TestRunner

About TestRunner

An official TCK run requires running all relevant tests in all configurations that are supported by the Implementation Under Test (IUT). For example, if the JDO implementation supports application identity and datastore identity and supports MySQL and Derby, then the qualifying TCK test run must test the entire configuration list (less automatically excluded tests) against application identity and datastore identity for both MySQL and Derby. When we add different mappings for Chapter 18 (ORM) tests, the same test will also need to be run with different mappings.

While developing the JDO implementation and removing bugs, it is useful to run a subset of configurations. This should be easily supported as well. For example, it should be easy to run just a few tests against application identity on MySQL.

This project involves rewriting maven.xml so that the same TCK tests can be run in multiple configurations.

Requirements

1. The TCK development team must be able to define the following configuration parameters for each TCK test:
   *Test description string
   *Test class
   *Test data (both input test data and, in some cases, a standard to compare to after updates)
   *Security on/off
   1.#2 Developers must be able to run the TCK on the JDORI (JPOX) out of the box, and later on the IUT, without changing anything, simply by invoking a specific maven goal.
2. The JDO implementation team must be able to run a subset of the above configurations in various environments: *Application identity / Datastore identity
   *Mapping file (.orm) (different databases need different .orm files)
   1.#4 To configure a test run, users should be able to set all configuration parameters in a text file.
3. The user must be able to specify either a single test or a list of tests to be run with a given set of configuration parameters.
4. The user must be able to specify multiple configuration settings within one configuration file, each to be run with one or more test cases.
5. Each test execution must report the full set of parameters, as listed above.

Functional Specification

The proposed version of maven.xml runs the TCK on all databases and identity types specified in the properties jdo.tck.dblist and jdo.tck.identitytypes. It runs the tests specified in jdo.tck.cfglist. Properties are set either in project.properties, in configuration files or on the maven command line. There are no separate goals for different configurations.

Custom Goals

These are the major custom goals of maven.xml:

*runtck.jdori - runs the TCK on the JDO Reference Implementation
*runtck.iut - runs the TCK on the implementation under test
*installSchema - installs the database schema
*enhance.jdori - enhances the class files using the JDO RI enhancer
*enhance.iut - enhances the class files using the implementation under test's enhancer

Configuration Files

Maven looks for the following configuration files in test/conf:

*configurations.list
A list of files. Each file listed is a test configuration
file.

*test configuration files
Each of these files sets values for
*jdo.tck.testdescription - an optional string describing the purpose of these tests
*jdo.tck.classes - a list of one or more test classes.
*jdo.tck.testdata - fully qualified file name (not required by all tests)
*jdo.tck.standarddata - fully qualified file name (not required by all tests)
*jdo.tck.mapping - file designator that maven.xml uses to build a javax.jdo.option.Mapping value and corresponding schema name

*exclude.list
A list of test classes NOT to execute during a TCK test run

Questions:

1. Currently we have two properties in project.properties that can be used to specify the implementation's support for application or datastore identity. Should we use these instead of the identity.list file, allowing override only on the command line? Craig thinks that we should use the "jdori" entries when running the jdori maven goals, and use the "iut" entries when running the iut maven goals.

Command Line Options

-Djdo.tck.cfglist=<configuration file list>
Overrides test/conf/configuration.list by supplying one or more space-separated test configuration files
-Djdo.tck.dblist=<database list>
Overrides the property value in project.properties by supplying one or more space-separated database names
-Djdo.tck.identitytypes=<identity type list>
Overrides the property value in project.properties by supplying one or more space-separated identity types (applicationidentity or datastoreidentity) to use for this run.

Databases

The property jdo.tck.dblist contains a list of databases supported. The property is set in project.properties and can be overridden on the maven command line.

Identity Types

The property jdo.tck.identitytypes contains a list of identity types to be run. The property is set in project.properties and can be overridden on the maven command line.

build and rebuild Goals

These goals do not recognize the command line options. They do a full build and run the TCK on the JDORI for all configurations, databases, and identity types listed in the configuration files. [Is this a good idea? Craig says yes, but can it be implemented in time? Michelle responds, sorry, now that we use the command line to override the properties set in project.properties, it is not possible to know where the property was set, so the build and rebuild goals will behave like the custom goals with respect to command line options.]

Schema Installation

Schema installation takes a significant amount of time and it is impossible to "cheaply" determine if a schema is up-to-date. By default, maven.xml installs the schema for all databases, identity types, and mapping values set for the current invocation only during a full build (maven build or rebuild goals). The user may override this behavior by adding the installSchema target to the command line with the runtck.jdori or runtck.iut goals to force schema installation.

Packaging

The files needed for test execution are copied into four directory trees:
*target/enhanced/jdori/applicationidentity/
*target/enhanced/jdori/datastoreidentity/
*target/enhanced/iut/applicationidentity/
*target/enhanced/iut/datastoreidentity/

Each of these trees contains:
*enhanced test classes
*jdo metadata files
*orm metadata files
*xml test data files

Currently, the jar files are created and then the directory tree is deleted. However, the classes have to be re-enhanced every time an orm or xml file is modified so that the jar file can be recreated. To avoid this, we will not delete the tree.

Mapping and Schema Names

There may be multiple orm files for a single persistence capable (pc) class or package. The orm files are distinguished by different mapping designators in their filenames. For example, a given test configuration may run a set of tests on databases derby and mysql with mapping 4. The mapping metadata files are package-derby4.orm and package-mysql4.orm; the schema files are derby/schema4.xml and mysql/schema4.xml. maven passes javax.jdo.option. Mapping=<database + mapping designator> to the test, for example javax.jdo.option.Mapping=derby4.

An invocation of maven installs all schemas into the database at once, each into its own named schema. The schema name is hard-coded into the .sql DDL file and is named by the identity type and mapping designator, for example applicationidentity4. The schema name may be specified either with the <orm> schema attribute or with the javax.jdo.mapping.Schema property. To avoid having to add the schema information to each orm file, the property to specify the schema is passed to the test on the command line.

Test Data

Test data files are xml files that define data values for one or more instances of one or more classes in a packages. The data is used to create in-memory data that tests may persist and update. If the test does not perform any updates, the test compares the retrieved persisted data to a new instance of in-memory data. If there are updates, the test compares the retrieved data to an instance of a standard given in a second xml file. The property jdo.tck. testdata is the test data file and jdo.tck.standarddata is the standard.

Security

Support for security will be implemented later.

Exclude List

The exclude list is passed to the java test runner. If a class is included in the exclude list, the class will not be executed by any test configuration.

Design

Flow Control

The language elements that control the looping and decisions in maven.xml are from the jelly core and util packages (http://jakarta.apache.org/commons /jelly/tags.html, http://jakarta.apache.org/commons/jelly/libs/util/tags.html)

Organization of Maven Goals

There are two design changes from the previous version of maven.xml:

1. Each goal that requires iteration has setProperties as a prerequisite goal. setProperties parses the command line and/or configuration files and sets properties.
2. Each goal that requires iteration contains one or more forEach loops. The inner loop invokes another target that does the actual work.

Legacy Discussion

Below are the ideas we discussed before settling on the current solution.

Role of build.xml

Because classpath and system properties must be set before the test runner is invoked, build.xml must play a role in the implementation. Michael will investigate whether the build process can read from the configuration file so that all parameters can be set in one location, or whether the user will have to set some parameters in build.xml. This affects identity and security settings.

Test Runner and Identity Setting

Today the JDOTCK enhances the persistence capable and persistence aware classes into its own directory called enhanced. The current set of JDO metadata files is copied to a directory called metadata. These two directories are in the beginning of the classpath to make sure that we always find the enhanced version of the pc class. Under tests we have different metadata files for application identity (xxx.jdo.a) and datastore identity (xxx.jdo.d). The targets useapplicationidentity and usedatastoreidentity copy the metadata files into the metadata directory and skips the suffix .a or .d. When switching from one identity setting to another you need to clean the enhanced classes, copy the metadata and enhance the pc classes.

This does not support easy switching the identity setting. The idea is to keep the enhanced class files for application and datastore identity in separate directories together with their metadata. During JDOTCK setup we would enhance for both identity types and keep the result in separate directories. Then configuring the identity setting for a test class (or a list of test classes) would be setting the correct classpath for the test run.

Lack of support for application or datastore identity

The user will be required to configure the build to specify the implementation's support for identity types. This will prevent test failure due to lack of support.

Open Questions

*What about nondurable identity? Currently, nondurable identity is not supported by the RI, so we cannot verify that the TCK runs properly with it. This situation is not likely to change with JDO 2.0.