Table of Contents

CORAL and COOL functional tests

This twiki page describes the functional tests of CORAL and COOL executed against all supported backends (Oracle, MySQL, SQLite, Frontier and CoralServer) and some of their internal implementation details. More condensed practical recipes for running the tests to validate software builds can be found in the PersistencyReleaseValidation page. Additional information about the servers aganst which these tests are executed can be found in the PersistencyTestServers page.

1 Functional test suites

There are two sets of test suites for CORAL/COOL:

- The **full test suites** include all functional tests developed over the years for CORAL and COOL. These tests take a long time (~15 minutes for CORAL, ~2 hours for COOL). They are meant to be manually executed by the development team to validate important code changes and new releases (see PersistencyReleaseValidation). They are launched using the `qmtestRun.sh` script, created by cmake from the `qmtestRun.sh.in` templates defined for CORAL and COOL.
- The **quick test suites for the nightlies** are subsets of the full test suites. They have been specifically prepared for the builds and tests that are automatically executed every night, as described below. They contain few selected tests to fit within approximately 5 minutes of execution time, while still providing basic coverage of all supported backends for both CORAL and COOL. This was agreed with EP-SFT in January 2015. They are launched using the `run_nightly_tests_cmake.sh` script, created by cmake from the `run_nightly_tests_cmake.sh.in` templates defined for CORAL and COOL.

The test suites are implemented as qmtest suites:

- The full test suites are CoralTest/qmtest/ALL.qms and CoolTest/qmtest/ALL.qms. They are the default qmtest suites executed by the `qmtestRun.sh` script.
- The quick test suites are CoralTest/qmtest/QUICK.qms and CoolTest/qmtest/QUICK.qms. They are the default qmtest suites executed by the `run_nightly_tests_cmake.sh` script.
    - On some platforms that do not support AFS (such as Mac), the quick test suites are further reduced in the nightlies to exclude Oracle and MySQL, bypassing the problem of looking up the `authentication.xml` file from AFS: in this case, the test suites are CoralTest/qmtest/QUICK_NO_AFS.qms and CoolTest/qmtest/QUICK_NO_AFS.qms. The choice of qmtest suite to execute takes place inside `run_nightly_tests_cmake.sh`.

For both CORAL and COOL, the `ALL.qms` and `QUICK.qms` qmtest suites are files that are manually edited and maintained, pointing to other existing individual tests (`.qmt` files) or collections of them (`.qms` files or directories). The `.qmt` files are written using XML syntax, where some of the XML attributes are generic to qmtest while others have been introduced for CORAL and COOL. For COOL, all `.qmt` files need to be edited manually and committed to SVN. For CORAL, another XML file testlist.xml contains the master list of all tests that should be present and their properties: individual `.qmt` files are generated by editing this file and then executing `./createconfig.py -r` in the same directory, rather than editing them directly (but any additions/removals/modifications to inidividual `.qmt` files should still be committed manually to SVN).

Note that functional tests may be executed also individually (as described below), not only within qmtest suites.

- For the tests executed within qmtest suites, some environment variables are set within the qmtest drivers, CoralTest/qmtest/LCG_QMTestExtensions.py for CORAL and CoolTest/qmtest/COOLTests.py for COOL.
- For the tests executed individually, outside qmtest suites, these environment variables are not set automatically and it is necessary to configure them manually if necessary/appropriate.
- Note in particular that the `CORAL_AUTH_PATH` and `CORAL_DBLOOKUP_PATH` described in the section below should normally be set manually for all manual tests, whether using qmtest driver scripts or not. These two variables are only set automatically by the qmtest drivers if the Linux `$USER` is `sftnight`, which happens during the automatic nightly tests.

2 Connection strings and table names for the tests (hardcoded values and how to override them)

Most CORAL and COOL tests involve creating and modifying tables on database servers. As many tests can be executed simultaneously by different developers and also for different slots and platforms within the nightlies, the configuration of the CORAL and COOL tests includes mechanisms to avoid clashes between these parallel threads of execution. For instance, this is meant to prevent an SLC6 test in the "dev3" nightly slot from deleting a table that has just been created by a CentOS7 test in the "dev4" slot.

Within the same database server, tables are uniquely identified by a name within a namespace. As the same Oracle server and the same MySQL server are used for all manual and automatic tests of CORAL and COOL (see PersistencyTestServers), the separation of different tests is achieved by properly configuring table names and their namespaces:

- The namespace for the tables used in a test corresponds to the the name of the Oracle "schema" or MySQL "database" where those tables are created. This is configured by choosing the CORAL **connection string** for writing tables in that test. As described in CoolConnectionStrings, a CORAL connection string (which is also included in a COOL connection string) uniquely specifies an Oracle or MySQL server name and a schema or database name on that server. The configuration of connection strings in CORAL and COOL tests is described in the section below.
- Each CORAL or COOL test may use several tables. However, all tables in the same test are configured to have names that start with a well defined **table prefix**. This table prefix, or "hash", is built to have distinct values on different nightly slots and platforms, as described in the section below.

2.1 Connection strings for the tests

The configuration of connection strings for individual tests relies on hardcoded defaults using database aliases, which can be overridden using runtime environment variables. The detailed strategy for configuring connection strings is different in CORAL and COOL. In both cases, the functional tests include both C++ and Python tests, which are configured using similar mechanisms, but in different parts of the code.

2.1.1 CORAL tests

Connection strings are defined in CoralCppUnitDBTest.h for CORAL C++ tests and in PyCoralTest.py for CORAL Python tests. These files are included by all relevant C++ and Python unit and integration tests. Unit tests hardcode the choice of a specific backend, while integration tests require the choice of database backend as their first command-line argument. The following mapping is defined for an integration test `<test>`:

- `<test> oracle[:oracle]` writes to alias `CORAL-Oracle-<qmuser>/admin` and reads from alias `CORAL-Oracle-<qmuser>/reader`
- `<test> oracle:frontier` writes to alias `CORAL-Oracle-<qmuser>/admin` and reads from alias `CORAL-Frontier-<qmuser>/reader`
- `<test> oracle:coral` writes to alias `CORAL-Oracle-<qmuser>/admin` and reads from alias `CORAL-CoralServer-Oracle-<qmuser>/reader`
- `<test> mysql[:mysql]` writes to alias `CORAL-MySQL-<qmuser>/admin` and reads from alias `CORAL-MySQL-<qmuser>/reader`
- `<test> mysql:coral` writes to alias `CORAL-MySQL-<qmuser>/admin` and reads from alias `CORAL-CoralServer-MySQL-<qmuser>/reader`
- `<test> sqlite[:sqlite]` writes and reads on `sqlite_file:/tmp/<user>_<hash>.db`

With the exception of SQLite tests (where <user> is the value of the $USER environment variable and <hash> is the table prefix described below), all other backends use CORAL aliases that depend on a variable <qmuser>. Up until the CORAL 3.1.5 release in LCG85 included, <qmuser> was actually hardcoded to sftnight: this was only changed after LCG85 and its value is now taken from the CORAL_QMTEST_USER environment variable (see CORALCOOL-2888). This is meant to provide more flexibility in the use of different database accounts for tests by different developers, as described in PersistencyReleaseValidation.

The CORAL test suite also includes one special test that does not use any of the aliases above. The "network glitch" test test_PyCoral_NetworkGlitch.py hardcodes a special connection string oracle://lcg_coral_nightly_proxy/<nguser>, where the Oracle network service name lcg_coral_nightly_proxy is actually a local ssh tunnel to an existing Oracle server. The schema name <nguser> for the test used to be hardcoded to lcg_coral_nightly until CORAL 3.1.5 in LCG85 included, but it is now possible to override it from the value of the CORAL_NETWORKGLITCHTEST_USER environment variable (see CORALCOOL-2888). This is also meant to provide more flexibility in the use of different database accounts for tests by different developers, as described in PersistencyReleaseValidation.

2.1.2 COOL tests

COOL tests are generally configured to use as COOL connection string the value of the environment variable COOLTESTDB. For COOL C++ tests, this is centrally defined in CoolDBUnitTest.h. For COOL Python tests, this is implemented independently, but in the same way, in each test, allowing also the option to use a command-line argument as connection string (see for instance test_IDatabaseSvc.py). Some special tests that involve copying data from a source to a destination database use two connection strings taken from the values of the two environment variables COOLTESTDB_SOURCE and COOLTESTDB_TARGET (see for instance test_Replication.py). Another special case is the "regression" test, where data is written using one connection string and possibly read back through another (e.g. written via Oracle and read back via Frontier or CoralServer): this is normally handled by using a CORAL alias in COOLTESTDB and resolving it for read-write or read-only replicas, but for faster manual tests the option to set the read-back connection string via environment variable COOLTESTDB_R is also provided in testReferenceDb_driver.py.

The value of COOLTESTDB (as well as COOLTESTDB_SOURCE or COOLTESTDB_TARGET when relevant) must be set manually when executing a single test, but it is automatically configured in COOLTests.py for tests executed within a qmtest suite:

- Oracle tests use COOL connection string COOL-Oracle-<qmuser>/<hash>
- MySQL tests use COOL connection string COOL-MySQL-<qmuser>/<hash>
- Oracle tests with readback from Frontier use COOL connection strings COOL-Frontier-<qmuser>/<hash> and COOL-FrontierCache-<qmuser>/<hash> (for accessing Frontier servers or Squid caches, respectively)
- Oracle tests with readback from CoralServer use COOL connection string COOL-CoralServer-Oracle-<qmuser>/<hash>
- MySQL tests with readback from CoralServer use COOL connection string COOL-CoralServer-MySQL-<qmuser>/<hash>
- SQLite tests use COOL connection string sqlite://;schema=/tmp/sqliteTest-<hash>.db;dname=<hash>, where <hash> is the table prefix described below

In the connection strings above, for all backends <hash> is the table prefix described below: for COOL, this is used to indicate the name of a COOL "database", i.e. a self-consistent set of tables, whose names all start with the database name as their prefix. With the exception of SQLite tests, the COOL connection strings for all other backends include CORAL aliases that depend on a variable <qmuser>. Up until the CORAL 3.1.3 release in LCG84 included, <qmuser> was always defined from the value of the environment variable $USER, i.e. the name of the Linux user running the tests: this was changed in CORAL 3.1.4 and its value is now taken from the COOL_QMTEST_USER environment variable (see CORALCOOL-2888). This is meant to provide

more flexibility in the use of different database accounts for tests by different developers, as described in PersistencyReleaseValidation.

2.2 Table names for the tests

For all tests, table names are built by adding a test-specific suffix to a table name prefix that generally depends from the platform and nightly slot where tests are being executed. The motivation for using different table names for different platforms and slots, within the same table namespace (Oracle schema or MySQL database) is to avoid interference between the different simultaneous nightly builds that are executed every night. For instance, this prevents a dev3 SLC6 test from deleting a table that has just been created by a dev4 CC7 test. The mapping of any given platform and nightly slot is implemented in different ways in CORAL and COOL:

- For CORAL, a unique table name prefix is built based on the values of the environment variables SLOTNAME and CORAL_BINARY_TAG_HASH. This is implemented in CoralCppUnitTest.h for C++ tests and in PyCoralTest.py for Python tests. The environment variable SLOTNAME (see CORALCOOL-2933 and SPI-918) is set by the package lcgjenkins, which is used together with lcgcmake by EP-SFT in their nightly build infrastructure: its value is "dev4" in the "dev4" nightly slot, for instance. The environment variable CORAL_BINARY_TAG_HASH is uniquely hardcoded depending on the BINARY_TAG platform identifier in the CMakeLists.txt file of the CoralTest package; this file must be modified every time a new platform is added (see for instance CORALCOOL-2895).
    - Note that a similar strategy is used in PyCoralTest.py to define a unique port number based on the values of SLOTNAME and CORAL_BINARY_TAG_HASH for the CORAL "network glitch" test. This is meant to avoid interferences between different instances of this test running simultaneously on the same client node for different nightly slots and/or platforms. a local port number is used by this test to create an ssh tunnel to an Oracle database, which can be stopped temporarely to simulate a network glitch.
- For COOL, a unique table name prefix is defined in COOLTests.py, where a hash is built from the directory name where the tests are executed (which contains the slot name in lcgjenkins/lcgcmake) and from the BINARY_TAG platform identifier. For simplicity the table name prefix will be called a "hash" in the following also for CORAL, even if it is built as a hash only for COOL.

It should be noted that the mechanism above is useful not only for automatically executed nightly tests, but also for tests executed manually by the development team. Both the CORAL and COOL implementations avoid clashes between tests executed by the same developer on different platforms. They also avoid clashes between tests executed in a nightly slot and those executed on the same platform by a developer, as the SLOTNAME environment variabe should remain unset in the latter case. It is nevertheless recommended that different table namespaces (i.e. different Oracle schemas and MySQL databases) should be used by the automatic nightly tests and by each developer: this is described in the following subsection about connection strings for the tests.

how propagate unique hadh to tests?

add also port number!

3 Setting the runtime environment for the tests

CORAL and COOL builds are performed with cmake. This is described in the PersistencyCMake page, which also contains a section about the runtime environment setup for these builds. To use the libraries and executables from a specific local (e.g. trunk) or release (e.g. LCG84) build of CORAL or COOL, environment variables such as PATH and LD_LIBRARY_PATH must be set appropriately. In summary, this can be done using one of these two commands:

- run `cc-run <command> [<arguments>]` to execute a command with the appropriate CORAL/COOL environment
- run `cc-sh` to enter a temporary `bash` shell with the appropriate CORAL/COOL environment

The `cc-run` and `cc-sh` scripts are deployed in the install directory of each build and, in principle, can be directly used from there.

- For local builds, this is always true. For instance, run `/build/CORAL/trunk/x86_64-slc6-gcc49-dbg/cc-sh` to enter a temporary shell and execute a CORAL test from there.
- For release builds, which are prepared on build nodes and then copied/relocated to AFS or cvmfs, some issues affect all release up to LCG84 included (see CORALCOOL-2873 and SPI-855), but everything should be fixed as of the LCG85swan1 release. For instance, run `/cvmfs/sft.cern.ch/lcg/releases/LCG_85swan1/COOL/3_1_4/x86_64-slc6-gcc49-dbg/cc-sh` to enter a temporary shell and execute a COOL test from there.

As previously mentioned, CORAL/COOL tests can be executed either individually or within qmtest test suites. In the latter case, the `qmtestRun.sh` and `run_nightly_tests_cmake.sh` scripts, which can also be found in the install directory of each build, should be used. It should be noted that these scripts already take care of setting up the runtime environment by internally calling the appropriate `cc-run` script, hence it is not necessary to explicitly call `cc-run` to call them.

In summary, several options are available. For instance, after changing directory to the install directory of the relevant build with `cd /home/avalassi/COOL/trunk/x86_64-slc6-gcc49-dbg`, one can execute individual or subsets of COOL tests as follows:

- run `./cc-run test_CoolKernel_Record` to execute the Record test standalone from the original shell
- run `./cc-sh` and then `test_CoolKernel_Record` to execute the Record test standalone from a COOL shell
- run `./qmtestRun.sh coolkernel.record` to execute the Record test within the general qmtest wrapper (run `./cc-run qmtest -D CoolTest/qmtest/ ls -R` to list all tests)
- run `./qmtestRun.sh coolkernel.record coolkernel.recordadapter` to execute the Record and RecordAdapter tests within the general qmtest wrapper
- run `./qmtestRun.sh` to execute the full test suite within the general qmtest wrapper (this defaults to executing the `ALL` test suite)
- run `./qmtestRun.sh ALL` to execute the full test suite within the general qmtest wrapper
- run `./qmtestRun.sh QUICK` to execute the quick (nightly) test suite within the general qmtest wrapper
- run `./run_nightly_tests_cmake.sh` to execute the quick (nightly) test suite within the nightly qmtest wrapper

export CORAL_TESTSUITE_SLEEPFOR01466=1

export CORAL_SQLITE_SYNCHRONOUS_OFF=1

4 The authentication.xml, dblookup.xml and tnsnames.ora files (CORAL_AUTH_PATH, CORAL_DBLOOKUP_PATH and TNS_ADMIN)

The functional tests of CORAL and COOL are executed against all supported backends (Oracle, MySQL, SQLite, Frontier and CoralServer). More information about the servers against which these tests are executed can be found in the PersistencyTestServers page. The configuration of connection strings for individual tests relies on hardcoded defaults using database aliases, which can be overridden using runtime environment variables as described in another section. In any case, three files are extremely important to all tests and must be located when the tests start:

- The `authentication.xml` file contains usernames and passwords for the Oracle and MySQL connection strings used by the tests. It is looked up via the `CORAL_AUTH_PATH` environment variable.
- The `dblookup.xml` file contains the mapping between the aliases used by the tests and the connection strings for the corresponding replicas. It is looked up via the `CORAL_DBLOOKUP_PATH` environment variable.
- The `tnsnames.ora` file contains the mapping between the Oracle instance names used by the connection strings and the corresponding host names and ports. It is looked up via the `TNS_ADMIN` environment variable.

In most cases, `CORAL_AUTH_PATH` and `CORAL_DBLOOKUP_PATH` are set to the same value, indicating a directory that contains both `authentication.xml` and `dblookup.xml`. This directory must be access-restricted as `authentication.xml` contains database credentials.

Whenever available, AFS is used to simplify the distribution and access control for all of these files. By default, nightly tests on Linux look up these three files from the following AFS directories:

- `authentication.xml` and `dblookup.xml` from `/afs/cern.ch/sw/lcg/app/pool/db/` (access-protected and readable by `sftnight` and by CORAL/COOL team members with their AFS token)
- `tnsnames.ora` from `/afs/cern.ch/sw/lcg/app/releases/CORAL/internal/oracle/admin` (where it is available as a symbolic link to the master copy on `/afs/cern.ch/project/oracle/admin`, together with a custom `sqlnet.ora` file for CORAL/COOL tests which is also needed to control Oracle client-server communication)

On platforms where AFS is not available (or where the AFS filesystem is available but AFS tokens are not available to the users running the jobs), various options are possible. In the nightlies (e.g. for Mac), Oracle and MySQL tests are skipped and only SQLite tests are executed. For manual tests, developers can work around the issue by copying the relevant files to a directory of their choice and setting the `CORAL_AUTH_PATH`, `CORAL_DBLOOKUP_PATH` and/or `TNS_ADMIN` environment variables accordingly. By default, if `CORAL_AUTH_PATH` and `CORAL_DBLOOKUP_PATH` are not set, their values often default to `$HOME/private` as this is where they typically reside. Using custom `authentication.xml` and `dblookup.xml` files (typically in `$HOME/private`) is also needed for developers who want to use their own Oracle or MySQL test accounts, for instance.

For the tests executed within qmtest test suites, the default configuration of these three environment variables is defined within the qmtest drivers, CoralTest/qmtest/LCG_QMTestExtensions.py for CORAL and CoolTest/qmtest/COOLTests.py for COOL. It is these two files that contain special settings for user `sftnight` (depending on the platform) and also set `CORAL_AUTH_PATH` and `CORAL_DBLOOKUP_PATH` to `$HOME/private` if not yet defined.

5 Running the tests

6 Nightly builds and tests (jenkins, cdash, lcgcmake)

The CORAL and COOL software is built and tested automatically every night, thanks to the infrastructure provided by the SPI team in EP-SFT.

The nightlies test the software "in three dimensions":

- **Platform** (e.g. x86_64-slc6-gcc49-dbg). The tests are built on all production platforms, as well as a few development platforms. Linux SLC6/CC7 on Intel processors with 64-bit builds using gcc49 is currently the only supported platform in June 2016, but support for other platforms is being added, including MacOSX 10.11 El Capitan (e.g. CORALCOOL-2884), gcc6.1 (CORALCOOL-2895), clang 3.8 (CORALCOOL-2901), ARM64 (CORALCOOL-2857). Windows builds are no longer supported. Some of the nightlies are executed using virtual machines instead of the native O/S they are testing.
- **Day of the week** (e.g. Mon, Tue... Sun). All projects are compiled every night. Usually the building and testing starts around midnight (using tarballs created from the appropriate software tags as collected at 23.15, i.e. 45 minutes in advance) and results are provided later in the morning. The produced binaries and results are kept for one week (but a summary of the results of earlier builds and tests can, in principle, be recovered from AFS backup even later - if they were installed on AFS in the first place, which is not the case for all platforms).
- **Slot** (e.g. dev2, dev3, dev4, experimental...). Projects are built in different configurations, i.e. a set of versions of AA packages that are supposed to work together. For instance, one configuration slot may test a preview of the "HEAD" of all projects, while other configuration slots may test branches with patches over the current production release. The "dev" slots are generally more stable than the "experimental" slot, where new development platforms and new disruptive extenal package versions tend to appear first. Presently (June 2016), only results from the "dev" slots are copied to AFS, while those from the "experimental" slot stay on the build nodes and are published on the web. The top of the PersistencyReleaseNotes page normally contains a brief description of the differences between the currently active dev slots.

The tests are executed using jenkins and can be controlled on the "LCG Externals" tab of the phsft-jenkins.cern.ch web interface. This makes it possible to start new unscheduled tests (please check with the SPI team in EP-SFT before!), by clicking on "Build with Parameters" in the appropriate slot, e.g. lcg_experimental. If you made important changes to the code between 23.15 and midnight, and you want these to be included in the next nightly build, this inteface also allows you to recreate the tarballs from your software tags, by clicking on "Build with Parameters" in the lcg-prepare-builds section.

The results of the nightlies are available on the "LCGSoft" tab of cdash.cern.ch. For those platforms and slots that get installed on AFS, the binaries and results are also available under `/afs/cern.ch/sw/lcg/app/nightlies`, for instance on `/afs/cern.ch/sw/lcg/app/nightlies/dev2/Fri/CORAL` for the dev2 build on CORAL on the most recent Friday.

The nightly builds and tests of CORAL and COOL are started by jenkins within a job that performs a full build of the LCG stack using lcgcmake. The relevant configuration for CORAL and COOL builds and tests is all contained within the externals/CMakeLists.txt file. In particular, this is the file that sets up the nightly tests to use the `run_nightly_tests_cmake.sh` script.

The nightlies are all executed as user `sftnight`. On the main Linux platforms, a valid AFS token for `sftnight` is available for the duration of the build job.

-- AndreaValassi - 2016-06-24

This topic: Persistency > PersistencyTests
Topic revision: r15 - 2016-08-10 - unknown