TechnologyCompatibilityKit

The JDO 2.0 Technology Compatibility Kit

About the Technology Compatibility Kit

The JDO Technology Compatibility Kit (TCK) tests that a JDO implementation is in compliance with the standard. This page contains information for developers of the JDO 2.0 TCK.

TCK ToDo

This is the list of tasks required to complete the JDO 2.0 TCK. We welcome new contributors! If you wish to contribute, please send email to jdo-dev@db.apache.org. See Apache JDO Mailing Lists for subscription information.

Activity wiki page	Description	Who	Expected Completion Date
TestRunner	Rewrite maven.xml so that the same TCK tests can be run in multiple configurations. For example, the same TCK test program needs to be run with and without security turned on, and with application identity and datastore identity. When we add different mappings for Chapter 18 (ORM) tests, the same test will also need to be run with different mappings.	Michelle Caisse	6/10/05 done
XMLMetadata	Develop xml metadata tests (Chapter 18)	Michelle Caisse, Michael Watzek & others	?/05
QueryTests11	Finish JDO 1 TCK query test classes	Michael Bouschen	done March 05
QueryTests	Write tests for the query language enhancements and new query api's	Michael Bouschen, Michael Watzek	?/05
Detached Objects	Design and write tests for detached objects	Matthew Adams	?/05
–	Rewrite examples in Chapter 15 to use the Company model; associate each example with a specific test case via assertion conditional text.	Craig Russell	?/05
RunRules	Write the rules vendors must follow in running the TCK to demonstrate compliance with the specification	Craig Russell	9/05
–	Complete the list of assertions from the Proposed Final Draft, create wiki page listing assertions needing to be implemented	Michelle Caisse	9/05 done
AllTheOtherTests	Tests of new JDO 2 features, other than query or metadata	Karan Malhi, others	?/05
TechnologyCompatibilityKit	Write a description of how to write a test case	Michelle Caisse	9/05 in progress

About the Testing Strategy

There are several variables involved in testing a relational JDO implementation:
*the database schema,
*the data itself,
*the database vendor,
*the object model, and
*the mapping. A single schema, data set, object model and mapping will be used to test conformance for as much of the specification as is possible. For those areas that aren't covered by this master variable set, the TCK will include variations on the master variable set for testing.

It is a design goal of the TCK to reduce the number of permutations of these variables. In order to minimize the number of database schemata, the TCK aims to generate, via a DDL generator, any database schemas required. It is not expected that the first version of the JDO 2.0 TCK will include such a generator. Until such time, database schemata will be maintained manually. In an effort to eliminate the number of vendor-specific database products, SQL92 will be used to express the database schemata, and implementations will be able to test their conformance against any number of database vendors' products. We have not yet determined to what degree variations from the TCK's schemata will affect conformance test results.

How to Develop a Test

These instructions assume that you already know how to check out the JDO repository. See SubversionRepository for more information.

1. Choose an assertion or group of related assertions to test (see JDO TCK Assertions, below).
2. Choose a package, new or existing, for the tests. If you are uncertain where to place the test, raise a discussion on jdo-dev.
3. Decide which assertions belong in a single test. Group up to about six related assertions in one test class, if appropriate. For example, a setter and getter pair should generally be tested together, with separate assertions for each within the test case.
4. Choose a name for the test. Use a descriptive name in camel case. Browse the existing tests for some examples.
5. Obtain the test template, copy and paste to a file in the appropriate package, and replace the placeholders:
   PACKAGE - The name of the package below org.apache.jdo.tck in which this test is placed.
   TEST_SUPERCLASS - The name of the class that his test class extends
   TITLE - A descriptive title for this test
   KEYWORDS - A list of keywords describing the content of this test
   ASSERTION_ID - The assertion id as listed in the assertions spreadsheet
   ASSERTION_TEXT - The assertion text as listed in the assertions spreadsheet
   TEST_NAME - The name of this java test
   PC_CLASS - The name of the persistence capable class instantiated. If none, delete localSetUp(). If more than one, add additional addTearDownClass(PC_CLASS.class) invocations.
   6. Decide which test superclass to extend for this test. If your test belongs to a package with its own superclass, use it. Check what class other test classes in the package extend. Otherwise extend org.apache.jdo.tck.JDO_Test. If you are starting a new package, consider whether there are methods or fields that you should factor into a new class which you would extend.
6. If your test requires instantiating a pc class, choose one or more existing persistence capable classes from org/apache/jdo/tck/pc/*. If the test requires simple pc data, use mylib.PCPoint and/or mylib.PCRect. If the test requires a complex model with relationships, use one or more classes in the Company package. If no existing pc classes are suitable for your test, see Writing a Persistence Capable Class, below.
7. Write the test (see Guidelines for Writing Test Code, below).
8. If the test requires a new mapping or test data, provide a schema file, mapping file, xml test data, and a configuration file (see Configuration Files, Schemas, Metadata, and XML Test Data, below). Also, add an entry for your configuration file to test/conf/configurations.list. Otherwise, write a temporary configuration file for debugging and add an entry to alltests.conf for this test. The temporary configuration file looks like this:

```
jdo.tck.description = Run one test for debugging
jdo.tck.testdata =
jdo.tck.standarddata =
jdo.tck.mapping = 0
jdo.tck.classes = org.apache.jdo.tck.your_package_and_test_name
```

9. If the test requires new metadata for an existing mapping, add it to the .orm and .jdo files that correspond to your pc classes. Update the files for both application and datastore identity. Elements described in the orm dtd should be added to the .orm file. Otherwise, add them to the .jdo file. See Configuration Files, Schemas, Metadata, and XML Test Data, below, for more information on file names and locations.
10. Install the database schema for the test. If you are using the standard schema (jdo.tck.mapping=0), schema installation takes about 15 minutes. However, it only needs to be repeated after changes to the schema.

```
maven -Djdo.tck.cfglist=myConfig.conf installSchema
```

11. Execute the test with

```
maven -Djdo.tck.cfglist=myConfig.conf runtck.jdori
```

12. Debug the test.
13. Run svn add for any new files you have created for check-in. Do **not** add your temporary config file, if you needed one.
14. Execute the entire test suite to verify that your changes have not created any regressions.

```
maven build
```

15. Create a patch and submit to jdo-dev for review. From the tck20 directory, do:

```
svn diff > myPatch.patch
```

16. Make changes as agreed upon by the community. When all changes are complete, submit the patch for check-in.

17. Update the assertions spreadsheet with your test information. Change the Implemented column to "yes" and add the test path and name to the Test column.

JDO TCK Assertions

The JDO team went through the JDO specification and identified assertions the JDO TCK needs to test. An assertion consists of an assertion number and the corresponding text of the JDO specification describing what the test class needs to check. Usually an assertion is implemented by one JDO TCK test class and the test class covers one assertion. There is an assertion spreadsheet listing all the assertions with their number, text and whether they are already implemented in the current version of the JDO TCK. This assertion overview is organized in multiple sheets, one sheet per chapter of the JDO specification. You find the current version of the spreadsheet below.

Assertion Number

An assertion number (e.g. A14.6-15) consists of the section number of the JDO spec followed by a consecutive number. Please note, a few chapters in the JDO 2.0 spec have different numbers compared to JDO 1.0. We decided to keep the assertion numbers as defined for JDO 1.0. So in the following cases the assertion number does not match the chapter number of the JDO 2.0 spec:

Chapter	Assertion	JDO 2.0
Extent	A15	19
Enhancer	A20	21
Interface StateManager	A21	22
JDOPermission	A22	23

Current Spreadsheet Version

Download the current version of the assertions spreadsheet. It is in Star{{`Office/Open}}`Office format. This version adds new assertions to the query chapter covering JDOQL extensions in JDO 2.0. Furthermore it updates the assertions about JDOPermissions.

Guidelines for Writing Test Code

Use the coding standards of the Geronimo Project.

See the annotated SampleTest for pointers on how to write test code.

Cleanup

Each test in the TCK must leave the datastore clean when it exits.

The TCK uses the JUnit testing framework. JUnit encourages test classes to separate the real task to be tested from testing environment setup, relatively testing environment cleanup. TCK classes follow this implementation strategy.

For this reason, all TCK test classes extend abstract class "org.apache.jdo.tck.JDO_Test". This class provides two hooks that subclasses may override for test environment setup and test environment cleanup:

*protected void localSetUp()
*protected void localTearDown()

TCK classes usually set up persistent data in method "localSetUp" and they cleanup that that data in method "localTearDown". The real testing tasks are implemented in methods having the prefix "test".

Class JDO_Test implements a default strategy for "localTearDown": All persistent data that has been added for tear down is cleaned up automatically. Thus, TCK classes usually do not override "localTearDown".

JDO_Test defines three methods adding persistent data for tear down:

*protected void addTearDownObjectId(Object oid)
*protected void addTearDownInstance(Object pc)
*protected void addTearDownClass(Class pcClass)

The first two methods may be used to add single persistent instances for tear down. Method "addTearDownInstance" is convenience delegating to addTearDownObjectId. The last method may be used to add persistent classes for tear down. In the latter case, the extents of all added classes are deleted.

Note: The order of adding tear down instances and classes is significant. The default implementation of "localTearDown" first deletes all added instances in exactly that order they have been added. Afterwards it deletes the extents of all classes in exactly that order they have been added.

SQuirreL is a useful tool for checking the contents of tables in your database. See SquirrelSqlClient for more information.

Configuration Files, Schemas, Metadata, and XML Test Data

Read this section if you are writing a test that requires xml test data (like org.apache.jdo.tck.mapping.CompletenessTest) or requires a new mapping and schema.

There are a number of files in addition to the Java test class that must be present for a test to run.

File type	Path	Purpose
configuration	test/conf/*.conf	Associates mapping and test data with a test class.
schema	test/sql/*database*/*identitytype*/*.sql	Contains SQL DDL commands for generating the database tables in which persistence capable instances are stored.
orm metadata	test/jdo/*identitytype*/*package*/**.orm	Provides the mapping metadata required by the JDO implementation for mapping fields of pc instances to datastore entities. See Chapters 15 and 18 of the JDO 2.0 specification for more information.
test data	test/testdata/*package*/**.xml	Provides data (field values) for pc instances read by the Spring Framework used in some tests in the query package and mapping. CompletenessTest, which tests XML metadata.
jdo metadata	test/jdo/*identitytype*/*package*/**.jdo	Provides the metadata required by the JDO implementation for persisting pc instances. See Chapter 18 of the JDO 2.0 specification for more information.

Each invocation of maven is driven by a list of configuration files. If you do not set *jdo.tck.cfglist* on the command line, maven will read the list of configurations from test/conf/configurations.list, installing the schema and running tests for all configurations. Typically when you debug a test, you will specify a single configuration on the command line. These properties are set in a configuration file:

*jdo.tck.description - A text description of the purpose of this test configuration
*jdo.tck.testdata - The full path below test/testdata to the xml test data file
*jdo.tck.standarddata - The full path below test/testdata to the xml standard data file, used in cases where the test modifies the data before persisting it; the retrieved data is then compared to the standard data, not the original test data
*jdo.tck.mapping - A number that specifies both the orm mapping file and schema file to be used for this test (more below). A value of 0 is used for the standard schema and metadata files.
*jdo.tck.classes - The fully-specified name of the test class to be executed

The default schema file is named schema.sql. There is one schema.sql for each database and identity type. It creates database tables for all of the pc classes used in the tests. In order to test different mappings, some tests require a different schema. These are named schema_N_.sql, where *N* is the number assigned to the jdo.tck.mapping property. Each schema file first creates and sets and sql schema specific to the identity type and jdo.tck.mapping value. For example:

```
CREATE SCHEMA applicationidentity3;
SET SCHEMA applicationidentity3;
```

Each schema file must also drop all database entities before creating them to ensure that the schema will be clean at the start of each test run.

The default orm metadata file is named package-*database*.orm or *classname*~~*database*.orm. There is one .orm file for each database and package or class. Configurations that use a different mapping for a particular pc package use an orm metadata file named package~~*databaseN*.orm, where N is the jdo.tck.mapping value described just above. In this way, there is a one-to-one association between a schema_N_.sql file and a package-*databaseN*.orm file.

Provide a test data file for any new configuration that uses the Spring Framework methods to compare a persisted object graph to a standard. Currently the CompletnessTest and some tests in the query package use the Spring Framework to create an object graph from xml data. If the test modifies the test data before persisting it, you must provide a standard data file which specifies the object graph that will match the retrieved data.

Unless you write a new persistence capable class, you do not have to create a new jdo metadata file. The correct jdo file is found based on the class name, the identity type under test, and rules for file naming and location described in the specification.

Writing a Persistence Capable Class

(content to be provided)