The ODMG API

by Brian McCallister

Table of contents

1. Introduction

The *ODMG API* is an implementation of the ODMG 3.0 Object Persistence API (http://www.odmg.org/) . The ODMG API provides a higher-level API and query language based interface over the PersistenceBroker API (../../docu/guides/pb-guide.html) .

More detailed information can be found in the ODMG-guide (../../docu/guides/odmg-guide.html) and in the other reference guides (../../docu/guides/summary.html) .

This tutorial operates on a simple example class:

```
package org.apache.ojb.tutorials;

public class Product
{
/* Instance Properties */

private Double price;
private Integer stock;
private String name;

/* artificial property used as primary key */

private Integer id;

/* Getters and Setters */
...
}
```

The metadata descriptor for mapping this class is described in the mapping tutorial (../../docu/tutorials/mapping-tutorial.html)

When using 1:1, 1:n and m:n references (the example doesn't use it) the ODMG-api need specific metadata settings on relationship definition, the mandatory settings are listed in the ODMG-Guide (../../docu/guides/odmg-guide.html#metadata) - additional info see auto-xxx settings (../../docu/guides/basic-technique.html#cascading) and repository file settings (../../docu/guides/repository.html) .

As with the other tutorials, the source code for this tutorial is contained in the `tutorials-src.jar` which can be downloaded here (http://www.apache.org/dyn/closer.cgi/db/ojb/) . The source files are contained in the `org/apache/ojb/tutorial2/` directory.
You can try it out with the ojb-blank project which can be downloaded from the same place and is described in the Getting started (../../docu/getting-started.html) section.

Further information about the OJB odmg-api implementation can be found in the ODMG guide (../../docu/guides/odmg-guide.html) .

2. Initializing ODMG

The ODMG implementation needs to have a database opened for it to access. This is accomplished via the following code:

```
Implementation odmg = OJB.getInstance();
Database db = odmg.newDatabase();
db.open("default", Database.OPEN_READ_WRITE);

/* ... use the database ... */

db.close();
```

With method call `OJB.getInstance()` always a **new** org.odmg.Implementation (../../api/org/odmg/Implementation.html) instance will be created and `odmg.newDatabase()` returns a new `Database` instance.

Call `db.open(...)` opens an ODMG `Database` using the name specified in metadata for the database (../../docu/guides/odmg-guide.html#lookup-odmg) -- "default" in this case. Notice the `Database` is opened in read/write mode. It is possible to open it in *read-only* or *write-only* modes as well.

Once a `Implementation` instance is created and a `Database` has been opened it is available for use. Unlike `PersistenceBroker` instances (../../docu/guides/pb-guide.html) , ODMG `Implementation` and `Database` instances are threadsafe and can typically be used for the entire lifecycle of an application. There is no need to call the `Database.close()` method until the database is truly no longer needed.

The `OJB.getInstance()` function provides the ODMG `Implementation` instance required for using the ODMG API. From here on out it is straight ODMG code that should work against any compliant ODMG implementation.

3. Persisting New Objects

Persisting an object via the ODMG API is handled by writing it to the peristence store within the context of a transaction:

```
public static void storeNewProduct(Product product)
{
    // get the used Implementation instance
    Implementation odmg = ...;
    Transaction tx = odmg.newTransaction();
    tx.begin();
    // get current used Database instance
    Database db = odmg.getDatabase(null);
    // make persistent new object
    db.makePersistent(product);
    tx.commit();
}
```

Once the ODMG implementation has been obtained it is used to begin a transaction, obtain a write lock on the `Product`, and commit the transaction. It is very important to note that all changes need to be made within transactions in the ODMG API. When the transaction is committed the changes are made to the database. Until the transaction is committed the database is unaware of any changes -- they exist solely in the object model.

4. Querying Persistent Objects

The ODMG API uses the OQL query language for obtaining references to persistent objects. OQL is very similar to SQL, and using it is very similar to use JDBC. The ODMG implementation is used to create a query, the query is specifed, executed, and a list fo results is returned:

```
public static Product findProductByName(String name) throws Exception
{
    // get the used Implementation instance
    Implementation odmg = ...;
    Transaction tx = odmg.newTransaction();
    tx.begin();

    OQLQuery query = odmg.newOQLQuery();
    query.create("select products from "
                 + Product.class.getName()
                 + " where name = $1");
    query.bind(name);
    List results = (List) query.execute();
    Product product = (Product) results.iterator().next();

    tx.commit();
    return product;
}
```

5. Updating Persistent Objects

Updating a persistent object is done by modifying it in the context of a transaction, and then committing the transaction:

```
public static void sellProduct(Product product, int number)
{
    // get the used Implementation instance
    Implementation odmg = ...;
    Transaction tx = odmg.newTransaction();
    tx.begin();
    tx.lock(product, Transaction.WRITE);
    product.setStock(new Integer(product.getStock().intValue() -  number));

    tx.commit();
}
```

The sample code obtains a write lock on the object (**before** the changes are made), binding it to the transaction, changes the object, and commits the transaction. The newly modified `Product` now has a new `stock` value.

6. Deleting Persistent Objects

Deleting persistent objects requires directly addressing the `Database` which contains the persistent object. This can be obtained from the ODMG `Implementation` by asking for it. Once retrieved, just ask the `Database` to delete the object. Once again, this is all done in the context of a transaction.

```
public static void deleteProduct(Product product)
{
    // get the used Implementation instance
    Implementation odmg = ...;
    Transaction tx = odmg.newTransaction();

    tx.begin();
    // get current used Database instance
    Database db = odmg.getDatabase(null);
    db.deletePersistent(product);
    tx.commit();
}
```

It is important to note that the `Database.deletePerstient()` call does not delete the object itself, just the persistent representation of it. The transient object still exists and can be used however desired -- it is simply no longer persistent.