using Jena model factories. Configuration of the latter is specified in the configuration document (see paragraph 4).

Find a Jena model factory by a name and then create a Jena model (see methods *run()*, *loadConfiguration()*, *getJenaModelFactoryConfiguration()* and *getJenaModelFactory()* in sample file *Ifc2RdfExporter.java*):

```
import fi.hut.cs.drumbeat.common.config.document.ConfigurationDocument;
import fi.hut.cs.drumbeat.common.config.ConfigurationPool;
import fi.hut.cs.drumbeat.common.config.ConfigurationItemEx;
import fi.hut.cs.drumbeat.rdf.modelfactory.config.JenaModelFactoryPoolConfigurationSection;
import fi.hut.cs.drumbeat.rdf.modelfactory.JenaModelFactoryBase;
import com.hp.hpl.jena.rdf.model.Model;
…

// load the configuration document
ConfigurationDocument.load(configFilePath);

// get a Jena model factory configuration pool
ConfigurationPool<ConfigurationItemEx> jenaModelFactoryConfigurationPool =
        JenaModelFactoryPoolConfigurationSection.getInstance().getConfigurationPool();

// get a Jena model factory configuration by name
ConfigurationItemEx jenaModelFactoryConfiguration =
        jenaModelFactoryConfigurationPool.getByName(jenaModelFactoryName);
```

[12] http://jena.apache.org/documentation/javadoc/jena/com/hp/hpl/jena/rdf/model/Model.html
[13] http://jena.apache.org/

```
// create a Jena model factory
JenaModelFactoryBase jenaModelFactory =
        JenaModelFactoryBase.getFactory(configuration);

Model jenaModel = outputSchemaJenaModelFactory.createModel()
```

As mentioned above the following methods export IFC schemas and models to Jena models. A Jena model can be in-memory or not. If the Jena model is not in-memory then newly added triples will be automatically exported to the real triple store.

Jena models can be exported to text files by using method *Model.write(outputStream, formatLanguage)* or *Model.write(writer, formatLanguage)*[14] where parameter *formatLanguage* defines the RDF text format, for example "RDF/XML", "N-TRIPLE", "TURTLE", "N3", etc. For example (see methods exportSchema() and exportModel() in sample file Ifc2RdfExporter.java):

```
import java.io.FileWriter;
import fi.hut.cs.drumbeat.common.file.FileManager;
…
FileWriter writer = FileManager.createFileWriter(filePath);
jenaModel.write(writer, "N-TRIPLE");
```

After using Jena model do not forget to release the Jena model factory:

```
import fi.hut.cs.drumbeat.rdf.modelfactory.JenaModelFactoryBase;
…
jenaModelFactory.release();
```

3.6 Creating user-defined Jena model factories

This release package includes Jena model factories for creating instances of classes that implement interface com.hp.hpl.jena.rdf.model[15], e.g.: default in-memory model class, com.franz.agraph.jena.AGModel[16], virtuoso.jena.driver.VirtModel[17], or Stardog Model[18]. If you want to create a new Jena model factory, you have to create a class which inherits from abstract class *fi.hut.cs.drumbeat.rdf.modelfactory.JenaModelFactoryBase* and implements the its abstract methods (see sample files in folder *samples/java/net/linkedbuildingdata/rdf/modelfactory : allegrograph/AGJenaModelFactory.java*, *stardog/StardogJenaModelFactory.java*, *virtuoso/VirtuosoJenaModelFactory.java). For example:*

```
import com.hp.hpl.jena.rdf.model.Model;
import fi.hut.cs.drumbeat.rdf.modelfactory.JenaModelFactoryBase;

public class XXXJenaModelFactory extends JenaModelFactoryBase {
        @Override
        public Model createModel() throws Exception { … }
```

[14] http://jena.apache.org/documentation/javadoc/jena/com/hp/hpl/jena/rdf/model/Model.html
[15] http://jena.apache.org/documentation/javadoc/jena/com/hp/hpl/jena/rdf/model/Model.html
[16] http://www.franz.com/agraph/support/documentation/v4/javadoc/index.html
[17] http://docs.openlinksw.com/jena/
[18] http://docs.stardog.com/java/

```
		@Override
		public Model getModel() throws Exception { … }

		@Override
		public void release() throws Exception { … }
	}
```

3.7 Full sample Java file with comments

```java
import java.util.List;

import org.apache.jena.riot.RDFFormat;
import org.apache.log4j.PropertyConfigurator;

import com.hp.hpl.jena.rdf.model.Model;

import fi.hut.cs.drumbeat.common.config.ComplexProcessorConfiguration;
import fi.hut.cs.drumbeat.common.config.document.ConfigurationDocument;
import fi.hut.cs.drumbeat.ifc.convert.ifc2rdf.util.Ifc2RdfExportUtil;
import fi.hut.cs.drumbeat.ifc.convert.step2ifc.util.IfcParserUtil;
import fi.hut.cs.drumbeat.ifc.data.model.IfcModel;
import fi.hut.cs.drumbeat.ifc.data.schema.IfcSchema;
import fi.hut.cs.drumbeat.ifc.util.IfcModelAnalyser;
import fi.hut.cs.drumbeat.rdf.RdfUtils;
import fi.hut.cs.drumbeat.rdf.modelfactory.JenaModelFactoryBase;
import fi.hut.cs.drumbeat.rdf.modelfactory.MemoryJenaModelFactory;

public class Test {

   private String loggerConfigFilePath;
   private String configFilePath;
   private String inputSchemaFilePath;
   private String inputModelFilePath;
   private String outputSchemaFilePath;
   private String outputModelFilePath;
   RDFFormat outputFileFormat;
   boolean gzipOutputFile;

   public void run() throws Exception {

      //
      // load logger configuration
      //
      PropertyConfigurator.configure(loggerConfigFilePath);

      //
      // load converter configuration
      //
      ConfigurationDocument.load(configFilePath);

      //
      // load IFC schemas
      //
      List<IfcSchema> schemas = IfcParserUtil.parseSchemas(inputSchemaFilePath);

      // export IFC schema(s)
      //
      final JenaModelFactoryBase jenaModelFactory = new MemoryJenaModelFactory();
      for (IfcSchema schema : schemas) {
         // export IFC schema into in-memory Jena graph using default conversion context
         Model schemaGraph = jenaModelFactory.createModel();
         Ifc2RdfExportUtil.exportSchemaToJenaModel(schemaGraph, schema);

         // export the in-memory Jena graph to file
         RdfUtils.exportJenaModelToRdfFile(schemaGraph, outputSchemaFilePath,
outputFileFormat, gzipOutputFile);
      }

      //
      // load IFC model
      //
      IfcModel model = IfcParserUtil.parseModel(inputModelFilePath);

      // get default grounding rule sets
      ComplexProcessorConfiguration groundingConfiguration =
IfcModelAnalyser.getDefaultGroundingRuleSets();

      // ground nodes in the model
      IfcModelAnalyser modelAnalyser = new IfcModelAnalyser(model);

      modelAnalyser.groundNodes(groundingConfiguration);


      //
      // export IFC model
      //

      // export IFC model into in-memory Jena graph using default conversion context
      Model modelGraph = jenaModelFactory.createModel();
      Ifc2RdfExportUtil.exportModelToJenaModel(modelGraph, model);

      // export the in-memory Jena graph to file
      RdfUtils.exportJenaModelToRdfFile(modelGraph, outputModelFilePath,
outputFileFormat, gzipOutputFile);

   }

}
```

4 Configuration

4.1 The common configuration file

The common XML configuration file includes options of the conversion process. The most important sections which are discussed below are IFC-to-RDF conversion contexts (tags *<converter>* in section *<converterPool type="Ifc2Rdf">*) and output Jena models (tags *<jenaModel>* in section *<jenaModelPool>*).

```
<?xml version="1.0" encoding="UTF-8" ?>
<config>
    <!-- Converters -->
    <converterPool type="Ifc2Rdf">
        <converter name="…" enabled="true" default="true">…</converter>
        <converter name="…" enabled="true" default="false">…</converter>
        …
    </converterPool>

    <!—Jena Model factory -->
    <jenaModelPool>
        <jenaModel name="…" enabled="true" default="true">…</ jenaModel>
        <jenaModel name="…" enabled="true" default="false">…</jenaModel>
        …
    </jenaModelPool>
</config>
```

4.1.1 IFC-to-RDF conversion contexts

The converter pool with type *Ifc2Rdf* includes all conversion contexts that can be used during the conversion process. When there are more than one converter context is enabled (with attribute *enabled="true"*) then the converter tool will try to find fist context among them which is set as default (with attribute *default="true"*). If there is no default converter context then the first enabled context is used.

```
    <converterPool type="Ifc2Rdf">
        <converter name="OWL2_RL" enabled="true" default="true">
            <params>
                <param name="OwlProfile" value="OWL2_EL" />
                <param name="Options.PrintPropertyCardinality" value="true" />
                <param name="Options.PrintPropertyDomainAndRange" value="true" />
                <param name="Options.AvoidDuplicationOfPropertyNames" value="true" />
                <param name="Options.PrintPropertyDomainAndRangeAsUnion" value="false" />
                <param name="Options.ForceConvertRdfListToOloOrderedList" value="false" />
                <param name="Options.ForceConvertEnumerationValuesToString" value="false" />
                <param name="Options.ForceConvertBooleanValuesToString" value="false" />
                <param name="Options.ForceConvertPropertyToObjectProperty" value="true" />
                <param name="Options.ExportDebugInfo" value="false" />
                <param name="Ontology.Prefix" value="ifc" />
                <param name="Ontology.NamespaceFormat" value="http://linkedbuildingdata.net/schema/$Schema.Version$#" />
                <param name="Model.Prefix" value="model" />
                <param name="Model.NamespaceFormat" value="http://linkedbuildingdata.net/model/" />

            </params>
        </converter>
        <converter name="OWL1_DL" enabled="true" default="false">
            <params>
                <param name="OwlProfile" value="OWL1_DL" />
                <param name="Options.PrintPropertyCardinality" value="true" />
                <param name="Options.PrintPropertyDomainAndRange" value="true" />
                <param name="Options.AvoidDuplicationOfPropertyNames" value="true" />
                <param name="Options.PrintPropertyDomainAndRangeAsUnion" value="false" />
                <param name="Options.ForceConvertRdfListToOloOrderedList" value="false" />
                <param name="Options.ForceConvertEnumerationValuesToString" value="false" />
                <param name="Options.ForceConvertBooleanValuesToString" value="false" />
                <param name="Options.ForceConvertPropertyToObjectProperty" value="true" />
                <param name="Options.ExportDebugInfo" value="true" />
                <param name="Ontology.Prefix" value="ifc" />
                <param name="Ontology.NamespaceFormat" value="http://linkedbuildingdata.net/schema/$Schema.Version$#" />
                <param name="Model.Prefix" value="model" />
                <param name="Model.NamespaceFormat" value="http://linkedbuildingdata.net/model/" />
            </params>
        </converter>
    </converterPool>
```

The meanings and allowed values of the converter parameters are:

OwlProfile	Specifies the used OWL profile, must be one of the following values: OWL1_Lite, OWL1_DL, OWL1_Full, OWL2_EL, OWL2_QL, OWL2_RL, OWL2_Full
Options.PrintPropertyCardinality	Indicates whether to export property cardinalities
Options.PrintPropertyDomainAndRange	Indicates whether to export property domains and ranges
Options.AvoidDuplicationOfPropertyNames	Indicates whether to create sub properties to avoid duplication of property names
Options.PrintPropertyDomainAndRangeAsUnion	Indicates whether to allow properties domains and ranges as unions
Options.ForceConvertRdfListToOloOrderedList	Indicates whether to use olo:OrderedList instead of rdf:List
Options.ForceConvertEnumerationValuesToString	Indicates whether to force converting enumeration values to strings even if the OWL profile supports enumerations (OWL DL, OWL Full, OWL 2 RL, OWL 2 Full)
Options.ForceConvertBooleanValuesToString	Indicates whether to force converting boolean values to strings even if the OWL profile supports Boolean type (OWL DL, OWL Full, OWL 2 RL, OWL 2 Full)
Options.ForceConvertPropertyToObjectProperty	Indicates whether to consider all properties as object properties even if the OWL profile supports data properties
Options.ExportDebugInfo	Indicates whether to export debug info
Ontology.Prefix	The short ontology prefix, e.g. “ifc”.
Ontology.NamespaceFormat	The ontology namespace (usually includes parameter *$Schema.Version$*).
Model.Prefix	The short model prefix, e.g. “model”.
Model.NamespaceFormat	The ontology namespace.

Read the conversion specification document for more details.

4.1.2 Output Jena models

The output Jena model is selected from the Jena model pool by the specified value of argument *–osn/--output-schema-name* (in case of exporting IFC schema), or *–omn/--output-model-name* (in case of exporting IFC model). An output Jena model configuration include the following parameters:

- the class name of the Jena model factory, i.e. a class that extends the abstract class *fi.hut.cs.drumbeat.rdf.modelfactory.JenaModelFactoryBase* and implements the following abstract methods:

```
public abstract com.hp.hpl.jena.rdf.model.Model createModel() throws Exception;
public abstract com.hp.hpl.jena.rdf.model.ModelgetModel() throws Exception;
public abstract void release() throws Exception;
```

- server access parameters such as *ServerUrl*, *UserName*, *Password* and *ModelId*.

In this release package there are a few Jena model factory classes:

- *fi.hut.cs.drumbeat.rdf.modelfactory.allegrograph.AGJenaModelFactory* – creates a Jena model in AllegroGraph triple store.
- *fi.hut.cs.drumbeat.rdf.modelfactory.virtuoso.VirtuosoJenaModelFactory* – creates a Jena model in OpenLink Virtuoso triple store.
- *fi.hut.cs.drumbeat.rdf.modelfactory.stardog.StardogJenaModelFactory* – creates a Jena model in Stardog triple store.

```
<jenaModelPool>
    <jenaModel name="Memory1" default="true">
        <class>fi.hut.cs.drumbeat.rdf.modelfactory.MemoryJenaModelFactory</class>
    </jenaModel>
    <jenaModel name="AllegroGraph1" default="true">
        <class>fi.hut.cs.drumbeat.rdf.modelfactory.allegrograph.AGJenaModelFactory</class>
        <params>
            <param name="ServerUrl" value="http://localhost:10035/" />
            <param name="UserName" value="admin" />
            <param name="Password" value="admin" />
            <param name="ModelId" value=" IFC" />
        </params>
    </jenaModel>
    <jenaModel name="Virtuoso1" default="true">
        <class>fi.hut.cs.drumbeat.rdf.modelfactory.virtuoso.VirtuosoJenaModelFactory</class>
        <params>
            <param name="ServerUrl" value="jdbc:virtuoso://localhost:1111" />
            <param name="UserName" value="admin" />
            <param name="Password" value="admin" />
            <param name="ModelId" value="IFC" />
        </params>
    </jenaModel>
    <jenaModel name="Stardog1" default="true">
        <class>fi.hut.cs.drumbeat.rdf.modelfactory.stardog.StardogJenaModelFactory</class>
        <params>
            <param name="ServerUrl" value="http://localhost:5822" />
            <param name="UserName" value="admin" />
            <param name="Password" value="admin" />
            <param name="ModelId" value="IFC" />
        </params>
    </jenaModel>
</jenaModelPool>
```

4.2 The logging configuration file

The Ifc2RdfExporter program uses Apache Log4J API for logging messages. The logging configuration is usually written in XML format, but can also be written in Java properties or JSON format. The included logging .xml file in this release package is configured so that the program will logs only messages at INFO, WARNING and ERROR levels to the console and some specific logging files: for common messages, messages from the parser and the converter. To enable debug messages of the converter or other program layers, you need to modify the logging properties. For example:

```
<logger name="fi.hut.cs.drumbeat.ifc.convert">
    <level value="debug"/>
    <appender-ref ref="FILE_PARSER"/>
```

```
        </logger>
```

Please read the Apache Logging Services manual[19] to understand more about Log4J mechanism and how to configure it.

[19] http://logging.apache.org/log4j/1.2/manual.html