Crascit

Caffeine-powered software

Test Fixtures With CMake/CTest

BY CRAIG SCOTT / ON OCTOBER 18, 2016 / IN SOFTWARE INFRASTRUCTURE

New test properties were added in CMake 3.7.0 to support test *fixtures*. Put simply, a fixture allows setup and cleanup tasks to be associated with a group of tests. On their own, fixtures are quite useful, but they also integrate well with the existing RESOURCE_LOCK and DEPENDS test properties to support some interesting use cases. This article explains the new fixture features and provides examples showing how they complement the existing resource and dependency functionality.

Motivating Example

Consider the scenario where various test cases might interact with a database. Some initial setup of that database may be required before the test cases can access it, such as creating accounts, setting permissions, loading initial data, etc. Furthermore, when the test cases have all completed, they may have left behind a large amount of data which is no longer needed, or the accounts that were added may need to be removed again to prevent misuse or unbounded growth of new accounts.

Prior to CMake 3.7.0, the typical way this would be achieved is to define a new test for the setup task(s) and another for the cleanup task(s). Explicit dependency relationships would then be defined between each individual test and the setup/cleanup tasks, which would look something like the following:

```
add_test(NAME dbSetup   COMMAND ...)
add_test(NAME dbCleanup COMMAND ...)
add_test(NAME dbTest1   COMMAND ...)
add_test(NAME dbTest2   COMMAND ...)

set_tests_properties(dbTest1 dbTest2 PROPERTIES DEPENDS dbSetup)
set_tests_properties(dbCleanup       PROPERTIES DEPENDS dbTest1
dbTest2)
```

When the full set of tests are executed by CTest, this arrangement would ensure that `dbTest1` and `dbTest2` would only run after `dbSetup` completed and `dbCleanup` would only run after both `dbTest1` and `dbTest2` completed. There are, however, two main problems with this:

Problem 1: Tests run even if setup fails

The `DEPENDS` test property only specifies that a particular test cannot run until after the tests it depends on have completed. The relationship says nothing at all about the success or failure of the tests, only the execution order. Thus, in the above example, if the `dbSetup` test fails, the `dbTest1` and `dbTest2` tests will still run. Since database setup has failed in that scenario, one would expect that `dbTest1` and `dbTest2` would not succeed, but the real problem would actually have been the setup, not whatever `dbTest1` and `dbTest2` were meant to be testing. If the results are being recorded from run to run and the historical trend of tests are being analysed, the `dbTest1` and `dbTest2` results will show failures which aren't really true. Ideally, in such situations the `dbTest1` and `dbTest2` tests should be skipped, which would result in a more accurate reflection of the situation in the historical trends.

Another downside to this situation is that the test cases may take non-trivial time to run. Ideally, the tests should fail as early as possible so that the test results can be returned quickly. This encourages developers to run tests often, but if the setup test fails and each of the dependent tests still run and take non-trivial time, the overall turnaround time could be unnecessarily long. In the example scenario, if the database couldn't be set up, each of the dependent tests may take longer than usual if they involve timeouts trying to connect, etc., making this particular point especially relevant.

Problem 2: Test selection might cause setup or cleanup tests to be omitted

Users can instruct CTest to run only a subset of tests. For example, tests can be selected by regular expression:

```
ctest -R Database
```

CTest can also be instructed to re-run only those tests which failed in the previous run:

```
ctest --rerun-failed
```

In these situations, the test execution set can end up containing test cases but not their associated setup or cleanup tests. Thus, when the test subset is run, setup and/or cleanup may not have occurred and tests can fail due to preconditions not being properly set up. This leads to similar downsides as the first problem, with test failures being inaccurately reported in historical test trend analyses and overall test turnaround time potentially being longer than it should be.

One might reasonably expect that if a test is listed in another test's `DEPENDS` property, it would automatically be added to the test execution set, but the `DEPENDS` relationship does not provide such a strong dependence. The `DEPENDS` property is strictly only about ordering and if another test on which a particular test `DEPENDS` is not in the execution set, then the behaviour is as though the dependent relationship didn't exist. Therefore, it does not ensure setup or cleanup tests are included when a subset of tests are executed, regardless of how the test execution set is specified.

The Solution: Fixtures

The concept of test fixtures was added in CMake 3.7.0 to address the above problems. A setup or cleanup task is implemented as just another test case, having all of the usual test case behaviours including support for all test properties and the ability to pass or fail. The fixture relationships between test cases are then specified using the following test properties:

- FIXTURES_REQUIRED
- FIXTURES_SETUP
- FIXTURES_CLEANUP

The first step is to add the fixture requirement to each test for which the setup/cleanup tasks apply. This is done by listing the fixture name in those tests' `FIXTURES_REQUIRED` test property. A fixture name is a non-empty, case-sensitive string and the developer is free to choose any names which make sense for the project.

The second step is to implement the setup task for the fixture as just another ordinary test case, only this time the fixture name is added to the setup test's `FIXTURES_SETUP` test property. Similarly, cleanup tasks are also implemented as an ordinary test case with the fixture name added to that test's `FIXTURES_CLEANUP` test property.

The earlier database example can be modified to use a fixture rather than using `DEPENDS` relationships:

```
 add_test(NAME dbSetup   COMMAND ...)
 add_test(NAME dbCleanup COMMAND ...)
 add_test(NAME dbTest1   COMMAND ...)
 add_test(NAME dbTest2   COMMAND ...)

 set_tests_properties(dbTest1 dbTest2 PROPERTIES
FIXTURES_REQUIRED Db)   set_tests_properties(dbSetup
PROPERTIES FIXTURES_SETUP    Db)   set_tests_properties(dbCleanup
PROPERTIES FIXTURES_CLEANUP  Db)
```

The setup and cleanup tests will execute only once for the entire set of tests, they are not executed once for every test requiring the fixture. If setup and cleanup needs to be performed on a per-test basis, this is better handled by making it part of each test directly rather than using CMake's fixtures functionality. Many existing test frameworks like GoogleTest and Boost Test already have good support for per-test fixtures and CMake's fixtures functionality seeks to complement rather than replace them. For the above example, test execution would be as follows:

1. Run `dbSetup`.
2. If `dbSetup` passed, then run `dbTest1` and `dbTest2`. If `dbSetup` failed, then `dbTest1` and `dbTest2` will be marked as skipped.
3. Regardless of the result of `dbSetup`, `dbTest1` or `dbTest2`, run `dbCleanup`.

If the user instructed CTest to run with a subset of tests which included `dbTest1` and/or `dbTest2`, then `dbSetup` and `dbCleanup` would automatically be added to the execution set if they were not already included. Furthermore, if a setup test fails, the other tests that require the fixture are skipped, resulting in faster overall turnaround of the test run and more accurate reporting of what actually failed. These are the primary benefits of using CMake's fixture functionality.

It is important to note that neither `dbSetup` nor `dbCleanup` have `Db` as a `FIXTURES_REQUIRED` test property. If a setup test listed the same fixture it was setting up as a requirement, it would effectively require itself, which would be a circular dependency. Similarly, if a cleanup test listed the same fixture it was cleaning up as a requirement, it would be waiting for itself to complete before it could execute, which again would be a circular dependency.

It is perfectly fine, however, for setup and cleanup tests to require a different fixture. A test can also require more than one fixture or be a setup and/or cleanup test for multiple fixtures. A fixture can also have just setup tests, just cleanup tests or even no setup or cleanup tests at all (although a fixture with no setup or cleanup tests would be of limited value since it would ultimately have no effect on test execution). The following demonstrates most of these features:

```
 add_test(NAME setupFoo   COMMAND ...)
 add_test(NAME setupBar   COMMAND ...)
 add_test(NAME cleanupFoo COMMAND ...)
 add_test(NAME cleanupBar COMMAND ...)
 add_test(NAME testFoo    COMMAND ...)
 add_test(NAME testBar    COMMAND ...)
 add_test(NAME testBoth   COMMAND ...)
 add_test(NAME oddball    COMMAND ...)

 set_tests_properties(setupFoo   PROPERTIES FIXTURES_REQUIRED
Oddball)  set_tests_properties(testFoo    PROPERTIES
FIXTURES_REQUIRED Foo)  set_tests_properties(testBar
PROPERTIES FIXTURES_REQUIRED Bar)  set_tests_properties(testBoth
PROPERTIES FIXTURES_REQUIRED "Foo;Bar")
 set_tests_properties(oddball    PROPERTIES FIXTURES_SETUP
Oddball)  set_tests_properties(setupFoo   PROPERTIES
FIXTURES_SETUP    Foo)  set_tests_properties(setupBar
PROPERTIES FIXTURES_SETUP    Bar)
 set_tests_properties(cleanupFoo PROPERTIES FIXTURES_CLEANUP
Foo)  set_tests_properties(cleanupBar PROPERTIES
FIXTURES_CLEANUP  Bar)
```

All three of the fixture-related test properties are lists, so when assigning multiple fixture names, they must be specified as a CMake list (i.e. a semicolon-separated string). The `testBoth` test case in the above example demonstrates how to do so correctly.

Also note that because the setup and cleanup tests never list the fixture they apply to in their own `FIXTURES_REQUIRED` test property, they can be run in isolation. For the above example, one could execute just the two cleanup tests as follows:

```
ctest -R cleanup
```

Using Fixtures With RESOURCE_LOCK

Consider the database example once more. CTest has the ability to execute tests in parallel, so it may be necessary to restrict access to the database to only one test at a time. This is what `RESOURCE_LOCK` is intended to address and it would be quite common to see it used in conjunction with test fixtures for exactly this sort of scenario. Some or all of a fixture's tests may also specify a `RESOURCE_LOCK` to serialise their execution. The resource label has no relationship to fixture names and it may be necessary to lock not only the regular test cases, but also the setup and cleanup test cases. For example:

```
 add_test(NAME dbSetup   COMMAND ...)
 add_test(NAME dbCleanup COMMAND ...)
 add_test(NAME dbTest1   COMMAND ...)
 add_test(NAME dbTest2   COMMAND ...)

 set_tests_properties(dbTest1 dbTest2 PROPERTIES
FIXTURES_REQUIRED Db)  set_tests_properties(dbSetup
PROPERTIES FIXTURES_SETUP    Db)  set_tests_properties(dbCleanup
PROPERTIES FIXTURES_CLEANUP  Db)

 set_tests_properties(dbSetup dbCleanup dbTest1 dbTest2
     PROPERTIES RESOURCE_LOCK SerialDb
 )
```

The name of the resource lock could be anything, including the same value as used for a fixture if desired. Note however, that reusing a fixture name for a resource may lead to confusion, so consider carefully whether reusing the same name is a wise choice.

Using Fixtures With DEPENDS

A fixture is not limited to just one setup and/or one cleanup test, they can specify any number of such tests as needed. In such cases, if execution order needs to be specified, the `DEPENDS` test property can be used. The following example highlights how this can be done to make the setup stages more modular and to report any setup failures in a more fine-grained way:

```
add_test(NAME copyConfig     COMMAND ...)
add_test(NAME startDb        COMMAND ...)
add_test(NAME setPermissions COMMAND ...)
add_test(NAME dbTest         COMMAND ...)
add_test(NAME cleanupDb      COMMAND ...)

set_tests_properties(dbTest PROPERTIES FIXTURES_REQUIRED Db)
set_tests_properties(copyConfig startDb setPermissions
    PROPERTIES FIXTURES_SETUP Db
 )
set_tests_properties(cleanupDb PROPERTIES FIXTURES_CLEANUP Db)
```

```
 set_tests_properties(startDb       PROPERTIES DEPENDS
copyConfig)
 set_tests_properties(setPermissions PROPERTIES DEPENDS startDb)
```

The above could alternatively have been implemented as three separate fixtures:

```
 add_test(NAME copyConfig     COMMAND ...)
 add_test(NAME startDb        COMMAND ...)
 add_test(NAME setPermissions COMMAND ...)
 add_test(NAME dbTest         COMMAND ...)
 add_test(NAME cleanupDb      COMMAND ...)

 set_tests_properties(startDb        PROPERTIES FIXTURES_REQUIRED
DbConfigured)
 set_tests_properties(setPermissions PROPERTIES FIXTURES_REQUIRED
DbRunning)
 set_tests_properties(dbTest         PROPERTIES FIXTURES_REQUIRED
DbReady)
 set_tests_properties(copyConfig     PROPERTIES FIXTURES_SETUP
DbConfigured)
 set_tests_properties(startDb        PROPERTIES FIXTURES_SETUP
DbRunning)
 set_tests_properties(setPermissions PROPERTIES FIXTURES_SETUP
DbReady)
```

The use of fixtures would result in cleaner behaviour in this case (e.g. the test to start the database wouldn’t run if the configuration files couldn’t be copied), but if a project has a lot of tests using many fixtures, coming up with unique fixture names could get tedious. In such cases, simple `DEPENDS` relationships may be deemed sufficient.

Another situation where combining `DEPENDS` with fixtures can be useful is to force CTest to run one group of tests before another. Simply make the setup test(s) of one fixture depend on the cleanup test(s) of another:

```
 add_test(NAME setupFoo   COMMAND ...)
 add_test(NAME setupBar   COMMAND ...)
 add_test(NAME cleanupFoo COMMAND ...)
 add_test(NAME cleanupBar COMMAND ...)
 add_test(NAME testFoo    COMMAND ...)
 add_test(NAME testBar    COMMAND ...)

 set_tests_properties(testFoo    PROPERTIES FIXTURES_REQUIRED
Foo)
 set_tests_properties(testBar    PROPERTIES FIXTURES_REQUIRED
Bar)
 set_tests_properties(setupFoo   PROPERTIES FIXTURES_SETUP
Foo)
 set_tests_properties(setupBar   PROPERTIES FIXTURES_SETUP
Bar)
 set_tests_properties(cleanupFoo PROPERTIES FIXTURES_CLEANUP
Foo)
 set_tests_properties(cleanupBar PROPERTIES FIXTURES_CLEANUP
Bar)
 set_tests_properties(setupBar   PROPERTIES DEPENDS
cleanupFoo)
```

In this case, the `Bar` tests will only run after the `Foo` tests, including `Foo` ‘s cleanup. Importantly, even if any `Foo` test fails, the `Bar` tests will still run. The same effect could not be achieved using fixture test properties alone.

The key point to note is that fixtures and the `DEPENDS` property are not mutually exclusive, they can be combined as is most convenient for the project. In many cases, fixtures may be a more robust replacement for `DEPENDS` relationships, while for others the most effective arrangement may be a combination of both.

General Comments

Fixtures, resources and dependencies between tests are all related concepts. When used appropriately, they allow excellent control over what tests should run and when. Fixtures come with few drawbacks, about the only potential negative being related to naming. Fixture names are global to the whole build and the dependencies between tests are evaluated across the entire set of tests known to the build. As such, all the usual considerations for naming uniqueness apply as for any feature that uses global names (resources have similar naming considerations). For projects with a large number of fixtures, consider adopting a predictable naming convention to reduce the potential for accidental name clashes.

Finally, as the author of CMake's fixtures functionality, I'd be interested to hear about other novel uses fixtures might enable. Feel free to leave a comment if you come up with new ways to exploit the feature. As always, bug reports should be submitted to the CMake developers mailing list or Kitware's gitlab instance.

Share this:

5

Like this:

★ Like

Be the first to like this.

CMAKE CTEST

PREVIOUS

Avoiding Copies And Moves With auto

NEXT

Generated Sources In CMake Builds

Leave a Reply

Enter your comment here...