How to Contribute Codes to Tajo

How to Contribute Codes to Tajo

This page describes the mechanics of how to contribute codes to Tajo.

Setting up

Here are some things you will need to build and test Tajo. It does take some time to set up a working Tajo development environment, so be prepared to invest some time. Before you actually begin trying to code in it, try getting the project to build and test locally first. This is how you can test your installation.

Software Configuration Management (SCM)

Tajo uses GIT for its SCM system. There are some excellent GUIs for this, and IDEs with tight GIT integration, but as all our examples are from the command line, it is convenient to have the command line tools installed and a basic understanding of them.

Integrated Development Environment (IDE)

You are free to use whatever IDE you prefer, or your favorite command line editor. Note that

- Building and testing is often done on the command line, or at least via the Maven support in the IDEs.
- Set up the IDE to follow the source layout rules of the project.
- If you have commit rights to the repository, disable any added value "reformat" and "strip trailing spaces" features on commits, as it can create extra noise.

Build Tools

To build the code, install

- Apache Maven
- Oracle/Open Java 6

These should also be on your PATH; test by executing mvn and javac respectively.
As the Tajo builds use the external Maven repository to download artifacts, Maven needs to be set up with the proxy settings needed to make external HTTP requests. You will also need to be online for the first builds of Tajo project, so that the dependencies can all be downloaded.

Required Libraries

ProtocolBuffers

The Tajo build requires ProtocolBuffers 2.5.0. You must install ProtocolBuffer according to the package manager of your OS. You can check as follows if ProtocolBuffer is available:

```
$ protoc --version
libprotoc 2.5.0
```

Getting the source code

First of all, you need the Tajo source code. The official location for Tajo is the Apache Git repository. Most development is done on the master branch. You can get the source of the master branch as the following command:

```
git clone https://github.com/apache/tajo.git tajo
```

If you want to develop against a specific branch or release, visit [[https://github.com/apache/tajo]] and find the branch that you are interested in developing against. To copy this branch, run

```
git clone -b [BRANCH NAME] https://github.com/apache/tajo.git tajo
```

Making Changes

Before you start, send a message to the Tajo developer mailing list (dev@tajo.apache.org), or file a bug report in Tajo Jira. Describe your proposed changes and check that they fit in with what others are doing and have planned for the project. Be patient, it may take folks a while to understand your requirements. Modify the source code and add some (very) nice features using your favorite IDE.

But take care about the following points

- All public classes and methods should have informative. (Javadoc comments)
- Do not use @author tags.
- Contributions must pass existing unit tests.
- New unit tests should be provided to demonstrate bugs and fixes. JUnit is our test framework.

Generating a patch

Unit Tests

Please make sure that all unit tests succeed before constructing your patch and that no new javac compiler warnings are introduced by your patch. For building Tajo with Maven, use the following to run all unit tests and build a distribution.

```
mvn clean install -Pdist -Dtar
```

If you use your favorite IDE to run unit tests, you firstly check if the unit tests use MiniTajoYarnCluster or TpchTestBase. If so, you must give the JVM option '-Dtajo.test=TRUE' to your IDE's run configuration. This option forces TaskRunnerLauncherImpl to automatically set CLASSPATH and environment variables used to launch containers on Yarn cluster. If this JVM option is not set, TaskRunnerLaunchImpl depends on the environment variables of the shell.

Creating a patch

Check to see what files you have modified with:

```
git status
```

Add any new files with:

```
git add src/../MyNewClass.java
git add src/../TestMyNewClass.java
```

in order to create a patch, type:

```
git diff --no-prefix > TAJO-7777.patch
```

or if you have worked on your local branch, type:

```
git diff master --no-prefix > TAJO-7777.patch
```

These will report all modifications done on Tajo sources on your local disk and save them into the TAJO-7777.patch file. Read the patch file. Make sure it includes ONLY the modifications required to fix a single issue.

Please do not:

- reformat code unrelated to the bug being fixed: formatting changes should be separate patches/commits.
- comment out code that is now obsolete: just remove it.
- make things public which are not required by end users.

Please do:

- try to adhere to the coding style of files you edit;
- comment code whose function or rationale is not obvious;
- update documentation (e.g., package.html files, this wiki, etc.)

Naming your patch

A suggested patch naming scheme is below:

- <JIRA issue ID #>.<last name>.<yyMMdd>.<n>.patch.txt
- Example for TAJO

```
TAJO-578.Mattmann.032803.patch.txt
```

This:

1. helps identifier the contributor in the actual patch file by their last name (alternative, first letter of first name, and last name or some other alternative may be used).
2. indicates the date on which the patch was submitted in the file name
3. allows for multiple revisions (n parameter) if multiple patches in the same day
4. Adds a .txt at the end, b/c some browsers, and HTTPD servers, don't understand that the .patch file extension is actually a text file and thus downloads instead of displaying the file in JIRA.

Another alternative for patch naming is shown below:

- If the Jira issue number is TAJO-7777, patches for the master branch should be named: TAJO-7777.patch.
- If the branch is 'branch_0_3' and the issue number is TAJO-7777, patches for a non-master branch should be named: TAJO-7777-branch_0_3.patch

Applying a patch

To apply a patch either you generated or found from JIRA, you can issue

```
patch -p0 < cool_patch.patch
```

Contributing your work

- Finally, patches should be attached to an issue report in Jira via the Attach File link on the issue's Jira. Please note that the attachment should be granted license to ASF for inclusion in ASF works (as per the Apache License §5).
- Folks should run 'mvn clean install' before attaching the patch. Tests must all pass.
- If your patch involves performance optimizations, they should be validated by benchmarks that demonstrate an improvement.
- Please be patient. Committers are busy people too. If no one responds to your patch after a few days, please make friendly reminders. Please incorporate other's suggestions into your patch if you think they're reasonable. Finally, remember that even a patch that is not committed is useful to the community.
- Committers: for non-trivial changes, it is best to get another committer to review your patches before commit. Attach your patch on the relevant jira issue in order to share your patch, and then wait for a "+1" from another committer before committing.

Jira Guidelines

Please comment on issues in Jira, making their concerns known. Please also vote for issues that are a high priority for you. Please refrain from editing descriptions and comments if possible, as edits spam the mailing list and clutter Jira's "All" display, which is otherwise very useful. Instead, preview descriptions and comments using the preview button (on the right) before posting them. Keep descriptions brief and save more elaborate proposals for comments, since descriptions are included in Jira's automatically sent messages. If you change your mind, note this in a new comment, rather than editing an older comment. The issue should preserve this history of the discussion.

Stay involved

Contributors should join the Tajo mailing lists. In particular, the commit list (to see changes as they are made) and the dev list (to join discussions of changes and help users).

See Also

- Apache Contributors Tech Guide
- Apache Voting Process