Unittesting and Smoketesting for the LHCb Online Event Building Pipeline

Astrid Ch. Zieritz (Summer Student), Robert L. Gulyas (Supervisor), Flavio Pisani (Supervisor)

CERN, Summer 2024

Abstract

In this work, implementation of unittests for the Multiple Fragment Packet (MFP) and the Multiple Event Packet (MEP) are described as well as the implementation of a smoketest for the event building (EB) script, both implemented in the Large Hadron Collider beauty (LHCb) Online Framework's Gitlab CI/CD.

1. Introduction

The Large Hadron Collider beauty (LHCb) is a forward spectrometer, designed to study beauty and charm quarks, and it is one of the experiments at the Large Hadron Collider (LHC). This paper will mainly focus on the data formats and formatting from the LHCb experiment data acquisition (DAQ) pipeline, or more specifically the data right before and right after the event building (EB) process as well as the EB itself, as these are the parts for which the tests have been implemented.

2. Background

Figure 2 shows a simplified overview of what the data packets look like before and after the EB process. The experiment data production starts when two proton beams are bent to intercept each other, causing protons to collide. The beam consists of multiple proton bunches with a set spacing between them, and each bunch crossing is denoted as one *Event* (see Figure 1).

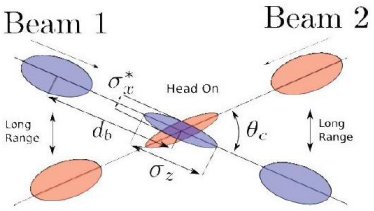

Figure 1: One bunch crossing/Event between bunches from beam 1 and beam 2 [1]

The data recorded by one data source for one event is called a *Fragment*. Multiple fragments from one data source for consecutive events are stored together to form a *Multiple Fragment Packet* (MFP).

The EB process is a process that takes multiple MFPs containing the same events together to form a *Multiple Event Packet* (MEP). This is done to enable the user to extract data for one specific event for all data sources. More on the EB process can be found in [2] and naming definitions can be found in [3].

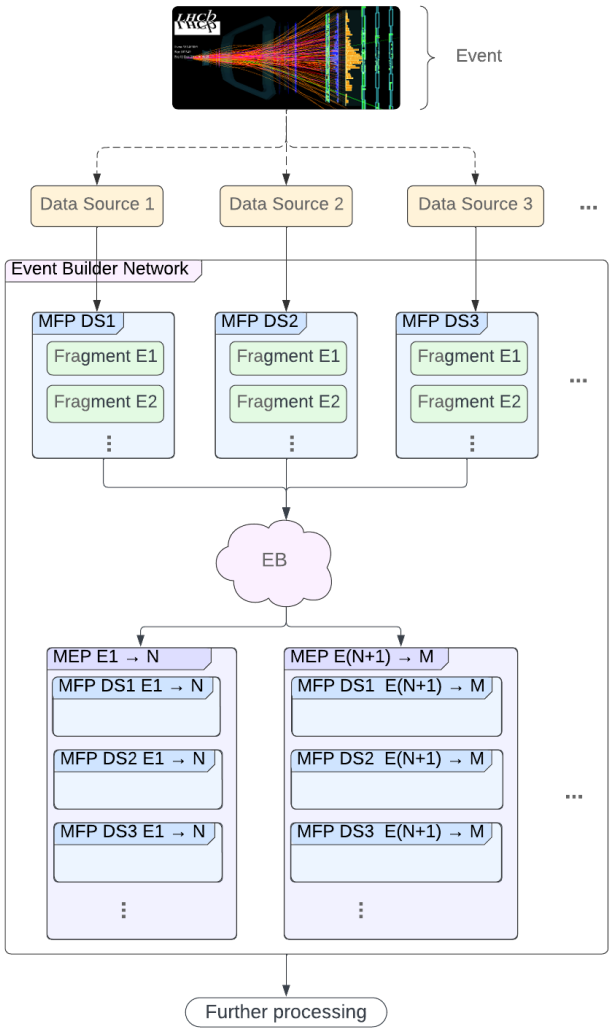

Figure 2: Packet structure in the event building network before and after the event building. Event image from [4]

3. Test Implementation

3.1. Unittests

The goal for the unittests is to ensure that future updates of the EB software will not break the already defined MFP and MEP interfaces and packet definitions as described in [3].

For the MFP test, a test.mfp file has been provided for the test. This file is both read to a buffer and loaded to the currently defined MFP class. The MFP test first goes through every value defined for the MFP to compare if the MFP class values are the same as the extracted values from the buffer. After this, the class functions are tested to ensure that they are returning correct values. Below follows an excerpt from some of the test code.

```
// Check magic number
int offset_magic = 0;
ASSERT_TRUE(is_within_bounds(filesize, offset_magic,
    sizeof(uint16_t)));
uint16_t buffer_magic = *reinterpret_cast<uint16_t*>(&
    mfp_buffer[offset_magic]);
ASSERT_EQ(mfp_struct->header.magic, buffer_magic);

// Function test
bool is_magic_val = (buffer_magic == EB::MFP_magic);
ASSERT_EQ(mfp_struct->is_header_valid(), is_magic_val);
```

For the MEP test, a test.mep file has been provided as well as a folder containing all the MFPs that make up the given test.mep. (A script for extracting the MFPs from a MEP has also been implemented.)
The MEP test first uses the given MFP files to construct an MEP, and then compares it to the original test.mep. The test.mep file can be changed with newly generated files for an updated test. After this, the MEP test checks the MEP class function returns to ensure that they are correct. Below follows an excerpt from some of the MEP test code.

```
// NMFPS
uint16_t gen_nmfps = count_files_in_folder(
    mfp_files_path);
// Copy into generated MEP
std::memcpy(gen_mep.data() + curr_offset, &gen_nmfps,
    size_nmfps);
/*...*/
ASSERT_EQ(std::memcmp(gen_mep.data(), mep_buffer.data(),
     mep_buffer.size()), 0)

// Function test
bool magic_valid = (gen_magic == EB::MEP_magic);
ASSERT_EQ(mep_struct->is_magic_valid(), magic_valid);
```

The unittests can be found in the LHCb online repository on Gitlab under *Online/EventBuilding/tests/* where the test files are named *MFP_check_test.cpp* and *MEP_check_test.cpp*. The script for extracting the MFPs from a given MEP is under *Online/EventBuilding/main/MFPs_from_MEP.cpp*.

3.2. Smoketest

The smoketest implemented is a test that runs the *run_eb.py* script which runs the EB to check if it "starts smoking", or if any errors occur or if it is working as expected.
The smoketest spawns three threads in addition to the main loop. The first thread runs the *run_eb.py* script and continuously logs its output to file. The second thread continuously reads the log file to look for error messages. The third thread monitors some of the script counters to check if the script is working as expected. The main loop checks the thread status as well as keeping track of a timer for the test execution. The smoketest will fail on errors found in the log file, execution errors or if the main loop times out. The smoke test will only succeed if all counters monitored reach a given threshold value before the test times out. A sequence diagram for the smoketest can be found in Figure 3 and the smoketest is implemented in the LHCb online git repository under *Online/EventBuilding/tests/smoketest_run_eb.py*.

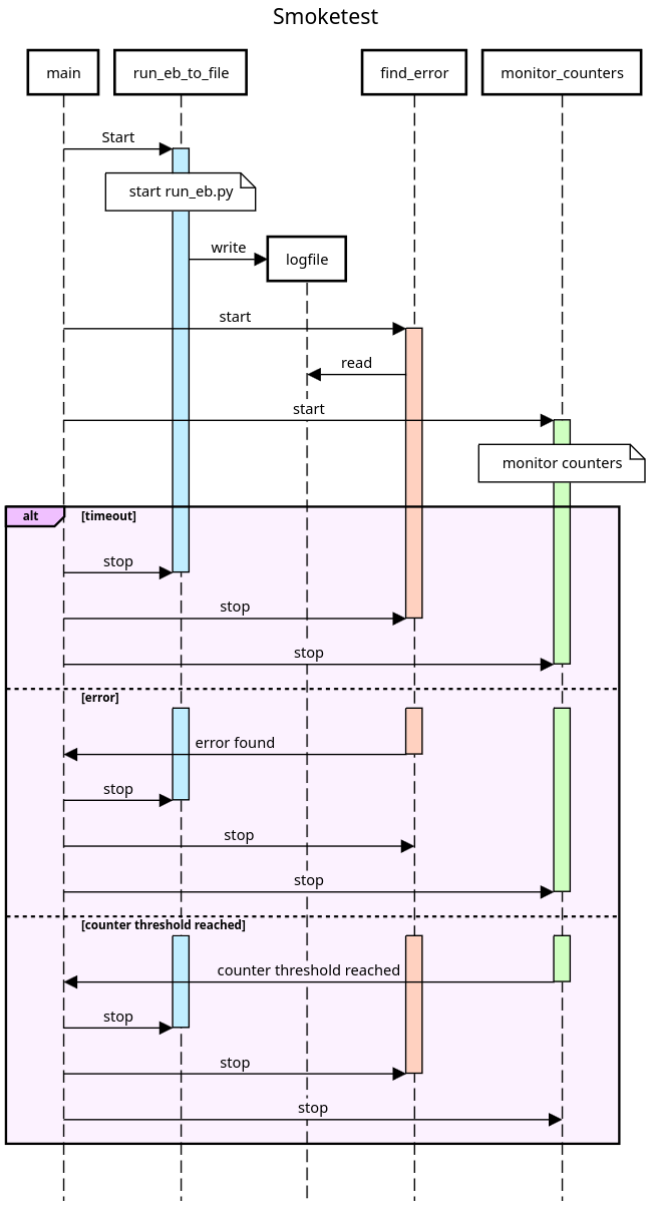

Figure 3: Sequence diagram of the smoketest

3.3. Pipeline Integration

The unittets and the smoketest are scrips that can be triggered manually for debugging purpose, but the CI/CD also runs them every time new code is pushed. The unittests have been implemented to run in the *build_application* stage of the pipeline, right after the build process finishes. The smoketest has been implemented in the *test* stage of the pipeline. An overview of the different pipeline stages can be found in Figure 4.

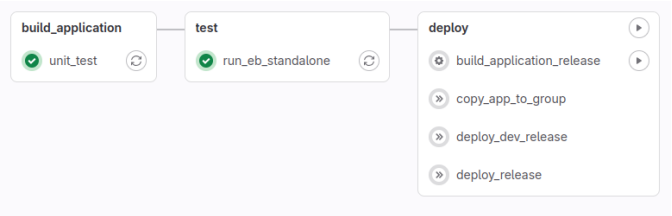

Figure 4: Stages and jobs for the LHCb online git repository pipeline

In addition to integrating the tests into the Gitlab pipeline, one new virtual machine (VM) has been set up to run Gitlab jobs specifically for LHCb online called *ebgitlabrunner02*. On this machine, two Gitlab runners are configured, one online docker runner and one online shell runner. Also, one online docker runner has been configured for the preexisting *ebgitlabrunner01*, which already has an online shell runner from previously.

4. Results

The unittests have been implemented and are running without error in the *build_application* stage of the pipeline using one of the docker runners, as seen in Figure 5.

```
$ ctest --rerun-failed --output-on-failure
Test project /swdev/eventbuilder_cicd/gitlab-builds/azieritz_eb_ci-3c
    Start 1: MFPFileTest.LoadMFPToStruct
1/2 Test #1: MFPFileTest.LoadMFPToStruct ......   Passed    0.31 sec
    Start 2: MEPGenTest.GenMEPFromMFPs
2/2 Test #2: MEPGenTest.GenMEPFromMFPs ........   Passed    0.20 sec
100% tests passed, 0 tests failed out of 2
Total Test time (real) =   0.54 sec
```

Figure 5: Unittest running in pipeline

While writing the unittests, it was found out that the MEP packets lost up to 3 bytes of data from the last MFP depending on the alignment of the MFP payload data. This was fixed and a new test.mep file was added, enabling the tests to run successfully.

The smoketest is also running successfully in the pipeline on the shell runners, as seen in Figure 6.

```
Monitor_counters::Host list received: ['tdeb02', 'tdeb03', 'tdeb06', 'tdeb07']
Monitor_counters::Monitoring 32 counters...
Monitor_counters::Service reached threshold: EB_TDEB02_RU_0/RU/barrier_count
Monitor_counters::Service reached threshold: EB_TDEB02_RU_0/RU/pre_barrier_count
Monitor_counters::Service reached threshold: EB_TDEB03_RU_0/RU/barrier_count
Monitor_counters::Service reached threshold: EB_TDEB03_RU_0/RU/pre_barrier_count
Monitor_counters::Service reached threshold: EB_TDEB06_RU_0/RU/barrier_count
Monitor_counters::Service reached threshold: EB_TDEB06_RU_0/RU/pre_barrier_count
Monitor_counters::Service reached threshold: EB_TDEB07_RU_0/RU/barrier_count
Monitor_counters::Service reached threshold: EB_TDEB07_RU_0/RU/pre_barrier_count
Monitor_counters::Service reached threshold: EB_TDEB02_RU_1/RU/barrier_count
Monitor_counters::Service reached threshold: EB_TDEB03_RU_1/RU/barrier_count
Monitor_counters::Service reached threshold: EB_TDEB06_RU_1/RU/barrier_count
Monitor_counters::Service reached threshold: EB_TDEB07_RU_1/RU/barrier_count
Monitor_counters::Service reached threshold: EB_TDEB02_RU_1/RU/pre_barrier_count
Monitor_counters::Service reached threshold: EB_TDEB03_RU_1/RU/pre_barrier_count
Monitor_counters::Service reached threshold: EB_TDEB06_RU_1/RU/pre_barrier_count
Monitor_counters::Service reached threshold: EB_TDEB07_RU_1/RU/pre_barrier_count
Monitor_counters::Service reached threshold: EB_TDEB02_BU_0/BU/barrier_count
Monitor_counters::Service reached threshold: EB_TDEB02_BU_0/BU/pre_barrier_count
Monitor_counters::Service reached threshold: EB_TDEB03_BU_0/BU/barrier_count
Monitor_counters::Service reached threshold: EB_TDEB03_BU_0/BU/pre_barrier_count
Monitor_counters::Service reached threshold: EB_TDEB06_BU_0/BU/barrier_count
Monitor_counters::Service reached threshold: EB_TDEB06_BU_0/BU/pre_barrier_count
Monitor_counters::Service reached threshold: EB_TDEB07_BU_0/BU/barrier_count
Monitor_counters::Service reached threshold: EB_TDEB07_BU_0/BU/pre_barrier_count
Monitor_counters::Service reached threshold: EB_TDEB02_BU_1/BU/barrier_count
Monitor_counters::Service reached threshold: EB_TDEB03_BU_1/BU/barrier_count
Monitor_counters::Service reached threshold: EB_TDEB06_BU_1/BU/barrier_count
Monitor_counters::Service reached threshold: EB_TDEB07_BU_1/BU/barrier_count
Monitor_counters::Service reached threshold: EB_TDEB02_BU_1/BU/pre_barrier_count
Monitor_counters::Service reached threshold: EB_TDEB06_BU_1/BU/pre_barrier_count
Monitor_counters::Service reached threshold: EB_TDEB03_BU_1/BU/pre_barrier_count
Monitor_counters::Service reached threshold: EB_TDEB07_BU_1/BU/pre_barrier_count
Main::Event detected: Counter threshold reached
Main::All processes terminated
Main::TEST OK: Counter threshold reached
```

Figure 6: Smoketest running in pipeline

5. Conclusion

The tests were successfully implemented and are running in the pipeline, but currently only in the *azieritz_eb_ci* branch of the repository.

Implementation of the tests resulted in detection of data loss during the construction of the MEP packets, enabling this bug to be fixed, and thus showing the importance of software testing.

5.1. Future Work

Future work may include implementing more tests to the pipeline to increase the code coverage of the tests, and also to merge the tests into the production branch.

References

[1] *Taking a closer look at lhc - lhc p collisions.* https://www.lhc-closer.es/taking_a_closer_look_at_lhc/0.lhc_p_collisions, (read 30.07.2024).

[2] F. Pisani, et al.: *Design and commissioning of the first 32 tbit/s event-builder.* 70(6), Jun 2023.

[3] *Raw-data transport format and odin fragment format, lhcb technical note.* (version 5, reference EDMS 2100937).

[4] *End of 2016 data taking period.* https://lhcb-outreach.web.cern.ch/2016/12/05/end-of-2016-data-taking-period/, (read 30.07.2024).