A practical introduction to TriActive JDO

Code e-mail it!

Transparently persistent data regardless of the underlying data store

Level: Introductory

Jeff Gunther (jeff.gunther@intalgent.com)
General Manager, Intalgent Technologies
November 4, 2003

TriActive JDO (TJDO) is a lightweight, open source implementation of Sun's Java Data Objects (JDO) 1.0 specification. With it, developers can use a transparent persistence mechanism with any JDBC-compliant database and any Java object. In this article, Java developer Jeff Gunther provides an introduction to TJDO by demonstrating how a domain model can be persisted to a MySQL database. In addition to illustrating specific implementation details for TJDO, the article reviews the basic concepts and components of the JDO specification.

As the Java platform rapidly advances into every corner of the modern enterprise, developers are inundated with acronyms for new technologies and specifications that promise to cure every IT ailment. Even within the Java community, it's difficult for developers and management alike to determine which technologies are simply a flash in the pan and which ones will be sustainable. If you surveyed an average group of Java developers and asked them how they persist data within their applications, you'd probably find a wide variety of answers. Some of the respondents might be using basic JDBC against a relational database; some might be using a pure object-oriented database; while others might be using Container Managed Persistence 2.0 (CMP) within an application server. Though each of these solutions can and do provide persistence services to developers, each technique has its own set of distinct disadvantages, restrictions, and overhead.

Related content:
Hands-on Java Data Objects
IBM developer kits for Java (downloads)
Subscribe to the developerWorks newsletter
developerWorks Toolbox subscription

Also in the Java zone:
Tutorials
Tools and products
Code and components
Articles

Similar to other initiatives within the Java community, the JDO specification was created through the Java Community Process under Java Specification Request 12. One of the primary objectives of the JDO architects was to provide developers a transparent mechanism to handle and manipulate persistent information regardless of the underlying data store. Since its inception, it has been designed to shield developers from the drudgery of developing a persistence infrastructure. Because developers have a common API that interoperates across a variety of data stores, development teams can postpone the decision of what data stores will be used and supported for a particular project. Each of these benefits diminishes the coding effort and allows developers to focus on other pertinent areas of the project. Like other specifications within the Java family, JDO eliminates the risk of becoming locked into a particular vendor. Each vendor's JDO implementation provides a distinct set of features and types of supported data sources. Depending on your project's requirements and budget, one vendor might be more attractive than another.

To give you an opportunity to test out JDO without having to purchase a license, we'll work with TriActive JDO (TJDO), a lightweight, open source implementation of the JDO 1.0 specification. TJDO supports a variety of JDBC-compliant databases as its data store.

Environment requirements

To fully test out TJDO using the provided files, your environment must meet the following minimum requirements:

- Java 2 SDK, Standard Edition 1.4 or later installed

- Apache Ant 1.5.3 or later installed
- MySQL Server 4.0.14 or later installed

See Resources for links to these technologies. Note that while the Ant build script should be cross-platform, the example class has only been tested and verified on Microsoft Windows XP. Before getting started reviewing some code and the basics of JDO, we'll start by installing and configuring the source code.

The source code
If you're like most developers, learning a new technology is an iterative process of trial and error. To help jump start your exploration of TJDO and expand your general knowledge of JDO, a complete build package, series of configuration files, and example source code are provided (see Resources for links to the source package and other required technologies). Additionally, if you want to integrate TJDO within another project, the provided package and Ant build script will assist you in completing the integration effort.

Installing and configuring the example code

To install the source code package, complete the following steps:

1. Download the source code package.
2. Unzip the j-tjdo.zip file into a temporary directory.
3. Create a database called tjdo within your MySQL environment
4. Open the `common.properties` file and modify the MySQL key/value pairs to match your environment. Here is an example:

```
...
mysql.server=127.0.0.1
mysql.database=tjdo
mysql.user=root
mysql.password=
```

Steps to use TJDO

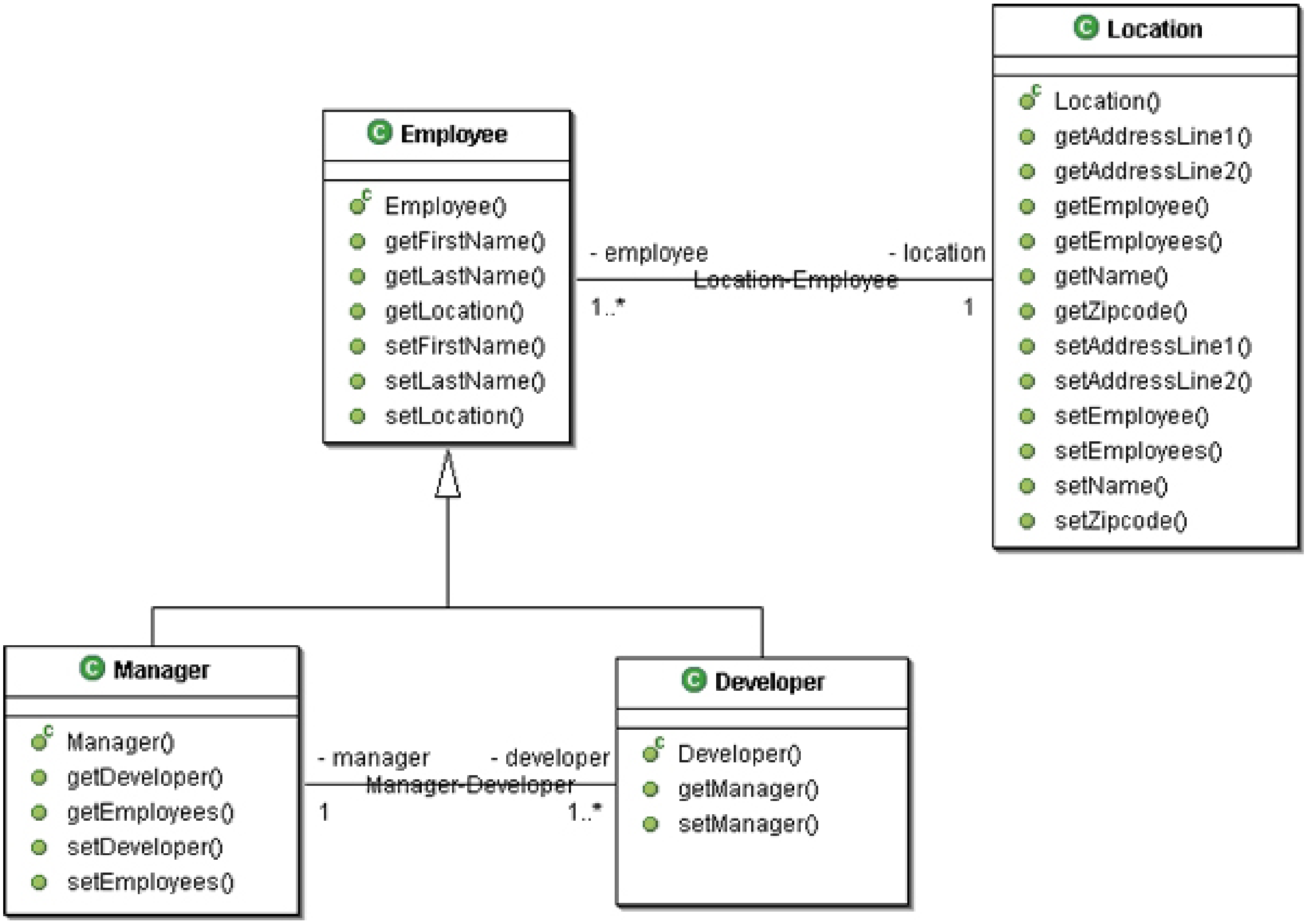
The primary goal of this article is to give you enough knowledge and information to get started using TJDO within your own applications. Before covering how to test out the provided example code, I'll review the general steps necessary to use TJDO. You'll find that many of these steps are applicable to other JDO implementations. These steps are not necessary for you to complete before continuing. They are simply provided to give you an outline for future projects. Each of these steps was completed during the development of the provided source package.

1. Create a domain model as plain old Java objects (POJOs).
2. Create the supporting application code.
3. Compile your classes.
4. Create an XML metadata file that describes the persistence behavior of the domain classes.
5. Enhance the bytecode of the compiled classes.
6. If you're using a relational database as the data store, create the appropriate database schema.

Example domain model

To provide a context for reviewing the basic concepts and APIs of JDO, the example source code includes a simple domain model. Figure 1 illustrates a UML diagram of the POJOs. In this example, a `Developer` instance represents a particular employee that develops software. Similarly, a `Manager` instance represents a particular employee that manages other employees. A `Location` instance represents a physical building where employees, developers and a manager work.

Figure 1. A UML diagram of the Java objects



These domain classes are used with supporting classes to persist data to MySQL.

JDO API basics

Now that we've covered how to configure your workstation and the basic domain model, we'll review how to use the JDO APIs to persist these objects and their relationships to MySQL. Because this article's purpose is to provide a broad introduction of JDO and TJDO, I won't dive into much detail about any particular feature of the JDO API. If you are interested in more information about a particular component, you'll find a link to the JDO 1.0 specification in Resources.

Understanding JDO concepts
"Hands-on Java Data Objects" provides an excellent introduction to JDO concepts and the background behind this technology. As the title implies, this tutorial also includes some practical, hands-on exercises to help developers get up to speed with this powerful new technology.

The code snippet in Listing 1 illustrates the initial properties necessary for connecting TJDO to MySQL.

Listing 1. TJDOTest.java file

```
0 ...
1    public static void main(String[] args)
2    {
3         Properties props = new Properties();
4
5         props.setProperty("javax.jdo.PersistenceManagerFactoryClass",
          "com.triactive.jdo.PersistenceManagerFactoryImpl");
6
7         props.setProperty("javax.jdo.option.ConnectionDriverName",
          "com.mysql.jdbc.Driver");
8         props.setProperty("javax.jdo.option.ConnectionURL",
          "jdbc:mysql://" + args[0] + "/tjdo?autoReconnect=yes");
9         props.setProperty("javax.jdo.option.ConnectionUserName", args[1]);
10        props.setProperty("javax.jdo.option.ConnectionPassword", args[2]);
11        props.setProperty("com.triactive.jdo.autoCreateTables", "true");
12
13        PersistenceManagerFactory pmf =
             JDOHelper.getPersistenceManagerFactory(props);
14        PersistenceManager pm = pmf.getPersistenceManager();
15
16        Transaction tx = pm.currentTransaction();
17
18        try
19        {
20            tx.begin();
21 ...
```

Let's step through the important parts of this code listing:

- The JDO specification requires that each JDO vendor provide a class that implements `PersistenceManagerFactory`. The `PersistenceManagerFactory` is used to obtain `PersistenceManager` instances. A `PersistenceManager` is the primary interface used during the development of supporting application code and is responsible for persisting data to MySQL.
- Lines 7 through 10 define the connection details to MySQL. Each of the arguments passed into this class are defined within the `common.properties` file found within the root of the build directory.
- If they don't exist, line 11 instructs TJDO to automatically create the supporting tables within MySQL at runtime.
- On lines 13 and 14, the properties are passed into a helper class, a `PersistanceManagerFactory` is created, and a `PersistanceManager` is created.
- Lines 16 through 20 ensure that all the data is transitionally consistent, by creating a `Transaction` object from `PersistanceManager` and starting it.

Listing 2 illustrates the domain model being manipulated and populated with data. First, you will create a `Location` object. Then you create two employees: one `Developer` and one `Manager`. Finally, we add the newly created employees to the `Location`.

Listing 2. TJDOTest.java file

```
...
       Location location = new Location();
       location.setName("SomeLocation");
       location.setAddressLine1("1234 Some Street");
       location.setAddressLine2("Suite 111");
       location.setZipcode("12345");

       ArrayList employees = new ArrayList();
       ArrayList developers = new ArrayList();

       Developer developer = new Developer();
       developer.setFirstName("Jane");
       developer.setLastName("Doe");
       developer.setLocation(location);
       employees.add(developer);
       developers.add(developer);

       Manager manager = new Manager();
       manager.setFirstName("John");
       manager.setLastName("Smith");
       manager.setLocation(location);
       manager.setEmployees(developers);
       employees.add(manager);

       location.setEmployees(employees);
...
```

The code snippet in Listing 3 illustrates the beauty of JDO. Each object created in Listing 2 is passed to the `PersistanceManager` into the `makePersistent` function. That's it. All the data was magically added to the database. You didn't have to worry about any `SQL INSERT` statements, database connections, or relationship tables. TJDO completely handles the job of taking the objects and inserting the data into the database.

Listing 3. TJDOTest.java file

```
...
pm.makePersistent(developer);
pm.makePersistent(manager);
pm.makePersistent(location);

tx.commit();
...
```

Creating the XML metadata files

Before the supporting application code and domain model can be used together, an XML metadata file must be created that describes the persistence-capable class. These metadata files are used during the bytecode enhancing process and at run time. For each class, a file named `<class-name>.jdo` must be created. Listing 4 illustrates the `Location.jdo` file used in the provided domain model:

Listing 4. Location.jdo file

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <!DOCTYPE jdo PUBLIC "-//Sun Microsystems, Inc.//DTD Java Data Objects
         Metadata 1.0//EN"
     "http://java.sun.com/dtd/jdo_1_0.dtd">
3
4   <jdo>
5       <package name="test">
6       <class name="Location" identity-type="datastore">
7         <field name="employees">
8           <collection element-type="Employee">
9           </collection>
10        </field>
11        <field name="name">
12          <extension vendor-name="triactive" key="length" value="max 50"/>
13        </field>
14        <field name="addressLine1">
15          <extension vendor-name="triactive" key="length" value="max 100"/>
16        </field>
17        <field name="addressLine2">
18          <extension vendor-name="triactive" key="length" value="max 100"/>
19        </field>
20        <field name="zipcode">
21          <extension vendor-name="triactive" key="length" value="max 5"/>
22        </field>
23      </class>
24      </package>
25   </jdo>
```

Let's examine the important parts of this XML file:

- Lines 6 through 23 define the attributes of the class and the appropriate field elements. The second attribute of the class element is called `identity-type`. TJDO only supports `datastore` as its identity type.
- Lines 7 through 10 define a collection field. This field is used to store the employees that work at a particular location. The `element-type` attribute of the collection element defines the type of class that is stored within the collection.
- Lines 11 through 22 define various string fields. The extension element is used by TJDO to determine the attributes of

the MySQL table.

For more information about the various elements and attributes of a JDO metadata file, see Resources.

Building, enhancing, and testing the example code

To compile, build, and test the package, complete the following steps:

1. Verify that a database called `tjdo` is created within your MySQL database server.
2. Type `ant clean` within the directory where you unpacked the source code to clean the environment.
3. Type `ant` to start the build process.

If your environment met the requirements and was properly configured, you should have seen something similar to Listing 5:

Listing 5. Successful build output

```
Buildfile: build.xml

init:
    [mkdir] Created dir: D:\TJDO\tjdo\dist

compile-common:

compile-module:
     [echo] Compiling ...
    [mkdir] Created dir: D:\TJDO\tjdo\build
    [mkdir] Created dir: D:\TJDO\tjdo\build\classes
    [javac] Compiling 5 source files to D:\TJDO\tjdo\build\classes

enhance:
     [copy] Copying 4 files to D:\TJDO\tjdo\build\classes
    [apply] Enhancing class test.Employee
    [apply] Enhancing class test.Developer
    [apply] Enhancing class test.Location
    [apply] Enhancing class test.Manager
    [apply] done.

package-common:
      [jar] Building jar: D:\TJDO\tjdo\dist\tjdo-demo.jar

default:

BUILD SUCCESSFUL
Total time: 5 seconds
```

Type `ant test` to test the provided code. If your environment was properly configured, the `tjdo` database within MySQL should have all the `Location`, `Developer` and `Manager` data within it.

Summary

TJDO, an open source implementation of Sun's Java Data Objects (JDO) 1.0 specification, provides developers a great way to transparently persistent data regardless of the underlying data store. Even through the JDO specification is just beginning its evolution, it has already filled a void within the Java community. JDO implementations, like TJDO, combine relational databases with the object-oriented Java language and provide developers a powerful tool that can be put to work today. To assist you in that task, the example package provides a build and packaging framework that can be used to incorporate TJDO into your own projects.

Resources

- Download the example source code used throughout this article.
- The TJDO project page on Sourceforge has all the information you'll need to get started using this JDO implementation.
- See the Java Data Objects specification.

- Visit the IBM developer kits page for a list of the SDKs for Java technology available from IBM.
- Download Ant 1.5.4 from the Apache Software Foundation.
- Download MySQL 4.0.14 from MySQL AB.
- For an excellent introduction to JDO, Paul Monday's tutorial "Hands-on Java Data Objects" (*developerWorks*, July 2002) is the place to start.
- JDO Central is an extensive resource for experienced and novice JDO developers alike. In particular, the Web-based forums are a great way to collaborate and share ideas about JDO.
- You'll find hundreds of articles about every aspect of Java programming in the *developerWorks* Java technology zone.

About the author

Jeff Gunther, a Studio B author, is the General Manager and founder of Intalgent Technologies, an emerging provider of software products and solutions using the Java 2 Enterprise Edition and Lotus Notes/Domino platforms. Jeff is an application and infrastructure architect with experience in architecting, designing, developing, deploying, and maintaining complex software systems. His diverse experience includes the full life-cycle development of software running on multiple platforms, from Web servers to embedded devices. He has been a part of the Internet industry since its early, "pre-Mosaic" days. You can contact Jeff at jeff.gunther@intalgent.com

Code e-mail it!
