# WORKOUT Task List

Revised: Oct 16, 2013

## Overview

WORKOUT is a sample of my work, where I designed and coded the project myself. The original WORKOUT was C code I wrote for someone else (so I can't put it on my web site), but the Java version was for a project that never got off the ground. I took the original designs, and rewrote the code as this standalone suite. I wanted to create a fairly large project that demonstrates my designing, coding, scripting, build process management, documentation, web programming, and so on.

– Scott McMahan

WORKOUT is a test program which does a lot of SQL, intended as a way to test a SQL monitoring package. The design centers on commands to create, update, and dump a set of tables with referential integrity. WORKOUT has a fairly simple set of commands, with the intention of being embedded in a scripting language for control structures.

- *OUTCOME*: To create a large sample of my work with full source code available.
- *PURPOSE*: To show people that I am able to design, specify, and implement a large, cross-platform project.
- *ACTION*: This large project will be implemented in pieces, following the tasks in this list. Because many of these tasks are quite large, some have been placed in "extended" categories which will be done as time permits.

Completed items are indicated by strikeout. Gray items are for the future. Light gray "invisible" items are things I have reduced in importance.

**Design note**: Because this is an example of my work, the design is malleable. This project could go in several directions. As of February 2013, I have decided to concentrate on finishing the core Java parts at the expense of some of the other parts.

**Implementation note**: This project is *huge*. I would need about six to eight months to complete it, if I worked on it full time. I can't do that. In the hopes that *something* is better than *nothing*, I have decided to implement the Oracle version first, and then worry about other databases later when the project is closer to having the complete code functionality implemented. Adding new databases is a huge time sink since I have to get the JDBC drivers set up, concoct DDL for that database, and so on—I'm not going to do all that until the code is farther along, because this is supposed to be a code sample of my work, not a sample of me adding jar files to my build process and writing DDL. I picked Oracle because I already had the table DDL written.

> Because of increased use of SQL Server in the software world, I decided to go ahead and implement support for that database. I can run command-line programs against either SQL Server or Oracle by changing one attribute in `build.xml`.

*SQL Server support: I have a Windows 2012 server in a VirtualBox virtual machine which runs an instance of SQL Server 2012. (I had this same configuration with Windows 2008/SQL Server 2008 and migrated to the 2012 versions.) The Java code connects to the SQL Server instance using Microsoft's JDBC driver. I also have a Windows 8 laptop running the Management Studio so I can more easily administer the instance than I could from a virtual machine console.*

Another issue is that **I am my own system administrator**. I have limited time to work on this example code, and often my time is spent more on system administration than actually working on code. I have to install and configure everything I use from scratch. A real-world project would not require as much administration overhead as this sample does, since it would be unlikely to include multiple databases, multiple server operating systems, and so on. I often spend 10x the amount of time setting up my development environment than I do actually developing.

Note on **ant vs Maven**: I like ant and have become comfortable with its power and flexibility. A lot of projects use Maven now, which is similar to ant. Maven seems to have been designed for build environments on platforms which do not have package managers to install software, so Maven has the ability to download common open-source frameworks. I use Fedora, which already has these frameworks in its package manager, so I don't see the same benefit.

Note on **POJO vs annotations**: Some projects use annotations in the code itself, and some use POJOs (plain old Java objects) when it comes to constructing beans. Each approach has benefits and drawbacks. The reason why I use POJO bean classes in this sample is because I want the ultimate flexibility to use the same beans with different frameworks, since this is an open-ended example that can be adapted to different frameworks. I do not want to use any particular framework's annotations in my Java code for this example. (A secondary, and compelling, reason to use POJOs is that this Java enterprise stuff is in a constant, overwhelming state of change and I'd rather rewrite some XML files than rewrite my code.)

## *Milestone releases*

- **0.1 (initial release)** – has working Java core classes and multi-threaded command interpreter, and can run both workout and Jython scripts
- **0.2 (second milestone)** – includes core functionality in Java, support for both SQL Server and Oracle, and a finished Tomcat configuration with both SQL Server and Oracle connectivity
- **0.3 (third milestone)** – includes a servlet that runs one workout command, and a Python test program to drive the servlet – this is the basic core of the scriptable interface for command-line programs, mobile apps, and AJAX – this is more to get all the plumbing working (installing a servlet in Tomcat, having a basic servlet skeleton, etc) than a final servlet design
- **0.4 (fourth milestone)** – implements the beans that hold each row in the table, a row bean that holds a complete set of row beans for one key, a LiteORM class to retrieve the rows of the table as beans, and a test program that uses the LiteORM class (this sample is a building block towards using a heavyweight ORM like Spring or Hibernate; it's also an example of me writing a three-table join in SQL)
- **0.5 (fifth milestone)** – fixed the first bug I've found in this code (forgot the `public` keyword on a method), added a Jython script to test the LiteORM class, redid the Jython script runner to

pass a database connection and the name of the database to the script (so the script no longer has to create a database connection itself, which will be important later on).

- **0.6 (sixth milestone)** – created JSP that uses EL and standard JSP tags to iterate through the tables and create HTML, and added the ability for the LiteORM class to be used as a bean from a JSP page
- **0.7 (seventh milestone)** – added Spring support to the sample, because Spring is an important framework; added generators and wrote a control program to run them with a Spring dependency-injection (a "generator" is a class that creates a sequence of rows for the workout table, which is run by a control program which Spring injects with a generator)
- **0.8 (eighth milestone)** – created a generic generator and a sequence interface to split the two, and added an example of how to wire a sequence to a generator using Spring
- **0.9 (ninth milestone)** – added two Visual Studio projects in C#, a DLL library with core functionality classes and a command-line driver which uses the classes. I wanted to take my existing WORKOUT code and get it into a DLL, and have a command-line driver to run it. I also wanted to add Visual Studio projects to my script that builds the web version of this code. From here on out, doing new VS stuff will be a lot easier with this groundwork in place.
- **0.10** (tenth milestone) – added an example of using Spring's JDBC template to dump the tables instead of using "raw" JDBC. I had a several month break between milestones because I was working on another project. This workout sample is a catch-22 because I want it to be an example of professional-quality code that I've constructed, but doing so takes a lot of time. I did not have time to get this code working with both Oracle and SQL Server.
- **0.11** (eleventh milestone) – added a Hibernate example.
- **0.12** (twelfth milestone) – added an example of C# 5 and .NET 4.5 threading with async/await, building on the older Windows code base. The original file interpreter was single-threaded and essentially concatenated all of the input files together that appeared on the command line. I wanted to experiment with making the Windows code base multi-threaded, so I used the new C# async/await. The code was developed on Windows 8 using Visual Studio 2012. Because of the changes, and the new dependency on C# 5 and .NET 4.5, I put the code in new packages.
- **0.13** (thirteenth milestone) – added basic web-based workflow, without any frameworks, as a scaffolding to design and build later web workflows. Also added jQuery and jQuery mobile to the web app. Also created a basic (no frameworks) workflow using jQuery mobile and tested it on mobile devices. This release is setting up the basis for more advanced workflows in the future.
- **0.14** (fourteenth milestone) – major milestone: Added Spring jars to my J2EE webapp in Tomcat; and added jQuery and jQuery mobile to my webapp. Created a table dump workflow in plain JSP scriptlets with JDBC to have something to work with. Continued bootstrapping with a plain JSP mobile interface. Then added a Spring MVC workflow to dump the tables. Then added a mobile interface for the workflow.
- **0.15** (fifteenth milestone) – added a local Hadoop instance and wrote a trivial map-reduce example to make sure it worked. Being my own system administrator means I have to install and configure something like Hadoop (which is not currently part of the Fedora package manager) from scratch.
- **0.16** (sixteenth milestone) – added a sample table dump program for Windows using Entity Framework (instead of ADO.NET).
- **0.17** (seventeenth milestone) – set up JUnit in the build process and added a sample test. I would like to have a lot of tests, but I don't have time to write the code I want to write, much

less the kind of comprehensive test suite this project needs.

- **0.18** (eighteenth milestone) – added Hibernate samples. I'm preparing for two efforts. First, I want to recreate the Hibernate examples using JPA and annotations (although my personal preference is for POJOs, I see annotation-based persistence objects a lot), and have a sample of using them from Spring. I also want to set up SVN and Jenkins, and add more unit tests that Jenkins will run automatically. Doing all this will take a while, especially since I'm my own system administrator and have to set things up from scratch.
- **0.19** (nineteenth milestone) – added a jQuery DataTable example which displays rows in the parent table. When a row is clicked, the page makes an AJAX call to get child rows, and shows them in a jQuery dialog.
- *next* – My goal is to have an end-to-end sample of my work with a complete web application from the database back-end through the presentation layer. Next I need an example of using Spring or Hibernate (or both) from within the Spring MVC workflow rather than straight JDBC. My goal is to take the table dump workflow, and add a way to run workout commands from the web interface. I need a controller that will accept command input and run the command, and then read output and direct it to a view. Then I need an AJAX way to call this controller and read the output and display it. I would also like to have a way for native mobile apps to run commands and do table dumps, using either a servlet or a controller that will be called from AJAX and return JSON so the native app can handle screen presentation.

## *Summer 2013 Tasks*

- ~~jQuery – download and set up latest version~~
- ~~jQuery Mobile – download and set up latest version~~
- ~~Create basic dump workflow~~
  - ~~display keys in W1 table – page 1~~
  - ~~select key to see row data – navigate to page 2~~
  - ~~select W2 or W3 key to see row data for the keys in those tables – page 3 (can display either W2 or W3)~~
- ~~Create mobile version of basic dump workflow~~
- ~~Get Spring working inside Tomcat web app~~
- ~~Spring MVC~~
  - ~~servlet configuration~~
  - ~~controller setup~~
  - ~~views~~
  - ~~table dump workflow~~
  - ~~mobile interface to table dump~~

On a higher level, I'm trying to decide how far to go with my example of database access through Spring and Hibernate. There's Spring, there's Hibernate, there's Spring and Hibernate together (injecting a Hibernate session factory using Spring), there's POJO-style Spring, there's POJO-style Hibernate, there's annotations-style Spring, there's annotations-style Hibernate, there's JPA, and so on. The number of permutations of these similar-but-different technologies makes me wonder at what point I've demonstrated I can do database access through these frameworks, and what's beyond the point of diminishing returns as far as my time investment. I'm always willing to adopt whatever variation on these technologies is best for any specific project. My limited time, however, may be better spent on something else. I have this same issue with web frameworks, which are similar-but-different technologies. There's Struts 1, Struts 2, Struts 2 with Java Server Faces, either Struts 1 or Struts 2 with jQuery, Spring MVC, and so on. I don't think I could possibly include sample code with all of these technologies in my example project.

## Documentation set

- ~~Scan of original plan~~
- ~~Task list (this document)~~ (under constant revision as I complete features)
- ~~WORKOUT commands specification~~

## General

- ~~I need a place on my web site to put all this stuff and some way to create a tarball (or ZIP file) with the source~~
- ~~Java package~~ and C# namespace will be `net.scottmcmahan.workout`

## Java core classes tasks

- I need my own local SVN repository for this – set it up and set up a backup mechanism (and, of course, a way to strip out all the `.svn` stuff for the Internet upload)
- Get MySQL JDBC driver
- ~~Get Oracle JDBC driver (doesn't this come with Oracle?) - yes, it comes with Oracle, but my samples are in a state of rot because they changed the JDBC driver since the last time I used it – I had to create new samples for Java and Jython~~
- ~~Get MS SQL Server JDBC driver (does MS provide one, or is there an open-source implementation of the T-SQL tcp/ip protocol?)~~
- ~~Set up project directory~~
- ~~Set up a JAR file directory~~
- ~~Set up a directory for SQL~~
- ~~Start files with DDL for each database~~ – we will defer MS SQL Server for now
- ~~Start a Java build process using ant for the new dirs~~
- Configure MySQL database
- ~~Configure Oracle database~~
- Skeleton test program to connect to MySQL to make sure JDBC works
- ~~Skeleton test program to connect to Oracle to make sure JDBC works~~
- ~~Get a run process set up to run Java programs in this project~~
- ~~Set up build to generate javadoc for classes~~
- Create a custom exception my classes can throw to disambiguate their exceptions from standard Java ones (not sure if I am going to do this?)

- ~~Create beans to store the three-number WORKOUT rows and test driver~~
- ~~Create subclasses for rows from the three tables (this will be important later so we can be unambiguous about which list of numbers is from which table)~~ I decided not to do it this way. I created each bean as its own class. Subclassing is unnecessarily complex here.
- ~~Create a Java class with implementations of the WORKOUT commands (stub it out first)~~
- ~~Create a Java class to parse a line with a WORKOUT command~~
- ~~Create command-line program to run one WORKOUT command~~
- ~~Create command-line program to run a script file with WORKOUT commands~~
- ~~Create command-line program to use multithreading to run a group of scripts in parallel~~
- ~~Get latest Jython interpreter and install its JARs~~
- ~~Create command-line program to run Jython script instead of just WORKOUT commands (could be done by extending command-line programs that run scripts)~~

## *Spring tasks*

A *generator* is a Java class which generates a sequence of rows for the workout tables, using the LISP idea of "lazy evaluation" where the *next* number in the sequence is created only when it is needed. A control program runs a generator, which is injected at runtime using Spring, so different generators can be run without code changes. To make this as flexible as possible, generators generate workout scripts which can be run by the script runner (possibly after being edited either by a person or an automatic process).

- ~~Set up project to use Spring by putting JARs and other dependencies in build.xml~~
- ~~Create generator interface~~
- ~~Create sample generator~~
- ~~Create generator runner~~
- ~~Create build target to run generator and generate a script~~
- ~~Create build target to run script emitted by generator~~
- ~~Create dump program that uses JDBC template~~

## *Java J2EE tasks*

- ~~Set up directory tree so Tomcat can run in a sandbox for this project~~
- ~~Get all the JARs in place so Tomcat can use them (for databases, WORKOUT, etc)~~
- ~~Simple JSP page to dump table~~
- ~~Issue WORKOUT commands via Servlet~~
- Need a servlet that will run either a file of commands or a Pyhon script – this will be the basis for other servlets and functionality
- Dump tables via Servlet
- Create a standard Servlet for mobile apps to use to talk to the server and issue WORKOUT commands / dump tables – this piece may need to respond using something like JSON or an

easily-parsed format

- Use ant to build a WAR file for this project in case it needs to run in some other J2EE server (like WebSphere)

## *Extended J2EE tasks*

- ~~Workflow example: Add Spring MVC~~
- Workflow example: Add Struts 1
- Workflow example: Add Struts 2
- Consider adding DB2 database support (which wouldn't be hard, since I already had this code working with DB2 at one point, but I'm not sure I want to administer a DB2 instance in addition to everything else)

## *Extended other Java tasks*

- ~~Fedora – get Hibernate JAR files on my system from repository~~
- ~~Create a command-line program that uses Hibernate to dump rows from WORKOUT tables~~

## *Android tasks*

- Write app to read rows into list of some sort using the adapter pattern, and display them
- Use normal Java code to get URL (I already have a sample of this)
- Write app to perform WORKOUT commands

## *iOS tasks*

- Write app to read rows from dump of WORKOUT tables into a UITableView, using table sections for RI sets
- Write app to perform WORKOUT commands
- Use AFNetworking to talk to Tomcat

In February 2013, I pulled the plug on further iOS development. I have not found anyone who wants me to do it. Everyone wants 3-5 years experience, and by the time I get that much experience, I will have bankrupted myself trying to keep up with Apple's development program and new devices. Besides, Objective-C is yet another language to learn, and if I don't use it for anything, there's no point learning it and forgetting it. I had even thought about using Core Data, but–again–this is yet another huge over-engineered framework to learn and immediately forget because I never use it. I had big plans for a mobile version, but can't justify putting any more time or money into something that isn't helping my career.

## *Windows tasks*

Summer 2013 note: I've pretty much given up on Windows development. When I talk to people, they want programmers with years of hands-on experience with .NET, and have *no interest* (basically doing everything but asking me why I'm wasting their time even talking to them) in anyone who understands how to model business workflows with software, knows several other and quite similar packages, and can pick up .NET easily. Apparently there is more supply than demand for .NET programmers right now. So I am going to stick to the Java technologies I know best.

- ~~Set up SQL Server database for WORKOUT~~
- ~~Write DDL for SQL Server~~ (well, the Management Studio wrote it for me)
- Write Windows 7 style desktop program to display dump using a data view bound to ADO.NET (i.e. do not use the WORKOUT class library)
- ~~Write C# assembly that implements the WORKOUT commands~~
- Finish implementing all of workout commands in the DLL – at first I just implemented the major ones
- ~~Multi-threaded program to run multiple input files – I created one using the new C# 5 and .NET 4.5, and Visual Studio 2012.~~
- Use XML-style documentation comments (`///`) and generate class documentation
- Create Windows 7 style desktop program to issue WORKOUT commands
- Create Windows 8 WinRT program to issue WORKOUT commands

Windows 8 is getting little traction in the computer industry so far. To put any more time into learning it, I would have to see a lot more interest and action with Windows 8 than I've seen so far. For now, I am dropping it along with iOS and concentrating on the Java stuff. I may still do the C# version since I have a lot of that code already written and would only need to polish it up. I need to create an assembly with the core code, and create a command-line driver program that uses the assembly. Then, the other components could use the assembly. Except the data display program, where I want to use C#'s data bindings and data controls to have an example of doing that – this would not use my assembly.

## *Windows extended tasks*

- Get ADO.NET drivers for other databases such as Oracle and support them (right now the Windows stuff supports only SQL Server)
- Write command-line dump program using LINQ to directly access WORKOUT tables using ADO.NET (i.e. do not use the class library)
- ~~Write EF program to dump tables~~
- Write EF program to create rows in the tables.

My bread and butter is Java. I have a lot of ideas for Windows, but I am not sure I can justify spending time on them. I'll do as much as I can.