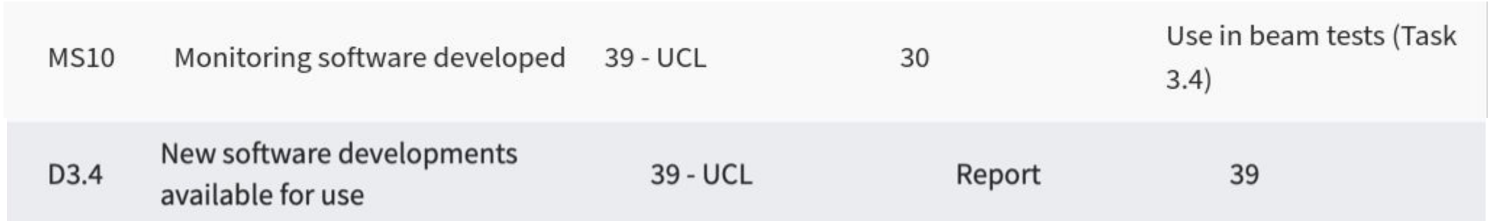
Online Monitoring at Test Beams with EUDAQ2 and Corryvreckan

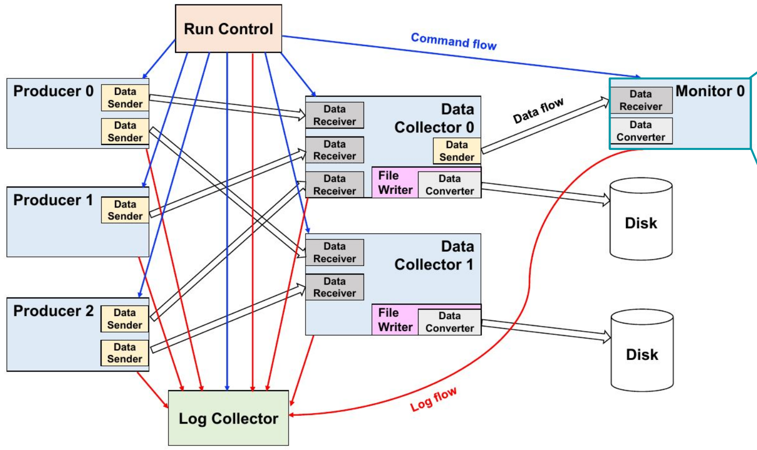
Andreas Loeschcke Centeno (University of Sussex)
(a.loeschcke-centeno@sussex.ac.uk)

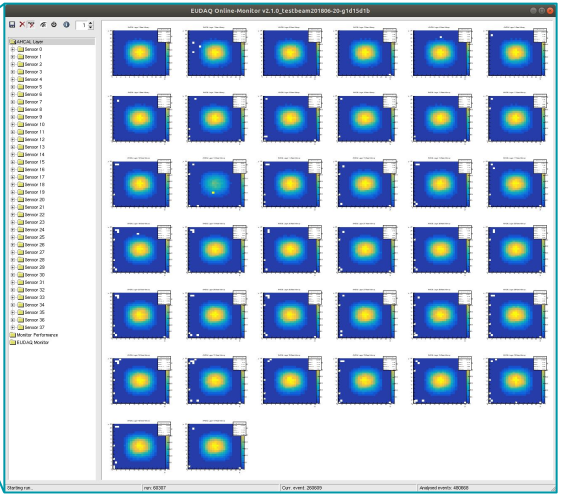
Online Monitoring and EUDAQ2

- AIDAinnova Task: “Development of versatile online monitoring for EUDAQ2”



MS10	Monitoring software developed	39 - UCL	30	Use in beam tests (Task 3.4)
D3.4	New software developments available for use	39 - UCL	Report	39





Starting run.. | run: 60307 | Curr. event: 260609 | Analysed events: 480668

Online Monitoring and EUDAQ2

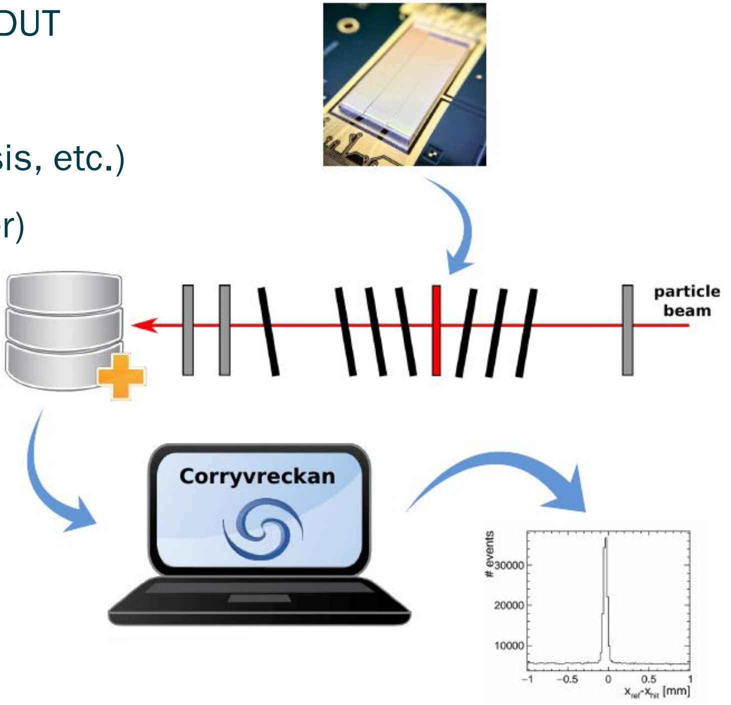
Limits:

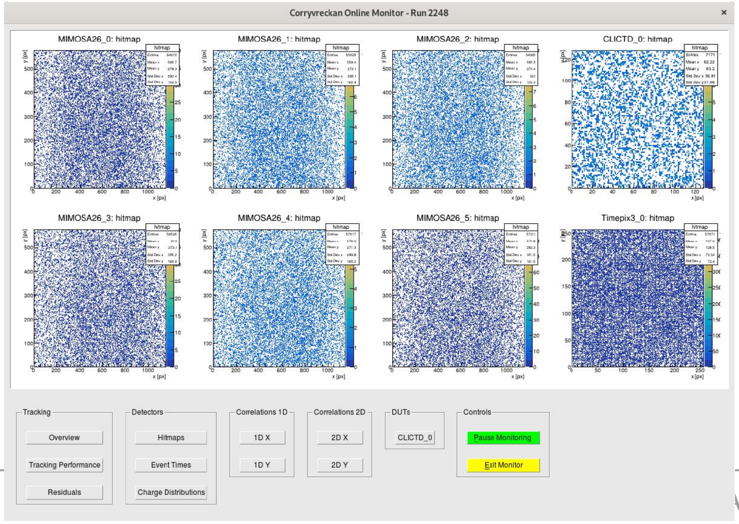
- Single input file
- No flexible event building/processing
- No information about detector geometry
- No direct possibility to do tracking from beam telescope data
  - track angles and track-based alignment
  - event association with DUT & DUT analysis
- BUT: commonly used test beam software provides all functionality we need:

  corryvreckan

Corryvreckan [reference]

- Reconstruction and analysis tool for test beam data
  - conceived to work with data from beam telescopes + DUT
- Modular structure allows for flexibility
  - modules for specific tasks (clustering, tracking, analysis, etc.)
  - EventLoaderEUDAQ2 (needs eudaq::StdEventConverter)
- Comes with an OnlineMonitor module

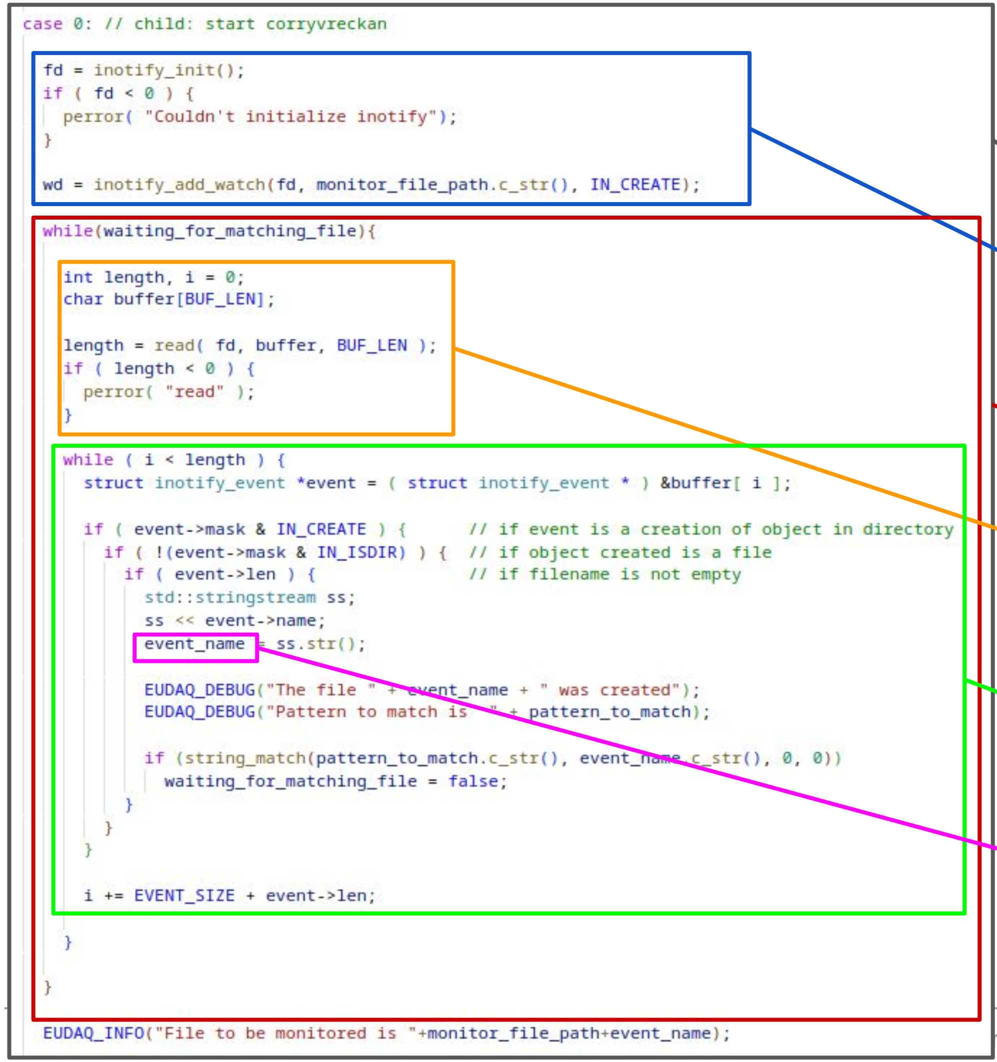




CorryMonitor

How to incorporate corry into EUDAQ2?

- Want to be able to control centrally from euRun instance
    - Call corry from a eudaq::Monitor class
- General minor improvements to user experience
- Corry needs to know name of data file
    - User input only before start of run
    - DC file naming pattern: `EUDAQ_FW_PATTERN = run$3R_$12D$X`
        - R -> Run Number
        - D -> Date and Time
        - X -> File Extension
    - Need to automatically find correct files to monitor



```
case 0: // child: start corryvreckan

fd = inotify_init();
if ( fd < 0 ) {
  perror( "Couldn't initialize inotify");
}

wd = inotify_add_watch(fd, monitor_file_path.c_str(), IN_CREATE);

while(waiting_for_matching_file){

  int length, i = 0;
  char buffer[BUF_LEN];

  length = read( fd, buffer, BUF_LEN );
  if ( length < 0 ) {
    perror( "read" );
  }

  while ( i < length ) {
    struct inotify_event *event = ( struct inotify_event * ) &buffer[ i ];

    if ( event->mask & IN_CREATE ) {      // if event is a creation of object in directory
      if ( !(event->mask & IN_ISDIR) ) {  // if object created is a file
        if ( event->len ) {               // if filename is not empty
          std::stringstream ss;
          ss << event->name;
          event_name = ss.str();

          EUDAQ_DEBUG("The file " + event_name + " was created");
          EUDAQ_DEBUG("Pattern to match is " + pattern_to_match);

          if (string_match(pattern_to_match.c_str(), event_name.c_str(), 0, 0))
            waiting_for_matching_file = false;
        }
      }
    }

    i += EVENT_SIZE + event->len;

  }

}
EUDAQ_INFO("File to be monitored is "+monitor_file_path+event_name);
```

```
#include <sys/wait.h>
#include <sys/stat.h>
#include <sys/inotify.h>
#include <regex>
#include <filesystem>
```

Fork to start corry

Setting up inotify

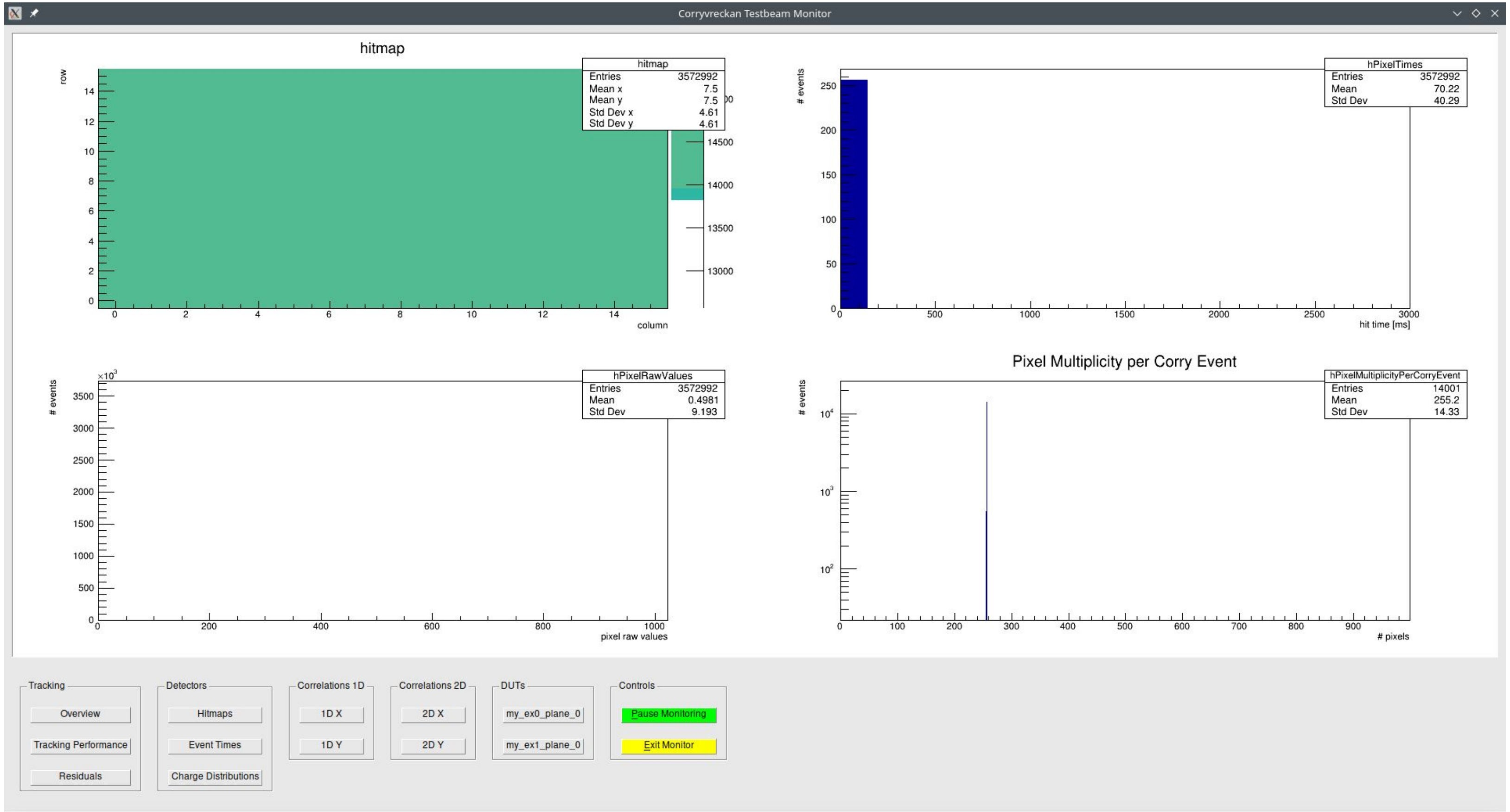
While waiting for the correct file to be created

Read the "event" (change in directory)

Check of any of the events read in this instance are the creation of the desired file

Variable storing file name

CorryMonitor in Action



```
hitmap
Entries     3572992
Mean x          7.5
Mean y          7.5
Std Dev x      4.61
Std Dev y      4.61
```

```
hPixelTimes
Entries     3572992
Mean          70.22
Std Dev       40.29
```

```
hPixelRawValues
Entries     3572992
Mean         0.4981
Std Dev       9.193
```

```
hPixelMultiplicityPerCorryEvent
Entries       14001
Mean          255.2
Std Dev       14.33
```

Changes To corryvreckan: EventLoaderEUDAQ2.cc

- In current state:

  corry finishes when reaching end of file

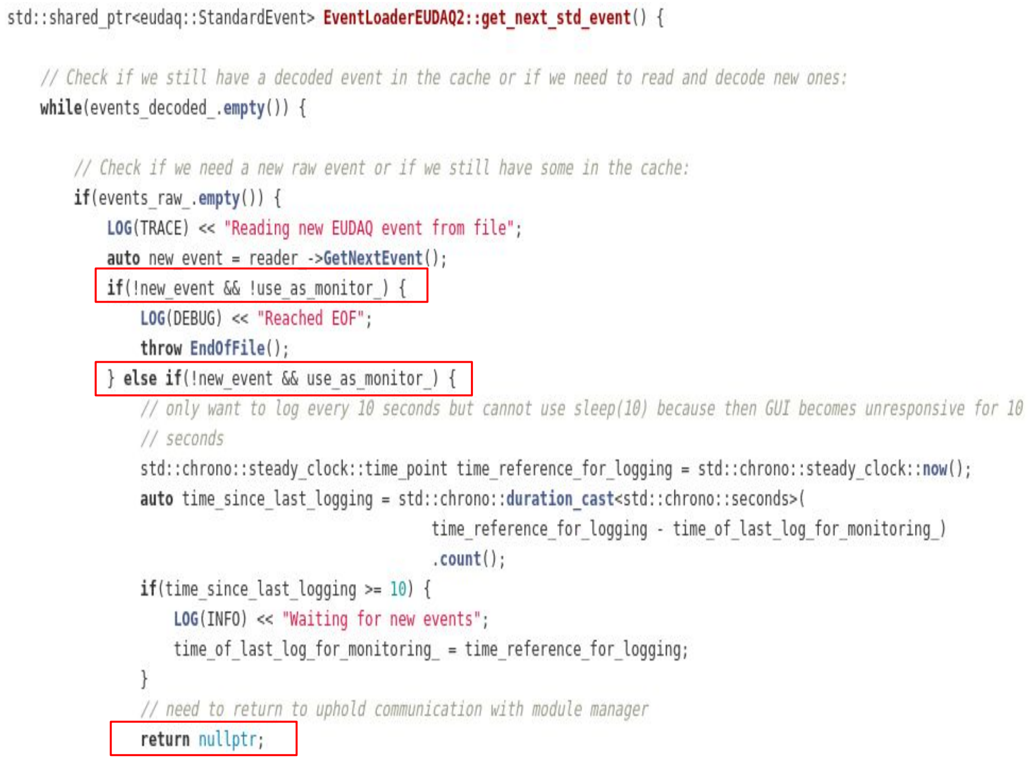
- When monitoring, we might need to wait for

  new data:

  - `use_as_monitor` flag for

    EventLoaderEUDAQ2 to prevent

    closing when no new events are found



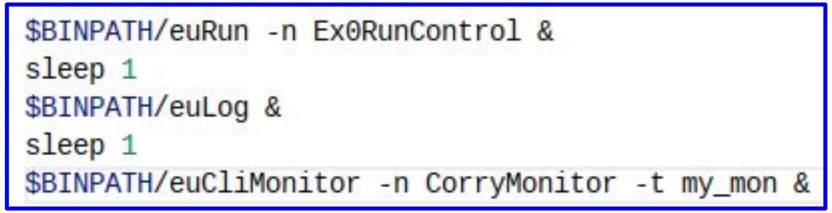
```
std::shared_ptr<eudaq::StandardEvent> EventLoaderEUDAQ2::get_next_std_event() {

    // Check if we still have a decoded event in the cache or if we need to read and decode new ones:
    while(events_decoded_.empty()) {

        // Check if we need a new raw event or if we still have some in the cache:
        if(events_raw_.empty()) {
            LOG(TRACE) << "Reading new EUDAQ event from file";
            auto new_event = reader_->GetNextEvent();
            if(!new_event && !use_as_monitor_) {
                LOG(DEBUG) << "Reached EOF";
                throw EndOfFile();
            } else if(!new_event && use_as_monitor_) {
                // only want to log every 10 seconds but cannot use sleep(10) because then GUI becomes unresponsive for 10
                // seconds
                std::chrono::steady_clock::time_point time_reference_for_logging = std::chrono::steady_clock::now();
                auto time_since_last_logging = std::chrono::duration_cast<std::chrono::seconds>(
                                                   time_reference_for_logging - time_of_last_log_for_monitoring_)
                                                   .count();
                if(time_since_last_logging >= 10) {
                    LOG(INFO) << "Waiting for new events";
                    time_of_last_log_for_monitoring_ = time_reference_for_logging;
                }
                // need to return to uphold communication with module manager
                return nullptr;
```

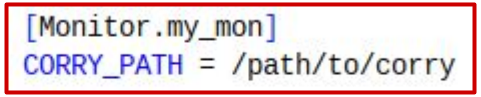
How to Monitor

1. Startup-script



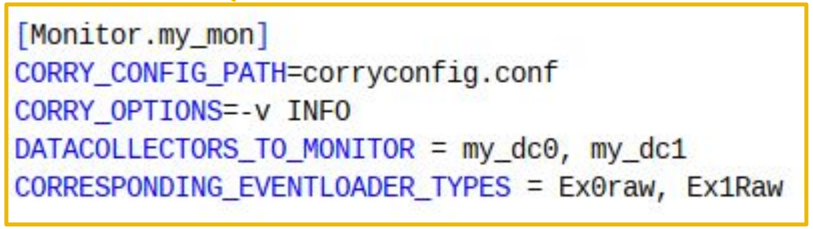
```
$BINPATH/euRun -n Ex0RunControl &
sleep 1
$BINPATH/euLog &
sleep 1
$BINPATH/euCliMonitor -n CorryMonitor -t my_mon &
```

2. EUDAQ2 .ini file



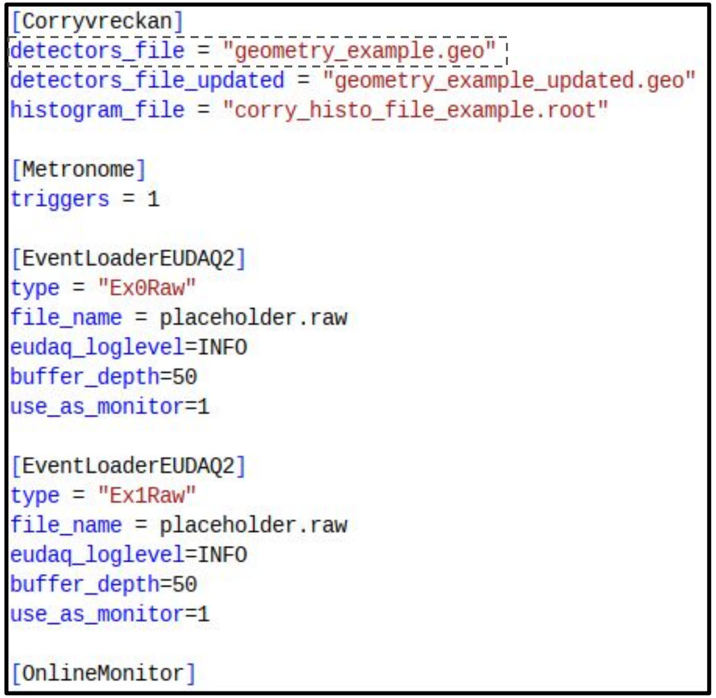
```
[Monitor.my_mon]
CORRY_PATH = /path/to/corry
```

3. EUDAQ2 .conf file



```
[Monitor.my_mon]
CORRY_CONFIG_PATH=corryconfig.conf
CORRY_OPTIONS=-v INFO
DATACOLLECTORS_TO_MONITOR = my_dc0, my_dc1
CORRESPONDING_EVENTLOADER_TYPES = Ex0raw, Ex1Raw
```

corryvreckan .conf file



```
[Corryvreckan]
detectors_file = "geometry_example.geo"
detectors_file_updated = "geometry_example_updated.geo"
histogram_file = "corry_histo_file_example.root"

[Metronome]
triggers = 1

[EventLoaderEUDAQ2]
type = "Ex0Raw"
file_name = placeholder.raw
eudaq_loglevel=INFO
buffer_depth=50
use_as_monitor=1

[EventLoaderEUDAQ2]
type = "Ex1Raw"
file_name = placeholder.raw
eudaq_loglevel=INFO
buffer_depth=50
use_as_monitor=1

[OnlineMonitor]
```

Conclusion

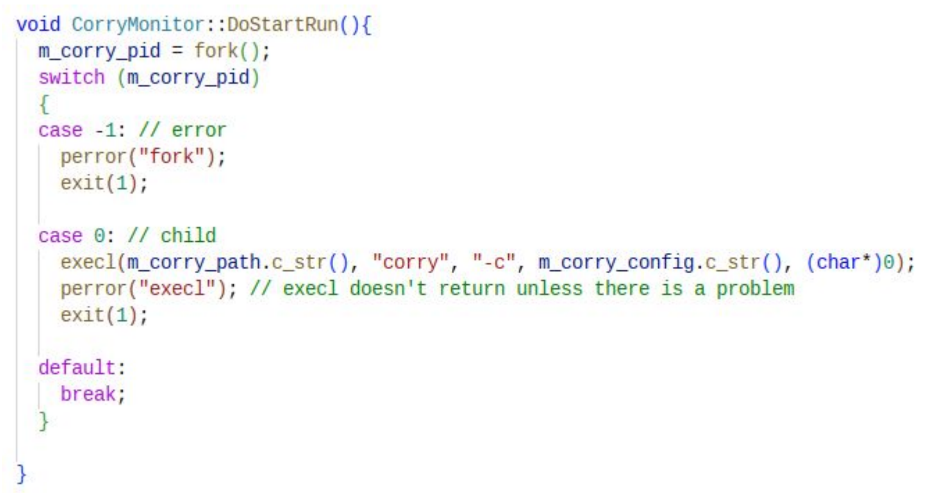
- Use corryvreckan from within eudaq for online monitoring of test beam
- Want to make it as convenient as possible for the user
- First tests seem to be working as intended
- But still a Work In Progress
- On track to roll auto beta version to dedicated testers

This project has received funding from the European Union's Horizon 2020 Research and Innovation programme under GA no 101004761.

Backup-Slides

CorryMonitor

- To be able to control corry from eudaq: call via fork
- can use procID for any manipulation

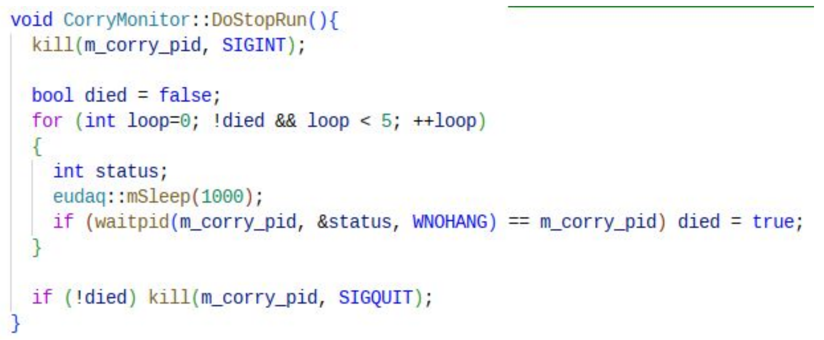


```
void CorryMonitor::DoStartRun(){
  m_corry_pid = fork();
  switch (m_corry_pid)
  {
  case -1: // error
    perror("fork");
    exit(1);

  case 0: // child
    execl(m_corry_path.c_str(), "corry", "-c", m_corry_config.c_str(), (char*)0);
    perror("execl"); // execl doesn't return unless there is a problem
    exit(1);

  default:
    break;
  }

}
```



```
void CorryMonitor::DoStopRun(){
  kill(m_corry_pid, SIGINT);

  bool died = false;
  for (int loop=0; !died && loop < 5; ++loop)
  {
    int status;
    eudaq::mSleep(1000);
    if (waitpid(m_corry_pid, &status, WNOHANG) == m_corry_pid) died = true;
  }

  if (!died) kill(m_corry_pid, SIGQUIT);
}
```