QAT Global
Code Sample

ComplexMO

Introduction

The ComplexMO sample is, as the name implies, a Complex Model Object sample designed to show the reader some of the more complex capabilities of MyBatis and the the QAT Global framework. This document explains the sample and how to setup and configure Spring and MyBatis within the ComplexMO sample.

1.1 Pre-requisites

In order to extract the maximum benefit out this sample, it's expected that the developers have

- Undergone basic Java training
- Undergone basic Spring training – WebServices, AOP and SpringUnit
- Reviewed the SysMgmt sample
- Acquire the complexmo-* projects from SVN or local file server or tech-lead/instructor. If downloading from SVN these projects are normally located under the trunk/Samples folder: complex-mo-*

1.2 Setting up DB

Within the complexmo-test project locate the **setup** folder. Based on the local database you are using locate the appropriate SQL script and execute it using the appropriate database client tool. See your instructor for more information. Normally it will be postgreSQL.

1.3 Building the project

Note when you first open the projects and if Eclipse tries to build them the build will fail since all the dependencies cannot be resolved at this time until they are first built and published.

In addition you will see IVY errors related to not being able to resolve certain dependencies defined in the project classpath.

This is okay since all the various projects and dependencies have not yet been built so the first thing you will need to do is build the projects.

1.3.1 Building and publishing for the first time

In order for IVY dependency management to work we need to build the projects and then publish them to a local IVY repository which will be located on your development machine. This in turn enables the dependency management capabilities of IVY to be leveraged in order to build the rest of our projects.

Start with the complexmo-Interface project.

1. Open the build.ant.xml file located in the *build* folder.
2. Run this ANT script which builds the project source and publishes a JAR file to your local IVY repository which in turn will allow subsequent projects to resolve and build correctly.
3. Review the output and notice how the various dependencies are resolved.

Next open up the complexmo-implementation project

1. Open the build.ant.xml file located in the *build* folder.

2. Run this ANT script which builds the project source and publishes a JAR file to your local IVY repository which in turn will allow subsequent projects to resolve and build correctly.
3. Review the output and notice how the various dependencies are resolved.

If after doing this any of the projects are still failing to build under Eclipse then right-click on the project and select IVY➔Resolve
This will tell Eclipse to re-resolve all of the IVY dependencies and should clear-up any build issues.

You will notice the test project does not have an ANT build script. With this project all you need to do is perform a build under Eclipse for everything to work prior to executing any tests. You may have to perform an IVY Resolve as mentioned above.

1.3.2 Subsequent changes and building

When you make a source change make sure you **re-run the build scripts** so the most recent version of your project JAR is published to your local repository and the projects that depend on these changes point to the "latest and greatest". For example if you update the interface project make sure you re-run the build script so the implementation projects has or uses the "latest and greatest" version.

1.4 Run the sample

Using the complexmo-test project run all the tests. This can be done by right-clicking on the complexmo-test project and selecting Run As and then JUnit Test.

This will execute all the tests found in this project. The JUnit view will appear in Eclipse and all the tests should run successfully and show green on the JUnit view. If not then attempt to diagnose the problem and contact your tech lead or instructor.

1.5 How to use this sample

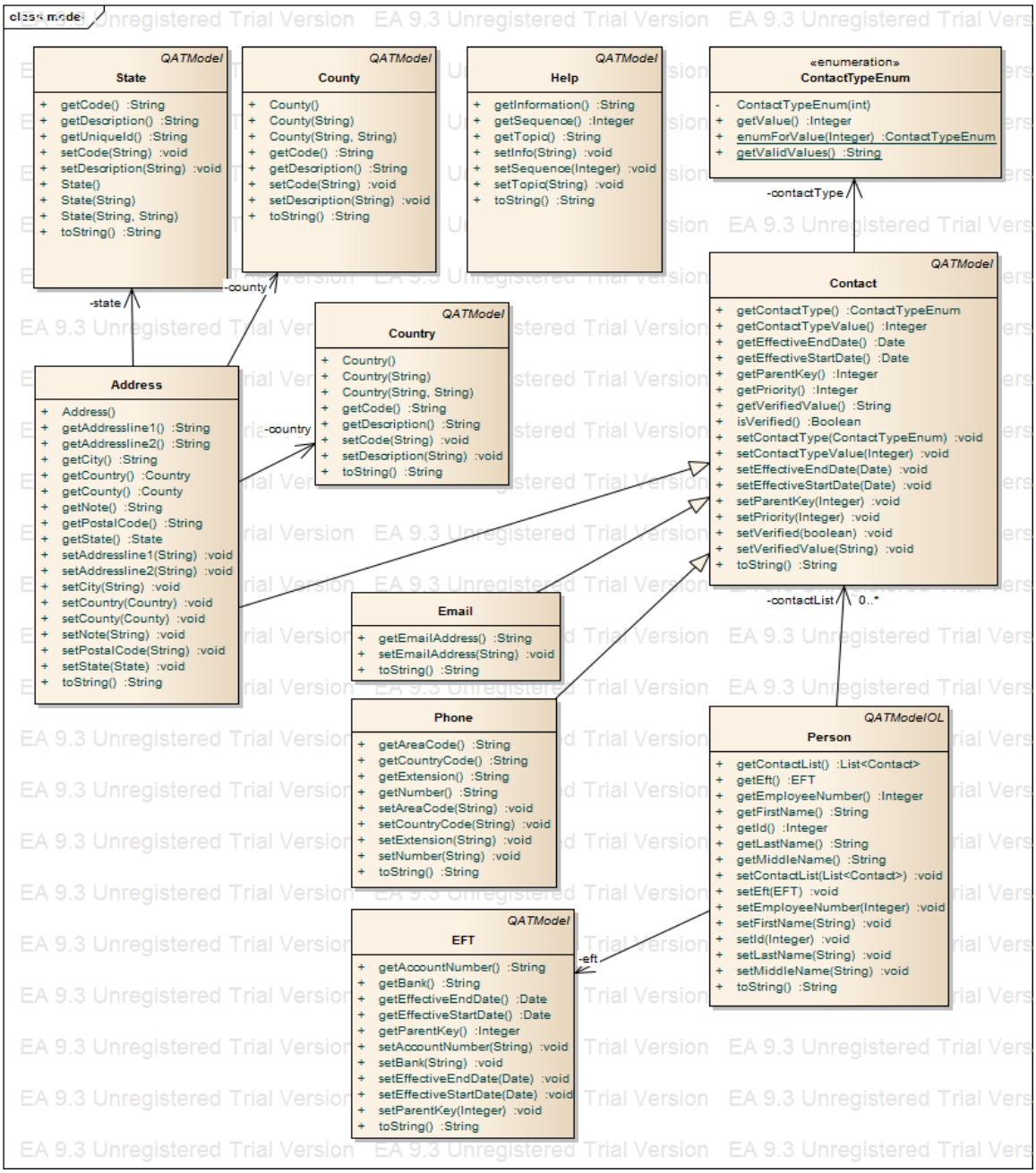
The source code and the output from the execution of the sample tests contain a tremendous amount of information that explains what is going on and how it all works so:

1. Read the code
2. Review the output from the test executions.

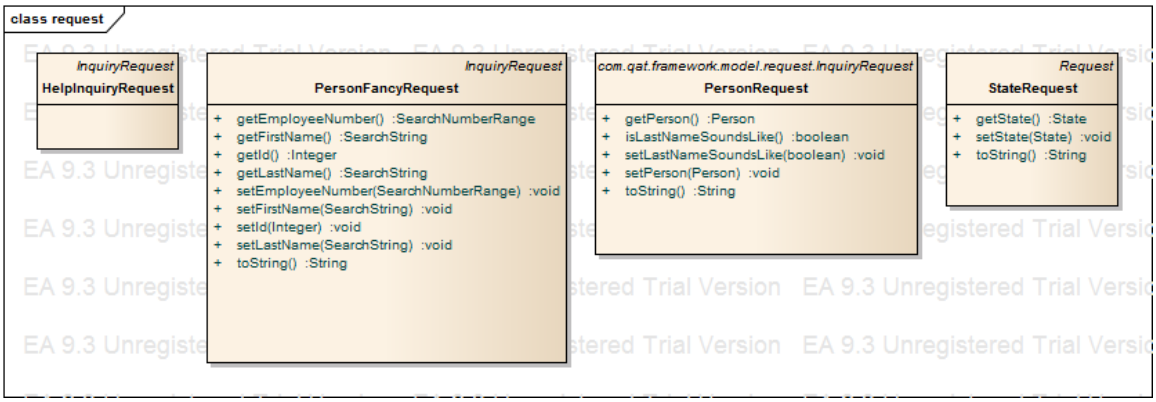
1.6 Class Diagram

Below you'll find the model object class diagram depicting all the model objects used in this sample and their relationships

1.6.1 The model package class diagram



1.6.2 The model.request package class diagram



1.6.3 The model.response package class diagram