INTERNATIONAL HELLENIC UNIVERSITY

Knowledge Management (in the Web)

OWL – API

Duration: 2hrs

Tutorial

Introduction

The OWL API is a Java API and reference implementation for creating, manipulating and serialising OWL Ontologies. The OWL API includes the following components:

- An API for OWL 2 and an efficient in-memory reference implementation
- RDF/XML parser and writer
- OWL/XML parser and writer
- OWL Functional Syntax parser and writer
- Turtle parser and writer
- KRSS parser
- OBO Flat file format parser
- Reasoner interfaces for working with reasoners such as FaCT++, HermiT, Pellet and Racer

Deployment Instructions

In order to deploy the OWL API from within the Netbeans environment, you need to perform the following steps:

1. Download the latest stable release of the API.
2. Create a new Netbeans library (Tools > Libraries).
3. Name the library "owl-api" and associate its jars and javadocs.
4. Create a new Netbeans project.
5. Include the library in the newly created Netbeans project.
6. Now you are ready to use the OWL API for creating Semantic Web applications!

Tutorial Topics

This tutorial introduces you to the basic OWL API functionalities, covering the following:

- *Loading Ontologies*
- *Saving Ontologies*
- *Obtain References to Entities*
- *Work with Data Types and Other Data Ranges*
- *Work with String, Data Values and Language Tags*
- *Adding Axioms*
- *Classes and Instances*
- *Property Assertions*
- *Deleting Entities*
- *Restrictions*

Loading Ontologies

```
// Get hold of an ontology manager
OWLOntologyManager manager = OWLManager.createOWLOntologyManager();

// Load an ontology from the Web
IRI iri = IRI.create("http://www.co-ode.org/ontologies/pizza/pizza.owl");
OWLOntology pizzaOntology = manager.loadOntologyFromOntologyDocument(iri);
System.out.println("Loaded ontology: " + pizzaOntology);

// Remove the ontology so that we can load a local copy.
manager.removeOntology(pizzaOntology);

// We can also load ontologies from files.  Create a file object that points to the local copy
File file = new File("/tmp/pizza.owl");
```

```
// Load the local copy
OWLOntology localPizza = manager.loadOntologyFromOntologyDocument(file);
System.out.println("Loaded ontology: " + localPizza);

// We can always obtain the location where an ontology was loaded from
IRI documentIRI = manager.getOntologyDocumentIRI(localPizza);
System.out.println("    from: " + documentIRI);

// Remove the ontology again so we can reload it later
manager.removeOntology(pizzaOntology);

// When a local copy of one of more ontologies is used, an ontology IRI mapper can be used
// to provide a redirection mechanism.  This means that ontologies can be loaded as if they
// were located on the Web. In this example, we simply redirect the loading from
// http://www.co-ode.org/ontologies/pizza/pizza.owl to our local copy above.
manager.addIRIMapper(new SimpleIRIMapper(iri, IRI.create(file)));

// Load the ontology as if we were loading it from the Web (from its ontology IRI)
IRI pizzaOntologyIRI = IRI.create("http://www.co-ode.org/ontologies/pizza/pizza.owl");
OWLOntology redirectedPizza = manager.loadOntology(pizzaOntologyIRI);
System.out.println("Loaded ontology: " + redirectedPizza);
System.out.println("    from: " + manager.getOntologyDocumentIRI(redirectedPizza));
```

Saving Ontologies

```
// Get hold of an ontology manager
OWLOntologyManager manager = OWLManager.createOWLOntologyManager();

// Load an ontology from the Web.  We load the ontology from a document IRI
IRI docIRI = IRI.create("http://www.co-ode.org/ontologies/pizza/pizza.owl");
OWLOntology pizzaOntology = manager.loadOntologyFromOntologyDocument(docIRI);
System.out.println("Loaded ontology: " + pizzaOntology);

// Save a local copy of the ontology.  (Specify a path appropriate to your setup)
File file = new File("/tmp/local.owl");
manager.saveOntology(pizzaOntology, IRI.create(file.toURI()));

// Ontologies are saved in the format from which they were loaded.
// We can get information about the format of an ontology from its manager
OWLOntologyFormat format = manager.getOntologyFormat(pizzaOntology);
System.out.println("    format: " + format);

// Save the ontology in owl/xml format
OWLXMLOntologyFormat owlxmlFormat = new OWLXMLOntologyFormat();

// Some ontology formats support prefix names and prefix IRIs.
// In our case we loaded the pizza ontology from an rdf/xml format, which supports prefixes.
// When we save the ontology in the new format we will copy the prefixes over
// so that we have nicely abbreviated IRIs in the new ontology document
if(format.isPrefixOWLOntologyFormat()) {
        owlxmlFormat.copyPrefixesFrom(format.asPrefixOWLOntologyFormat());
}
```

```
manager.saveOntology(pizzaOntology, owlxmlFormat, IRI.create(file.toURI()));

// Dump an ontology to System.out by specifying a different OWLOntologyOutputTarget
// Note that we can write an ontology to a stream in a similar way
// using the StreamOutputTarget class
OWLOntologyDocumentTarget documentTarget = new SystemOutDocumentTarget();

// Try another format - The Manchester OWL Syntax
ManchesterOWLSyntaxOntologyFormat manSyntaxFormat =
        new ManchesterOWLSyntaxOntologyFormat();
if(format.isPrefixOWLOntologyFormat()) {
        manSyntaxFormat.copyPrefixesFrom(format.asPrefixOWLOntologyFormat());
}
manager.saveOntology(pizzaOntology, manSyntaxFormat,
        new SystemOutDocumentTarget());
```

Obtain References to Entities

```
// In order to get access to objects that represent entities we need a data factory.
OWLOntologyManager manager = OWLManager.createOWLOntologyManager();

// Get a reference to a data factory from an OWLOntologyManager.
OWLDataFactory factory = manager.getOWLDataFactory();

// Create an object to represent a class.  In this case, we'll choose
// http://www.semanticweb.org/owlapi/ontologies/ontology#A
// as the IRI for our class. There are two ways to create classes (and other entities).

// The first is by specifying the full IRI.  First we create an IRI object:
IRI iri = IRI.create("http://www.semanticweb.org/owlapi/ontologies/ontology#A");

// Create the class
OWLClass clsAMethodA = factory.getOWLClass(iri);

// The second way is to use a prefix manager and specify abbreviated IRIs.
// This is useful for creating lots of entities with the same prefix IRIs.
// First create a prefix manager and specify the default prefix IRI
PrefixManager pm = new DefaultPrefixManager(
        "http://www.semanticweb.org/owlapi/ontologies/ontology#");

// Use the prefix manager and just specify an abbreviated IRI
OWLClass clsAMethodB = factory.getOWLClass(":A", pm);

// Creating entities in the above manner does not "add them to an ontology".
// They are merely objects that allow us to reference certain objects (classes etc.)
// for use in class expressions, and axioms (which can be added to an ontology).

// Create an ontology and add a declaration axiom to the ontology that declares the above class
OWLOntology ontology = manager.createOntology(IRI.create(
        "http://www.semanticweb.org/owlapi/ontologies/ontology"));

// We can add a declaration axiom to the ontology, that essentially adds the class to the
// signature of our ontology.
OWLDeclarationAxiom declarationAxiom = factory.getOWLDeclarationAxiom(clsAMethodA);
```

```
manager.addAxiom(ontology, declarationAxiom);

// Note that it isn't necessary to add declarations to an ontology in order to use an entity.
// For some ontology formats (e.g. the Manchester Syntax), declarations will
// automatically be added in the saved version of the ontology.
```

Work with Data Types and Other Data Ranges

```
// Get hold of a manager to work with
OWLOntologyManager manager = OWLManager.createOWLOntologyManager();
OWLDataFactory factory = manager.getOWLDataFactory();

// OWLDatatype represents named datatypes in OWL.
// Get hold of the integer datatype
OWLDatatype integer = factory.getOWLDatatype(OWL2Datatype.XSD_INTEGER.getIRI());

// Convenience methods of OWLDataFactory for common data types:
OWLDatatype integerDatatype = factory.getIntegerOWLDatatype();
OWLDatatype floatDatatype = factory.getFloatOWLDatatype();
OWLDatatype doubleDatatype = factory.getDoubleOWLDatatype();
OWLDatatype booleanDatatype = factory.getBooleanOWLDatatype();

// The top datatype (rdfs:Literal) can be obtained from the data factory
OWLDatatype rdfsLiteral = factory.getTopDatatype();

// Custom data ranges can be built from these basic datatypes.
// For example, create a data range for integers that are greater or equal to 18
OWLLiteral eighteen = factory.getOWLLiteral(18);

// Create the restriction.
OWLDatatypeRestriction integerGE18 = factory.getOWLDatatypeRestriction(integer,
OWLFacet.MIN_INCLUSIVE, eighteen);

// Restrict the range of the :hasAge data property to 18 or more
PrefixManager pm = new DefaultPrefixManager(
        "http://www.semanticweb.org/ontologies/dataranges#");
OWLDataProperty hasAge = factory.getOWLDataProperty(":hasAge", pm);
OWLDataPropertyRangeAxiom rangeAxiom = factory.getOWLDataPropertyRangeAxiom(
        hasAge, integerGE18);
OWLOntology ontology = manager.createOntology(IRI.create(
        "http://www.semanticweb.org/ontologies/dataranges"));

// Add the range axiom to our ontology
manager.addAxiom(ontology, rangeAxiom);

// Convenience methods for creating datatype restrictions on integers/doubles:
// For example: Create a data range of integers greater or equal to 60
OWLDatatypeRestriction integerGE60 =
factory.getOWLDatatypeMinInclusiveRestriction(60);

// Create a data range of integers less than 16
OWLDatatypeRestriction integerLT16 =
factory.getOWLDatatypeMaxExclusiveRestriction(18);
```

```
// Represent the intersection, union and complement of data types
OWLDataUnionOf concessionaryAge = factory.getOWLDataUnionOf(
        integerLT16, integerGE60);

// Coin names for custom data ranges.
OWLDatatype concessionaryAgeDatatype = factory.getOWLDatatype(
        ":ConcessionaryAge", pm);

// Create a datatype definition axiom
OWLDatatypeDefinitionAxiom datatypeDef = factory.getOWLDatatypeDefinitionAxiom(
        concessionaryAgeDatatype, concessionaryAge);

// Add the definition to our ontology
manager.addAxiom(ontology, datatypeDef);

// Dump our ontology
manager.saveOntology(ontology, new SystemOutDocumentTarget());
```

Work with String, Data Values and Language Tags

```
OWLOntologyManager manager = OWLManager.createOWLOntologyManager();
OWLDataFactory factory = manager.getOWLDataFactory();

// Get a plain literal with an empty language tag
OWLLiteral literal1 = factory.getOWLLiteral("My string literal", "");

// Get an untyped string literal with a language tag
OWLLiteral literal2 = factory.getOWLLiteral("My string literal", "en");

// Typed literals are literals that are typed with a datatype

// Create a typed literal to represent the integer 33
OWLDatatype integerDatatype = factory.getOWLDatatype(
        OWL2Datatype.XSD_INTEGER.getIRI());
OWLLiteral literal3 = factory.getOWLLiteral("33", integerDatatype);

// Shortcut methods on OWLDataFactory for creating typed literals with common datatypes
// Create a literal to represent the integer 33
OWLLiteral literal4 = factory.getOWLLiteral(33);

// Create a literal to represent the double 33.3
OWLLiteral literal5 = factory.getOWLLiteral(33.3);

// Create a literal to represent the boolean value true
OWLLiteral literal6 = factory.getOWLLiteral(true);
```

Adding Axioms

```
// Create the manager that we will use to load ontologies.
OWLOntologyManager manager = OWLManager.createOWLOntologyManager();

// Create an ontology and name it "http://www.co-ode.org/ontologies/testont.owl"
IRI ontologyIRI = IRI.create("http://www.co-ode.org/ontologies/testont.owl");
```

```
// Create the document IRI for our ontology
IRI documentIRI = IRI.create("file:/tmp/MyOnt.owl");
// Set up a mapping, which maps the ontology to the document IRI
SimpleIRIMapper mapper = new SimpleIRIMapper(ontologyIRI, documentIRI);
manager.addIRIMapper(mapper);

// Now create the ontology - we use the ontology IRI (not the physical URI)
OWLOntology ontology = manager.createOntology(ontologyIRI);

// Specify that A is a subclass of B.  Add a subclass axiom.
OWLDataFactory factory = manager.getOWLDataFactory();

// Get hold of references to class A and class B.
OWLClass clsA = factory.getOWLClass(IRI.create(ontologyIRI + "#A"));
OWLClass clsB = factory.getOWLClass(IRI.create(ontologyIRI + "#B"));

// Create the axiom
OWLAxiom axiom = factory.getOWLSubClassOfAxiom(clsA, clsB);

// Add the axiom to the ontology, so that the ontology states that A is a subclass of B.
AddAxiom addAxiom = new AddAxiom(ontology, axiom);

// Use the manager to apply the change
manager.applyChange(addAxiom);

// The ontology will now contain references to class A and class B.
for (OWLClass cls : ontology.getClassesInSignature()) {
        System.out.println("Referenced class: " + cls);
}

// We should also find that B is an ASSERTED superclass of A
Set<OWLClassExpression> superClasses = clsA.getSuperClasses(ontology);
System.out.println("Asserted superclasses of " + clsA + ":");
for (OWLClassExpression desc : superClasses) {
        System.out.println(desc);
}

// Save the ontology to the location where we loaded it from, in the default ontology format
manager.saveOntology(ontology);
```

Classes and Instances

```
// For example, suppose we wanted to specify that :Mary is an instance of the class :Person.
// First we need to obtain the individual :Mary and the class :Person

// Create an ontology manager to work with
OWLOntologyManager manager = OWLManager.createOWLOntologyManager();
OWLDataFactory dataFactory = manager.getOWLDataFactory();
String base = "http://example.com/owl/families/";
PrefixManager pm = new DefaultPrefixManager(base);

// Get reference to the :Person class (the full IRI: http://example.com/owl/families/Person)
OWLClass person = dataFactory.getOWLClass(":Person", pm);
```

```
// Get reference to the :Mary class (the full IRI: <http://example.com/owl/families/Mary>)
OWLNamedIndividual mary = dataFactory.getOWLNamedIndividual(":Mary", pm);

// Create a ClassAssertion to specify that :Mary is an instance of :Person
OWLClassAssertionAxiom classAssertion =
        dataFactory.getOWLClassAssertionAxiom(person, mary);

// Add the class assertion to the ontology
OWLOntology ontology = manager.createOntology(IRI.create(base));
manager.addAxiom(ontology, classAssertion);

// Dump the ontology to stdout
manager.saveOntology(ontology, new SystemOutDocumentTarget());
```

Property Assertions

```
OWLOntologyManager man = OWLManager.createOWLOntologyManager();
String base = "http://www.semanticweb.org/ontologies/individualsexample";
OWLOntology ont = man.createOntology(IRI.create(base));
OWLDataFactory dataFactory = man.getOWLDataFactory();

// We would like to state that matthew has a father who is peter.
// We need a subject and object - matthew is the subject and peter is the object.
// We use the data factory to obtain references to these individuals
OWLIndividual matthew = dataFactory.getOWLNamedIndividual(
        IRI.create(base + "#matthew"));
OWLIndividual peter = dataFactory.getOWLNamedIndividual(
        IRI.create(base + "#peter"));

// Link the subject and object with the hasFather property.
OWLObjectProperty hasFather = dataFactory.getOWLObjectProperty(
        IRI.create(base + "#hasFather"));

// Create the actual assertion (triple), as an object property assertion axiom
OWLObjectPropertyAssertionAxiom assertion =
        dataFactory.getOWLObjectPropertyAssertionAxiom(hasFather, matthew, peter);

// Finally, add the axiom to our ontology and save
AddAxiom addAxiomChange = new AddAxiom(ont, assertion);
man.applyChange(addAxiomChange);

// We can also specify that matthew is an instance of Person.
OWLClass personClass = dataFactory.getOWLClass(IRI.create(base + "#Person"));

// Create a Class Assertion to specify that matthew is an instance of Person.
OWLClassAssertionAxiom ax = dataFactory.getOWLClassAssertionAxiom(
        personClass, matthew);

// Add this axiom to our ontology.
man.addAxiom(ont, ax);

// Save our ontology
```

```
man.saveOntology(ont, IRI.create("file:/tmp/example.owl"));
```

Deleting Entities

```
// Load the pizza ontology.
OWLOntologyManager man = OWLManager.createOWLOntologyManager();
OWLOntology ont = man.loadOntologyFromOntologyDocument(
        IRI.create("http://www.co-ode.org/ontologies/pizza/pizza.owl"));

// We can't directly delete individuals, properties or classes from an ontology because
// ontologies don't directly contain entities - they are merely referenced by the
// axioms that the ontology contains.  We can use the OWLEntityRemove utility class, which
// will remove an entity (class, property or individual) from a set of ontologies.
OWLEntityRemover remover = new OWLEntityRemover(man, Collections.singleton(ont));
System.out.println("Number of individuals: " + ont.getIndividualsInSignature().size());

// Loop through each individual that is referenced in the pizza ontology and ask it
// to accept a visit from the entity remover.  The remover will automatically accumulate
// the necessary changes to remove the individual from the ontologies it knows about
for (OWLNamedIndividual ind : ont.getIndividualsInSignature()) {
        ind.accept(remover);
}

// Get all of the changes from the entity remover, which should be applied to remove all the
// individuals that we have visited from the pizza ontology.  Notice that "batch" deletes can
// essentially be performed - we simply visit all of the classes, properties and individuals that we
// want to remove and then apply ALL of the changes after using the entity remover to collect them
man.applyChanges(remover.getChanges());
System.out.println("Number of individuals: " + ont.getIndividualsInSignature().size());

// At this point, if we wanted to reuse the entity remover, we would have to reset it
remover.reset();
```

Restrictions

```
OWLOntologyManager man = OWLManager.createOWLOntologyManager();
String base = "http://org.semanticweb.restrictionexample";
OWLOntology ont = man.createOntology(IRI.create(base));

// Add an axiom to state that all Heads have parts that are noses.
// First we need to obtain references to our hasPart property and our Nose class
OWLDataFactory factory = man.getOWLDataFactory();
OWLObjectProperty hasPart = factory.getOWLObjectProperty(IRI.create(base + "#hasPart"));
OWLClass nose = factory.getOWLClass(IRI.create(base + "#Nose"));

// Now create a restriction to describe the class of individuals that have at least one
// part that is a kind of nose
OWLClassExpression hasPartSomeNose =
        factory.getOWLObjectSomeValuesFrom(hasPart, nose);

// Obtain a reference to the Head class so that we can specify that Heads have noses
OWLClass head = factory.getOWLClass(IRI.create(base + "#Head"));

// We now want to state that Head is a subclass of hasPart some Nose
```

```
OWLSubClassOfAxiom ax = factory.getOWLSubClassOfAxiom(head, hasPartSomeNose);

// Add the axiom to our ontology
AddAxiom addAx = new AddAxiom(ont, ax);
man.applyChange(addAx);
```