Additional features in EUDAQ2

Sohail AMJAD

UCL

AIDA-2020 3rd Annual Meeting
Bologna, Italy.

Apr 24, 2018

Proposed features

- **GUI Improvements:** To improve the Graphical User Interface by introducing new "states" and updating the RunControl GUI depending on status of the EUDAQ.
- **Producer Standalone Run:** To allow for the testing of the producer component without running full EUDAQ. This will allow efficient debugging of the "producer" while saving time.
- `.xml` **Configuration files**: To allow a more nested description of the configuration and initialisation parameters. Requires internal parsing and validation of the xml file.
- **ROOT TTree Converter:** Add a new format for data conversion to be used for different purposes.

Data Conversion in EUDAQ2

Currently two types of conversion are supported in EUDAQ2:

- **RawEvent2StdEvent:** It converts a Raw Event to Standard Event for pixel detectors. Historically it represents a tracking-hit event in sensor planes of the Beam Telescope.
- **RawEvent2LCEvent:** It converts a Raw Event to LCIO format. The Converter uses the raw event structure and fills parameters to a pre-defined `LCEvent` class. The `LCWriter` helps to write the output file.

TTree Converter - Event in EUDAQ2

- `eudaq::Event` is the base class. All other event types (for example for Data Converers) are to be derived from this class.
- It is created by the *Producer* and fed physics data from hardware.
- No prior knowledge of detector specific data structure. The `m_block` parameter of the Event holds what is only known to the hardware/detector.

Event Structure in EUDAQ2

An Event in the EUDAQ2 contains following variables.

variable	C++ type	Description
m_type	uint32_t	event type
m_version	uint32_t	version
m_flags	uint32_t	flags
m_stm_n	uint32_t	device/stream number
m_run_n	uint32_t	run number
m_ev_n	uint32_t	event number
m_tg_n	uint32_t	trigger number
m_extend	uint32_t	reserved word
m_ts_begin	uint64_t	timestamp at the begin of event
m_ts_end	uint64_t	timestamp at the end of event
m_dspt	std::string	description
m_tags	std::map<std::string, std::string>	tags
m_blocks	std::map<uint32_t, std::vector<uint8_t>>	blocks of raw data
m_sub_events	std::vector<EventSPC>	pointers of sub events

ROOT TTree Conversion

Basic idea is to store the event member variables in a TTree.

- The parameters are stored in branches of the ttree.
- Branches for basic variables like run number, trigger number etc are pre-defined and are filled on the go.
- Blocks of data along with the block id need special treatment.
- Branches to store blocks and block ids are created as the blocks are found. There can be multiple of them in an event or none.
- The sub-events are also stored in the Tree.

Converter Flow

The conversion works in following order.

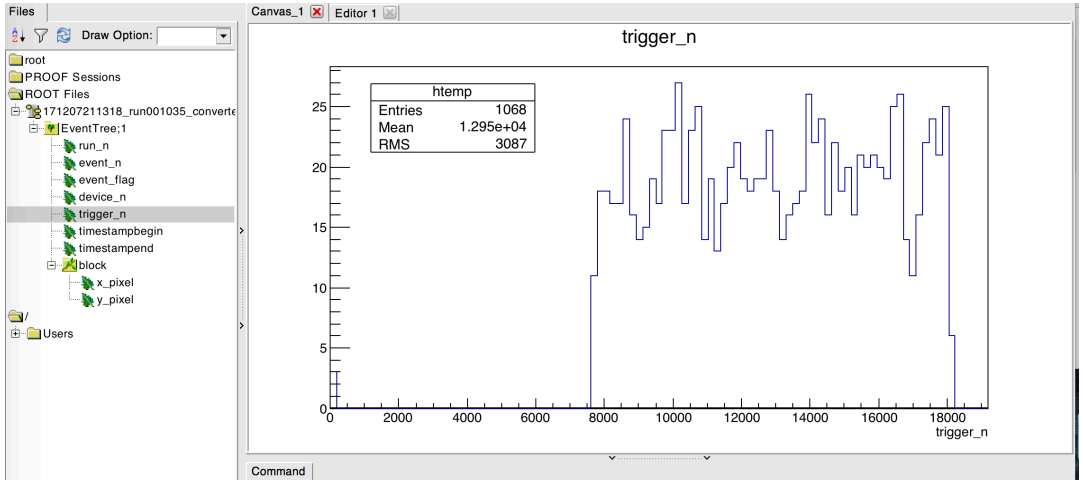
- The TTreeFileWriter contains pre-booked Tree Branches, prepares the output file and writes the TREE to it.
- TTreeEventConverter is the core converter for basic parameters of the `EUDAQ::RawEvent`
- RawEvent2TTreeConverter is called by EventConverter for conversion of ExtendedEvent (Sub structure in the Event).
- Ex0RawEvent2TTreeEventConverter does the conversion of data blocks and block ids. This part of the conversion is user specific.
  - As EUDAQ is unaware of the data structure, for moment we just put block and block ids in the TTree branches.
  - This part of the converter can be modified by the users to meet their needs.

Execution

- The conversion can be accomplished in similar way to other converters as for example LCConverter.
- The ROOT software is already set up during the EUDAQ2 compilation and installation.
- The command to execute is the following:
  `./euCliConverter -i inputfilename.raw -o outputfilename.root`
- For the moment the output file is stored to the disk. This behaviour can however be modified to use the output for other linked processes such as Monitoring.

Testing with AHCAL Data

The converter was successfully applied to AHCAL data from test beam. Basic parameters are converted as described before. The detector-specific data are treated as follows:



- The data contains two types of sub-events namely "CaliceObject" and "DESYTableRAW".
- A separate sub-converter which recognises each of these types and processes as per rules set by the user.

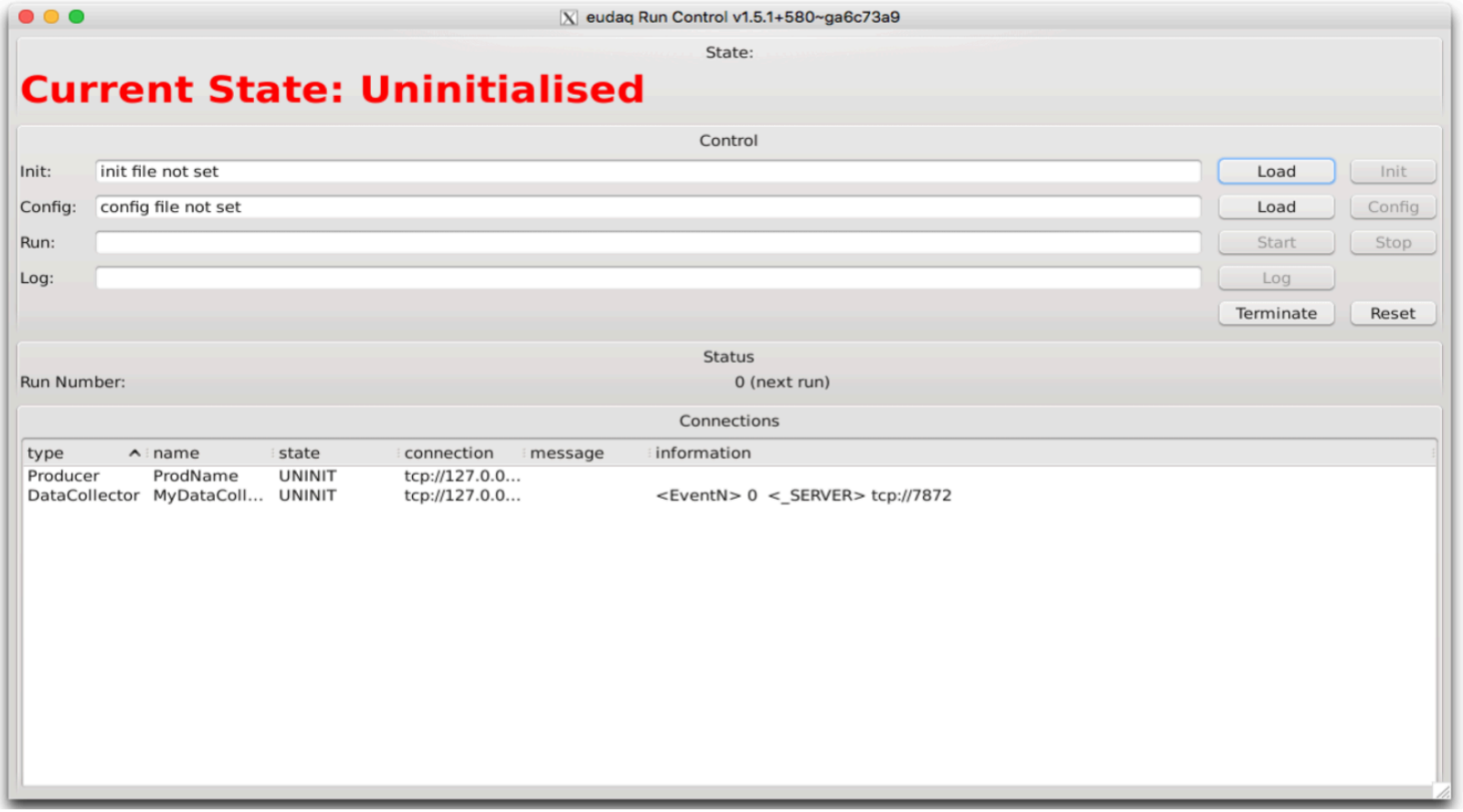
Summary and Plans

- The work is almost complete with testing done and documentation ready. Some final checks will be made before including in the central repository.
- Feedback is most welcome. The code can be shared upon request if required before a central release.
- The other additional features which have been mentioned before, are to be worked upon. The utility of those is to be assessed again while considering the effort it will take to implement those.
- More interfaces need to be developed, for example that to Si-W ECAL. Future combined beam tests involving complex new setups, such as that of CMS HGCAL and ILD AHCAL will be a test for the EUDAQ2.

Special Thanks to Yi Liu for his help during this work.

$\mathcal{THANKS}$

Improved GUI with state display



# of states increased, added `init` state, status display, multiple data collectors conflict, decoupling from RC.

Documentation Ready

TTree Converter in EUDAQ

EUDAQ2 Development Team

Sohail Amjad

Last update: February 2018

This document provides details of a working example of a converter from `raw` format to `ROOT TTree` format. It should be used as complementary to the EUDAQ manual. Any issues encountered during implementation of the instructions in this document should be reported at the dedicated platform.[1]

1 Event Structure in EUDAQ

In the EUDAQ data acquisition framework,`eudaq::Event` is the most important object in terms of data handling. It stores the data generated by the device under test (DUT). It is created by the `Producer` component of EUDAQ. The Producer then feeds physics data to fill it. It is the base class in EUDAQ and is serializable. Any object of type `Event` for any linked processes including Data Collector and Data Converter are to be derived from this class. The member variables of `eudaq::Event` are listed in table 1.

[1] https://github.com/eudaq/eudaq/issues