Tests Creation Guide

# Run all tests

1. Open console in tc-api directory
2. Run “**npm start**”
3. Run “**npm test**”

# Run single test

1. Open console in tc-api directory
2. Run “**npm start**”
3. Run “**./node\_modules/.bin/mocha ./test/test.some\_test.file**”

# Name convection and directory structure

1. Required filename for test is “test.[api name].js” (example: test.tops.js).
2. All tests from contest winner, should be put directly under test directory.
3. All reviewers tests should be put under corresponding directory under test directory, like accuracy, failure, security directory.
4. All sql files used for tests presetup and teardown should be put under sqls directory, sqls files for accuracy, failure, security tests, will be in different directory, directly under the accuracy, failure, security directory, like accuracy/sqls, failure/sqls etc.
5. All other test related files should be put under test\_files directory. Test\_files for accuracy, failure, security tests should be put under accuracy/test\_files, failure/test\_files etc.

# Prepare scripts for test data and cleanup

1. **Scripts location**

Put sql scripts under **<base\_dir>/sqls/<api\_name>/** directory.

Submitter tests must be put under **/test/** directory.

Reviewer tests must be put under **/test/accuracy/**, **/test/failure/** or **/test/security/** directory.

*Example:*

Submitter created 2 sql files for Tops API: common\_oltp\_\_clean, common\_oltp\_\_insert\_test\_data.

Above files should be put under /test/sqls/tops/ directory

1. **Naming convention**

Suggested file name has format:

<database\_name>\_\_<type>\_<optional\_description>

*Examples:*

* Script that cleans data in **common\_oltp** database, may have name **common\_oltp\_\_clean.**
* Script that inserts sample data to **common\_oltp** database, may have name **common\_oltp\_\_insert\_test\_data**
* Script that returns some data (SELECT query) from **common\_oltp** database, may have name **common\_oltp\_\_select\_user**

1. **Script size limit**

Maximum script size is 64KB. If file is bigger, then Informix driver will throw exception.

Suggested solution is to split file into parts and add suffix to file name “.partX” where X is part number.

*Examples:*

**topcoder\_dw\_\_insert\_test\_data** is too big file. We split it manually to **topcoder\_dw\_\_insert\_test\_data.part1** and **topcoder\_dw\_\_insert\_test\_data.part2.**

1. **JSON format (Recommended)**

Queries can be defined by .json package.

{

"name" : "<script\_name>",

"db" : "<database\_name>",

"sqlfile" : "<sql\_file\_name>"

}

Sqlfile can be single file or array of files

# Create Tests

1. **“Hooks”**

Mocha provides 4 hooks:

* before() – executed once before all tests
* after()– executed once after all tests
* beforeEach()– executed before each test
* afterEach ()– executed after each test

1. **Test template**

Below is simple sample how to setup and cleanup data.

Before all test we execute scripts **/sqls/xyz/tcs\_dw\_\_clean** and **/sqls/xyz/tcs\_dw\_\_insert\_test\_data**. After all test we execute only **/sqls/xyz/tcs\_dw\_\_clean.**

1. **var** testHelper = require('./helpers/testHelper');
2. **var** SQL\_DIR = \_\_dirname + "/sqls/xyz/";
3. describe('Test xyz API', **function** () {
5. */\*\**
6. *\* Clear database*
7. *\* @param {Function<err>} done the callback*
8. *\*/*
9. **function** clearDb(done) {
10. testHelper.runSqlFile(SQL\_DIR + " tcs\_dw\_\_clean", " tcs\_dw ", cb);
11. }
13. */\*\**
14. *\* This function is run before all tests.*
15. *\* Generate tests data.*
16. *\* @param {Function<err>} done the callback*
17. *\*/*
18. before(**function** (done) {
19. async.waterfall([
20. clearDb,
21. **function** (cb) {
22. testHelper.runSqlFile(SQL\_DIR + "tcs\_dw\_\_insert\_test\_data", "tcs\_dw", cb);
23. }
24. ], done);
25. });
27. */\*\**
28. *\* This function is run after all tests.*
29. *\* Clean up all data.*
30. *\* @param {Function<err>} done the callback*
31. *\*/*
32. after(**function** (done) {
33. clearDb(done);
34. });

37. *//test cases go here*
38. });

For read only API is better to use **before** and **after** hooks instead of **beforeEach** and **afterEach**, because data generation is slow.

1. **Running partial scripts**

Maximum file size is 64KB , because of this we may have to run multiple partial scripts.

*Example:*

We would like to run below scripts:

* topcoder\_dw\_\_insert\_test\_data.part1
* topcoder\_dw\_\_insert\_test\_data.part2

Sample before hook may look like:

1. before(**function** (done) {
2. async.waterfall([
3. clearDb,
4. **function** (cb) {
5. **var** files = testHelper.generatePartPaths(SQL\_DIR + "topcoder\_dw\_\_insert\_test\_data", "", 2);
6. testHelper.runSqlFiles(files, "topcoder\_dw", cb);
7. }
8. ], done);
9. });

testHelper.generatePartPaths method simply add suffix „.partX” to first parameter.

1. **Running json file**
2. before(**function** (done) {
3. async.waterfall([
4. clearDb,
5. **function** (cb) {
6. testHelper.runSqlFromJSON(SQL\_DIR + "common\_oltp\_\_insert\_test\_data.json", cb);
7. }
8. ], done);
9. });