Table of Contents

Coverity scans for the Persistency packages

The Persistency packages are scanned regularly using the Coverity code analyzer.

This is done thanks to the infrastructure provided by the EP-SFT group. This consists of a Coverity Connect server on coverity.cern.ch and a build node on buildcoverity.cern.ch.

1. Coverity server (set up the database to host the results of the Coverity scans)

The results of the Coverity scans for the Persistency packages can be viewed and analyzed by logging on the Coverity Connect instance on coverity.cern.ch.

Prepare projects, streams, triage stores and component maps for Coverity scans

To use the Coverity Connect server, refer to the Coverity Usage and Administration Guide (authentication required). Before using Coverity, you or your admin must have configured the following sets of server-side entities for you:

- One or more *users* and passwords
- One or more *projects*, owned by the relevant users
  - ♦ It is best to define separate projects for software packages whose defects should be analyzed independently, because the user-level view defines a project as the top-level context that can be changed with a drop-down menu
- One *stream* within each project, owned by the relevant users
  - ♦ When defects are committed to the database from a software build, they are sent to a single stream, which automatically defines the project they belong to
  - ♦ It is possible to have several streams under the same project, then define special menus in the user dashboard to see within a project only the defects coming from a well-defined stream
  - ♦ In principle it may also be possible to share the same stream across projects by creating in project B a "link" for a stream belonging to project A, but the benefits of this are not clear
- Optionally, one or more *triage stores* to store defect and triage history, owned by the relevant users
  - ♦ One stream uses one and only one triage store, but the same triage store may be used by different streams (e.g. the Default triage store is used by many streams)
  - ♦ If a dedicated triage store is not created, the Default triage store is used
- Optionally, one or more *component maps* to categorize defects by component within each project
  - ♦ One stream uses one and only one component map, but the same componenet map may be used by different streams (e.g. the Default component map is used by many streams)
  - ♦ If a dedicated component map is not created, the Default component map used
  - ♦ Unlike projects, streams and triage stores, component maps are all owned by the admin user and their ownership may not be transferred
    - ◊ Managing component maps requires the global (not per-project) role "project admin" (see "managing custom roles")

The following setup is being used for CORAL and COOL as of January 2016:

- Projects: two separate projects **CORAL** and **COOL** have been created
- Streams: two separate streams **CORAL-Stream-trunk** and **COOL-Stream-trunk** have been created, one in each of the CORAL and COOL projects
- Triage stores: a single triage store **CORALCOOL-TriageStore** has been created and has been associated to the two streams above
- Component maps: a single component map **CORALCOOL-CompMap** has been created and has been associated to the two streams above

    - Within this component map, several components such as "system", "gcc", "Boost", "Qt", "lcgexternal", "PyCool", "CORAL_SERVER" have been defined with associated file name rules
- Users: the two users valassi and avalassi can be equivalently used, they are project owners, stream owners and triage store owners for all entities described above

2. Coverity build node (build, analyze and commit Coverity scans)

The scans are prepared on a specific SLC6 node buildcoverity.cern.ch, where Coverity is installed. The new infrastructure allows software builds with c++11, which could not be used on the previous machine.

Log in using your AFS user name (e.g. *avalassi*):

```
ssh buildcoverity.cern.ch [-l username]
```

Prepared directories for the Coverity scans

The Persistency scans are prepared in `/builda/Persistency` and its subdirectories.

Extended attribute ACLs (see `getfacl` and `setfacl`) can be used to make sure that all relevant users can read and write the same files there.

Several subdirectories for the scans (two top-level directories for CORAL and COOL, with a single subdirectory for the trunk in each project) have been created from scratch in January 2016: please use the existing tags/platforms (update them from SVN if necessary) and do not add any other tags/platforms.

```
/builda/Persistency/CORAL/trunk
/builda/Persistency/COOL/trunk
```

In each `/builda/Persistency/<project>/trunk` directory, the results are created in a `cov-out` subdirectory (which must be deleted before each new scan), at the same level as `src` and the build and installation directories. For instance:

```
/builda/Persistency/CORAL/trunk/build.x86_64-slc6-gcc49-dbg
/builda/Persistency/CORAL/trunk/cov-out
/builda/Persistency/CORAL/trunk/src
/builda/Persistency/CORAL/trunk/x86_64-slc6-gcc49-dbg
```

Execute the Coverity scans: build, analyze, commit

In the following, we will describe how to prepare the scans for CORAL and COOL (the latter using the former). To prepare the Coverity scans, you must do the following:

- Check out the appropriate version of the code.

```
cd /builda/Persistency/CORAL
\rm -rf trunk
svn co svn+ssh://svn.cern.ch/reps/lcgcoral/coral/trunk

cd /builda/Persistency/COOL
\rm -rf trunk
svn co svn+ssh://svn.cern.ch/reps/lcgcool/cool/trunk
```

- Check that the local COOL build is set up to use the local CORAL build (this should contain `export CMAKE_PREFIX_PATH=/builda/Persistency/CORAL/trunk/$BINARY_TAG` if `$HOSTNAME` is buildcoverity.cern.ch).

```
cd /builda/Persistency/COOL/trunk
cat overrideSetupCMake.sh
```

- Set up the build environment (conventionally, we use the x86_64-slc6-gcc49-dbg platform). Then, remove old build fragments and old Coverity scan results and finally build the code through the Coverity wrapper. This is done via the build.sh script attached to this page.

```
cd /builda/Persistency/CORAL/trunk
../../scripts/build.sh
cd /builda/Persistency/COOL/trunk
../../scripts/build.sh
```

- If the project build through the Coverity wrapper is successful, the following output should be produced. For reference, the build on 2016 January 27 gave 478 (100%) for CORAL trunk after 5 minutes and 196 (100%) for COOL30x after 3 minutes.

```
XXX C/C++ compilation units (100%) are ready for analysis
The cov-build utility completed successfully.
```

- After building the code, let Coverity analyze the results. For reference, the analysis of the 2016 January 27 builds took 2 minutes and found 31 defects for CORAL and it took 1 minutes and found 43 defects for COOL. This is done via the analyze.sh script attached to this page.

```
cd /builda/Persistency/CORAL/trunk
../../scripts/analyze.sh
cd /builda/Persistency/COOL/trunk
../../scripts/analyze.sh
```

- After analyzing the code through Coverity, commit the results to the database so that it gets published on coverity.cern.ch. This is done via the commit.sh script attached to this page. Note that this step is quite slow: for reference, for the analysis of the 2016 January 27 builds it took 4 and 3 minutes to commit (in parallel!) the CORAL and COOL defects, respectively.

```
cd /builda/Persistency/CORAL/trunk
../../scripts/commit.sh
cd /builda/Persistency/COOL/trunk
../../scripts/commit.sh
```

3. Coverity server (analyze and triage the results of the Coverity scans)

After committing the results of Coverity scans, go back to the Coverity server. You may then analyze and triage all defects (and iterate after fixing them if needed).

Current status

- As of July 2012, ALL issues found by Coverity in CORAL and COOL had been fixed (an/or at least properly documented and filed for later resolution). External issues dur for instance to Boost, libstdc++ or ROOT had been dismissed (and in the case of ROOT, e.g. in the PyCool build, reported as bug ROOT-4380). The POOL issues will not be fixed as the relevant code has been moved to an ATLAS specific package.
    - The relevant Savannah tickets are task #20073 for COOL, task #20075 for CORAL, bug #95365 for CORAL_SERVER and task #20074 for POOL.

 - ♦ As described in these tickets, some issues can only be fixed with API changes in CORAL and COOL. These API changes were going to be released in November 2013 in CORAL 2.4.0 and COOL 2.9.0, but previously they had already been committed to the code protected with `#ifdef CORAL240CO` and `#ifdef COOL290CO` guards. In order to validate these changes, the `VersionInfo.h` files in the two projects had been locally modified to enable these and some other API extensions through the Coverity builds in July 2012. The COOL build had also been configured (using valassi's private requirements) to use the local CORAL build with the relevant API extensions too.

- In June 2015, Coverity scans were resumed, on a new PH-SFT Coverity server supporting c++11 for the first time. The gcc49 compiler, with c++11 enabled, was used for the first time (CMTCONFIG=x86_64-slc6-gcc49-dbg). Builds of the CORAL_3xx and COOL_3xx branches, using c++11 also in the public API, were executed for the first time. Builds were again (and for the last time) executed using CMT. ALL issues found by Coverity were fixed (and/or triaged and dismissed).
    - ♦ See CORALCOOL-2768 for more detaills.

- In January 2016, Coverity scans have been again resumed, after the move from CMT to cmake and the cleanup of the SVN repository (moving active development from the 3xx branch to trunk). The gcc49 compiler, with c++11 enabled, is being used again, this time using cmake for the first time (BINARY_TAG=x86_64-slc6-gcc49-dbg). Builds are now executed on CORAL and COOL trunk directly. Note en passant that ccache is disabled in these cmake builds, as missing defect files would be interpreted as due to fixed defects or source files removed from the repository. ALL issues found by Coverity have been triaged and dismissed (all real issues had already been fixed in previous campaigns).

-- AndreaValassi - 2016-01-27

This topic: Persistency > PersistencyCoverity
Topic revision: r26 - 2016-03-09 - AndreaValassi