DAS Java Developer Guide

Unable to render {include}
The included page could not be found.

How to get involved in development of Java RDB DAS?

This document is the development guideline for RDB DAS Java project.

- General Guide
- Getting Source code
- Setting up your development environment
- Importing DAS code and samples into your Development IDE
- Coding Guidelines
- Testing
- Maven Build Structure
- Reporting Issues and Providing patches

General Guide

Welcome to the Tuscany RDB DAS Java subproject project. We look forward to your participation and try to help you get on board. Feel free to ask your questions on the mailing list.

Here are some general guidelines we use in this project.

- Java RDB DAS sub-project aims to provide enterprise-grade Data Access Service to help SOA applications using SDO have data updates with Database Data Source at backend.
- Java RDB DAS provides feedback to Tuscany SDO and Spec as it evolves.
- The Java RDB DAS provides flexibility in accessing data with explicit CRUD as well as using SDO DataObjects.
- The Java RDB DAS infrastructure is very modularized and is designed to be highly extensible so users can customize it to fit their needs.

Getting Source code

The Java RDB DAS project Subversion repository is located at https://svn.apache.org/repos/asf/incubator/tuscany/java/das .
The repository can also be viewed online at http://svn.apache.org/viewvc/incubator/tuscany/java/das
Anyone can check code out of Subversion. You only need to specify a username and password in order to update the Subversion repository, and only Tuscany committers have the permissions to do so.

Checking out code from Subversion

Use the command as follows (note that it uses http scheme so if you're a committer change it to https):

```
svn checkout http://svn.apache.org/repos/asf/incubator/tuscany/java/das
```

Committing Changes to Subversion

Any Tuscany committer should have a shell account on svn.apache.org. Before you can commit, you'll need to set a Subversion password for yourself. To do that, log in to svn.apache.org and run the command svnpasswd.
Once your password is set, you can use a command like this to commit:

```
svn commit
```

If Subversion can't figure out your username, you can tell it explicitly:

```
svn --username <name> commit
```

Subversion will prompt you for a password, and once you've entered it, it will remember it for you. Note this is the password you configured with svnpasswd not your shell or other password.

Setting up your Development Environment

Prerequisites

Java RDB DAS requires the following:

- JDK 5.0+ (J2SE 1.5.0+)

- Apache Maven (2.0.4+)
- Subversion (1.2+)

Build tree structure

The build tree is designed to facilitate modular development and releases. Maven modules are grouped by how they are released under a hierarchy. Java RDB DAS currently have the below module hierarchy :

```
-java
 |-- das
     |-- distribution         DAS distributions
     |-- rdb                         DAS Core Source
     |-- samples              DAS Web, J2SE Sample Applications
```

The individual modules can be built separately or build with top-down build.

top-down build (recommended approach)

Check out all of the java source code.

```
svn checkout http://svn.apache.org/repos/asf/incubator/tuscany/java
```

Building the RDB DAS source code is simple

```
cd java/das
mvn
```

It should work irrespective of whether you have an empty Maven local repository. This assumes that maven is able to retrieve a SNAPSHOT version of SDO (and of course the rest of software that RDB DAS depends on) as we do not build anything other than RDB DAS here.

There can be occasional problems downloading artifacts from remote Maven repositories so if mvn fails with network related sounding messages sometimes just trying again can fix the problem.
Once you have done a top-down build, and your local maven repository is populated, you can start using the maven off line option to speed up the build process

```
mvn -o
```

Importing RDB DAS into your Development IDE

Using Eclipse

If this is the first time you are using your workspace with maven m2 local repository, you will need to tell your Eclipse workspace the location of the directory, and you can do this with the following command :

```
mvn -Declipse.workspace=[path-to-eclipse-workspace] eclipse:add-maven-rep
```

In order to generate the necessary project files to import the SCA modules to Eclipse, you can use the maven eclipse plugin

```
cd java/das
mvn -Peclipse eclipse:eclipse
```

Now, launch your Eclipse IDE, select File->Import->Existing projects into Workplace, and then select the base DAS directory (e.g java/das) and then press Finish, this should import all DAS core and Samples into your Eclipse Workspace.

Coding Guidelines

There are a few simple guidelines when developing for JAVA DAS:

- Formatting standards are defined by the .checkstyle and .pmd configurations in the source repository. Please be sure to check code is formatted properly before doing a checkin (see below). If you are unfamiliar with Checkstyle or PMD, please see http://checkstyle.sourceforge.net/ and http://pmd.sourceforge.net/ . Consistent formatting makes it easier for others to follow and allows diffs to work properly.
- Always include the Apache License Headers on all files and the following version tag:

```
@version $Rev$ $Date$
```

- Please attempt to accompany code with at least unit tests or verify it by existing tests before submitting a patch or checking in.
- Do not checkin IDE-specific resources such as project files.
- Prior to check-in, perform a clean build and run the complete battery of unit tests for the current module from the command line with Checkstyle enabled, as in:

```
mvn clean
mvn -o -Psourcecheck
```

- Please do not perform a checkin using an IDE as doing so is frequently problematic.
- Include a descriptive log message for checkins, for example "fixed such and such problem".

Naming conventions to increase consistency
Folder Names: Please use all lowercases and dashes in folder names (like in the jar names)

- Maven artifact id = tuscany-<folder name>

Package names: Package names within modules should include the module name so that source code can be located in the source tree easily.

Testing

All commits are expected to be accompanied by unit test and integration tests when appropriate. Unit tests should verify specific behavior relating to a single class or small set of related classes; integration tests verify code paths across subsystems. Testcases should be documented and clearly indicate what they verify. Also, avoid things that may cause side-effects when possible such as access of external resources.
Tuscany uses plain junit test cases to perform unit and integration testing. Web Samples use htmlunit Test cases to verify the expected behavior.
Note that we use surefire maven plugin to run the unit and integration tests, and in most cases, they are configured to match a **/*TestCase.java file name pattern. Because of this, if your test case has a different file name pattern, you might execute it from your IDE of choice, but the maven build won't execute the test.

Maven Build Structure

We use the term Module to refer to the leaf of maven tree.

- das/pom.xml's parent will be pom/parent/pom.xml
- Other poms will use the pom from the parent folder as parent pom
- Group id: org.apache.tuscany.das
- Version of our modules will be specified once in java/das/pom.xml, child poms don't need specify a version as they get it from their parent
- pom names begin Apache Tuscany DAS
- Eclipse projects are generated for all built modules using mvn -Peclipse eclipse:eclipse

Reporting issues and providing patches

Issue Tracking

Tuscany bug reports are handled via a JIRA issues list. Please use this list to report any bugs and track their status.

Reporting an Issue

Please search JIRA to see if the problem has already been reported. If it has not, please create a new JIRA issue. To help developers quickly resolve an issue, include as much information with your report as possible such as your platform, version numbers, error logs, configuration, steps to reproduce the problem, etc. Also, if possible, please include a testcase that demonstrates the problem.
Thanks for working with us to improve Apache Tuscany.

Submitting a Patch

To submit a patch, create an issue in JIRA that describes the problem and attach your patch file. Please include detailed steps to reproduce the problem in the issue description. Providing test cases in the patch will help us verify and apply it quicker. To create a patch, follow the steps below:

- Perform a full build with all tests enabled for the module the fix is for. Specific build procedures vary by sub-project.
- Confirm that the problem is fixed and include testcases where possible
- Generate the patch using svn diff File > patchfile
- Try to give your patch files meaningful names, including the JIRA number
- Add your patch file as an attachment to the associated JIRA issue
  Once you have submitted the patch it will be picked up for review.