OWL API

May 2008

Mari Carmen Suárez-Figueroa, Oscar Corcho

{mcsuarez, ocorcho}@fi.upm.es

Ontology Engineering Group. Departamento de Inteligencia Artificial

Facultad de Informática

Universidad Politécnica de Madrid

Loading and Saving an Ontology

```
public class Example1 {

    public static void main(String[] args) {
        try {
            // A simple example of how to load and save an ontology
            // We first need to obtain a copy of an OWLOntologyManager, which, as the
            // name suggests, manages a set of ontologies.  An ontology is unique within
            // an ontology manager.  To load multiple copies of an ontology, multiple managers
            // would have to be used.
            OWLOntologyManager manager = OWLManager.createOWLOntologyManager();
            // We load an ontology from a physical URI - in this case we'll load the pizza
            // ontology.
            URI physicalURI = URI.create("http://www.co-ode.org/ontologies/pizza/2007/02/12/pizza.owl");
            // Now ask the manager to load the ontology
            OWLOntology ontology = manager.loadOntologyFromPhysicalURI(physicalURI);
            // Print out all of the classes which are referenced in the ontology
            for(OWLClass cls : ontology.getReferencedClasses()) {
                System.out.println(cls);
            }
            // Now save a copy to another location in OWL/XML format (i.e. disregard the
            // format that the ontology was loaded in).
            // (To save the file on windows use a URL such as  "file:/C:\\windows\\temp\\MyOnt.owl")
            URI physicalURI2 = URI.create("file:/tmp/MyOnt2.owl");
            manager.saveOntology(ontology, new OWLXMLOntologyFormat(), physicalURI2);
            // Remove the ontology from the manager
            manager.removeOntology(ontology.getURI());
        }
        catch (OWLOntologyCreationException e) {
            System.out.println("The ontology could not be created: " + e.getMessage());
        }
        catch (OWLOntologyStorageException e) {
            System.out.println("The ontology could not be saved: " + e.getMessage());
        }
    }
}
```

Creating an Empty Ontology, Adding Axioms, and Saving (I)

```
public class Example2 {

    public static void main(String[] args) {
        try {
            // We first need to obtain a copy of an OWLOntologyManager, which, as the
            // name suggests, manages a set of ontologies.  An ontology is unique within
            // an ontology manager.  To load multiple copies of an ontology, multiple managers
            // would have to be used.
            OWLOntologyManager manager = OWLManager.createOWLOntologyManager();

            // All ontologies have a URI, which is used to identify the ontology.  You should
            // think of the ontology URI as the "name" of the ontology.  This URI frequently
            // resembles a Web address (i.e. http://...), but it is important to realise that
            // the ontology URI might not necessarily be resolvable.  In other words, we
            // can't necessarily get a document from the URI corresponding to the ontology
            // URI, which represents the ontology.
            // In order to have a concrete representation of an ontology (e.g. an RDF/XML
            // file), we MAP the ontology URI to a PHYSICAL URI.  We do this using a URIMapper

            // Let's create an ontology and name it "http://www.co-ode.org/ontologies/testont.owl"
            // We need to set up a mapping which points to a concrete file where the ontology will
            // be stored. (It's good practice to do this even if we don't intend to save the ontology).
            URI ontologyURI = URI.create("http://www.co-ode.org/ontologies/testont.owl");
            // Create a physical URI which can be resolved to point to where our ontology will be saved.
            URI physicalURI = URI.create("file:/tmp/MyOnt.owl");
            // Set up a mapping, which maps the ontology URI to the physical URI
            SimpleURIMapper mapper = new SimpleURIMapper(ontologyURI, physicalURI);
            manager.addURIMapper(mapper);

            // Now create the ontology - we use the ontology URI (not the physical URI)
            OWLOntology ontology = manager.createOntology(ontologyURI);
            // Now we want to specify that A is a subclass of B.  To do this, we add a subclass
            // axiom.  A subclass axiom is simply an object that specifies that one class is a
            // subclass of another class.
```

Creating an Empty Ontology, Adding Axioms, and Saving (II)

```
    // We need a data factory to create various object from.  Each ontology has a reference
    // to a data factory that we can use.
    OWLDataFactory factory = manager.getOWLDataFactory();
    // Get hold of references to class A and class B.  Note that the ontology does not
    // contain class A or classB, we simply get references to objects from a data factory that represent
    // class A and class B
    OWLClass clsA = factory.getOWLClass(URI.create(ontologyURI + "#A"));
    OWLClass clsB = factory.getOWLClass(URI.create(ontologyURI + "#B"));
    // Now create the axiom
    OWLAxiom axiom = factory.getOWLSubClassAxiom(clsA, clsB);
    // We now add the axiom to the ontology, so that the ontology states that
    // A is a subclass of B.  To do this we create an AddAxiom change object.
    AddAxiom addAxiom = new AddAxiom(ontology, axiom);
    // We now use the manager to apply the change
    manager.applyChange(addAxiom);

    // The ontology will now contain references to class A and class B - let's
    // print them out
    for(OWLClass cls : ontology.getReferencedClasses()) {
        System.out.println("Referenced class: " + cls);
    }
    // We should also find that B is a superclass of A
    Set<OWLDescription> superClasses = clsA.getSuperClasses(ontology);
    System.out.println("Superclasses of " + clsA + ":");
    for(OWLDescription desc : superClasses) {
        System.out.println(desc);
    }

    // Now save the ontology.  The ontology will be saved to the location where
    // we loaded it from, in the default ontology format
    manager.saveOntology(ontology);

}
catch (OWLException e) {
    e.printStackTrace();
}
```

Adding an Object Property

```
public class Example4 {

    public static void main(String[] args) {
        try {
            OWLOntologyManager man = OWLManager.createOWLOntologyManager();

            String base = "http://www.semanticweb.org/ontologies/individualsexample";

            OWLOntology ont = man.createOntology(URI.create(base));

            OWLDataFactory dataFactory = man.getOWLDataFactory();

            // In this case, 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.getOWLIndividual(URI.create(base + "#matthew"));
            OWLIndividual peter = dataFactory.getOWLIndividual(URI.create(base + "#peter"));
            // We want to link the subject and object with the hasFather property, so use the data factory
            // to obtain a reference to this object property.
            OWLObjectProperty hasFather = dataFactory.getOWLObjectProperty(URI.create(base + "#hasFather"));
            // Now create the actual assertion (triple), as an object property assertion axiom
            // matthew --> hasFather --> peter
            OWLObjectPropertyAssertionAxiom assertion = dataFactory.getOWLObjectPropertyAssertionAxiom(matthew, hasFather, peter);
            // Finally, add the axiom to our ontology and save
            AddAxiom addAxiomChange = new AddAxiom(ont, assertion);
            man.applyChange(addAxiomChange);

            man.saveOntology(ont, URI.create("file:/tmp/example.owl"));
        }
        catch (OWLOntologyCreationException e) {
            System.out.println("Could not create ontology: " + e.getMessage());
        }
        catch(OWLOntologyChangeException e) {
            System.out.println("Problem editing ontology: " + e.getMessage());
        }
```

Deleting Entities

```
try {
    // The pizza ontology contains several individuals that represent
    // countries, which describe the country of origin of various pizzas
    // and ingredients.  In this example we will delete them all.
    // First off, we start by loading the pizza ontology.
    OWLOntologyManager man = OWLManager.createOWLOntologyManager();
    OWLOntology ont = man.loadOntologyFromPhysicalURI(URI.create("http://www.co-ode.org/ontologies/pizza/2007/02/12/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.  For example, if an ontology contained a subclass axiom
    // SubClassOf(A, B) which stated A was a subclass of B, then that ontology would contain references
    // to classes A and B.  If we essentially want to "delete" classes A and B from this ontology we
    // have to remove all axioms that REFERENCE class A and class B (in this case just one axiom
    // SubClassOf(A, B)).  To do this, we can use the OWLEntityRemove utility class, which will remove
    // an entity (class, property or individual) from a set of ontologies.

    // Create the entity remover - in this case we just want to remove the individuals from
    // the pizza ontology, so pass our reference to the pizza ontology in as a singleton set.
    OWLEntityRemover remover = new OWLEntityRemover(man, Collections.singleton(ont));
    System.out.println("Number of individuals: " + ont.getReferencedIndividuals().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 changes which are necessary to remove the individual from the ontologies (the pizza
    // ontology) which it knows about
    for(OWLIndividual ind : ont.getReferencedIndividuals()) {
        ind.accept(remover);
    }
    // Now we get all of the changes from the entity remover, which should be applied to
    // remove all of 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 afer using the
    // entity remover to collect them
    man.applyChanges(remover.getChanges());
    System.out.println("Number of individuals: " + ont.getReferencedIndividuals().size());
    // At this point, if we wanted to reuse the entity remover, we would have to reset it
```