A generic ROOT monitoring tool for EUDAQ2

L. Forthomme (Helsinki Institute of Physics)

on behalf of the CMS and TOTEM Collaborations

8th Beam Telescopes and Test Beam workshop, Tbilisi, Georgia

January 27-31, 2020

General motivation

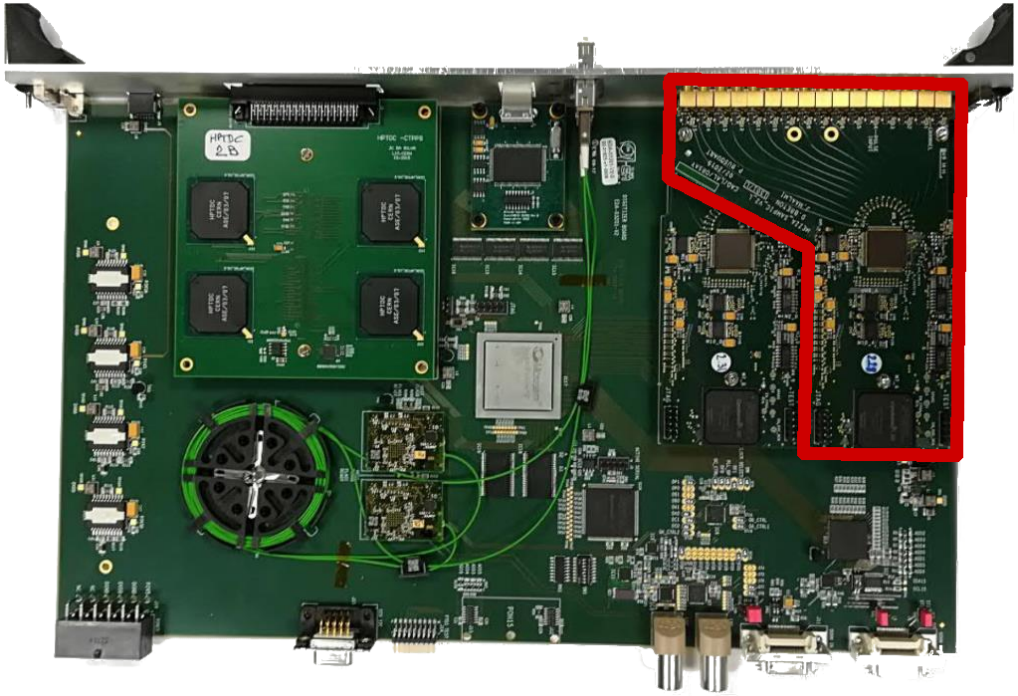
- **Scope**: CMS-TOTEM campaign for the testing of CMS Precision Proton Spectrometer (PPS) timing sensors at DESY-II test beam facilities
  - see E. Bossini's talk on Thursday for a general introduction to PPS, an in-depth description of DUTs and setup, and test beam results

- In need for a (fast...) and **scalable online monitoring tool** for:
  - spatial alignment of the DUT: "tomography" using EUTelescope's MIMOSAs, ... (feasible with EUDAQ1&2's `StdEventMonitor`)
  - characterisation of sensor performance (timing, signal shape and amplitude, ...)

- Two possible strategies considered:
  - large extension of `StdEventMonitor` to account for hardware- and user-specific variables monitoring (and subsequent extension to derivatives of the `eudaq::StandardEvent` object)
  - introduction of a **new monitoring tool** into the EUDAQ2 environment (**this presentation**)

- Developed, commissioned, and **integrated in EUDAQ since** `v2.4.2`

General motivation - PPS DUT readout & CMS-TOTEM motherboard



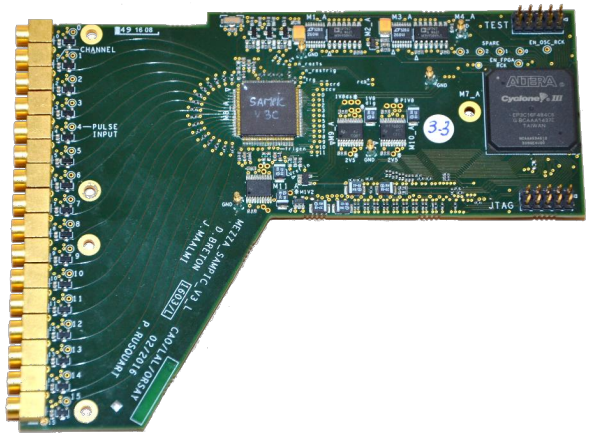
CMS-TOTEM digitiser board used intensively along LHC's run 2 operations

- readout: OptoRx optical link and/or USB 2.0 (Quick USB)
- designed to host up to two **16-channels Sampic mezzanines** (1 module highlighted in red)
- firmware (E. Bossini) handling digitiser sanity checks, frame building, and communication/data transfer

General motivation - Sampic ASIC (Nucl.Instrum.Meth. A787 (2015) 245)

Time- and waveform-to-digital (fast sampling) converter

- 16 channels/chip, up to 64 samples/hit at 10 GSa/s
- 1V ADCs dynamic range, full signal shaping allowing for extraction of advanced features (charge, amplitude, ...)
- 1.5 GHz bandwidth with 8 − 11 bits resolution (0.25 − 1.6 μs dead time/channel for full conversion)
- self-triggering, event building performed at the digitiser level
- 3.5 ps RMS time difference resolution after calibration



Integration of **slow control and readout software** into the EUDAQ2 environment (Aug-Sep 2019)

- DUT/MIMOSAs synchronisation handled through (FPGA) interfacing board propagating EUDET Trigger Logic Unit (AIDA-2020) to CMS-TOTEM digitiser board (→ Sampic event counter)
- basic `eudaq::Producer` handling slow control (steering through sequential ASCII configuration definition) and frames unpacking and recasting into EUDAQ's `StandardEvent` derivative
  - dynamic data format ($1 < n_{ch} < 32$/"event" frame)

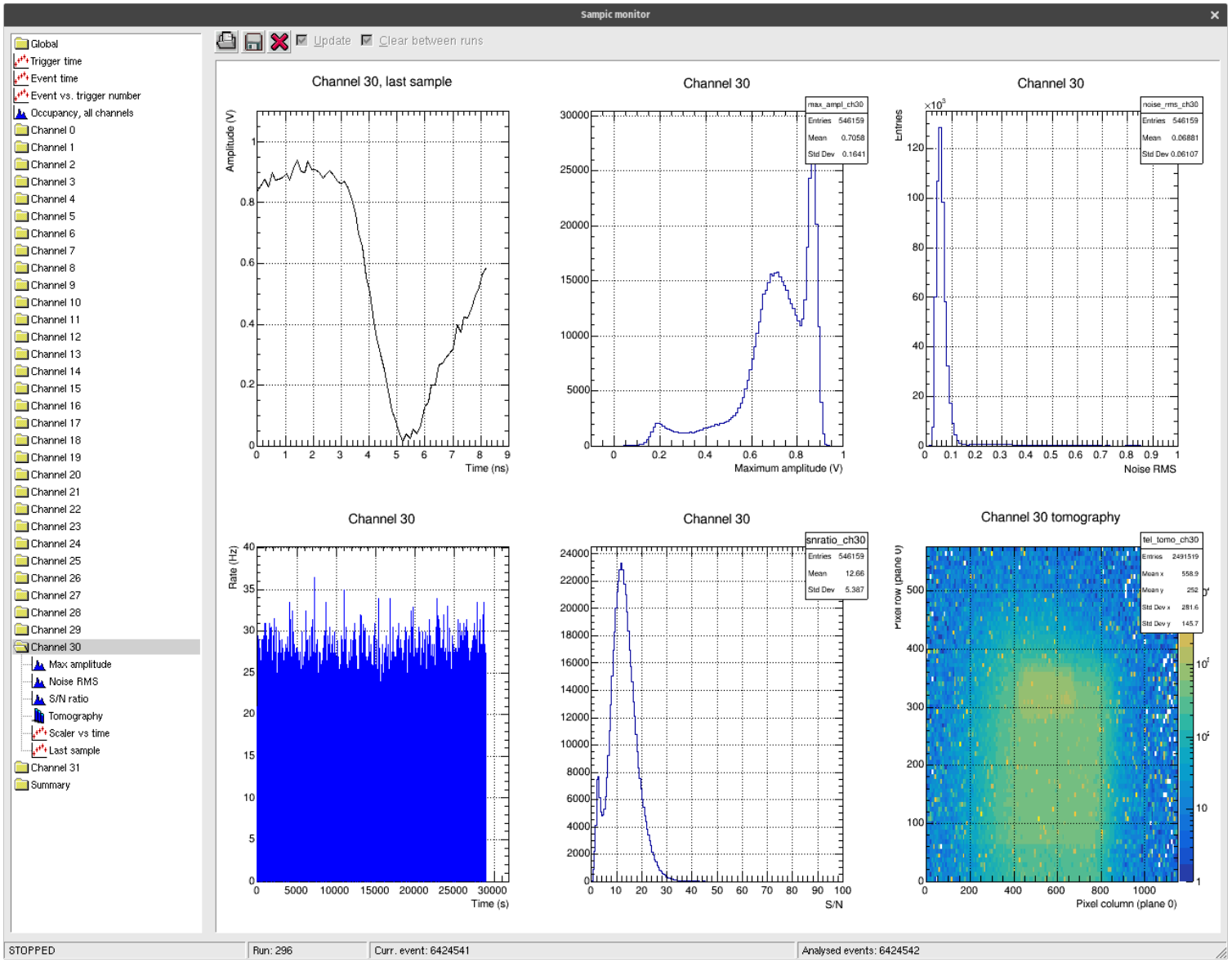
ROOT monitor for EUDAQ2

ROOT monitor

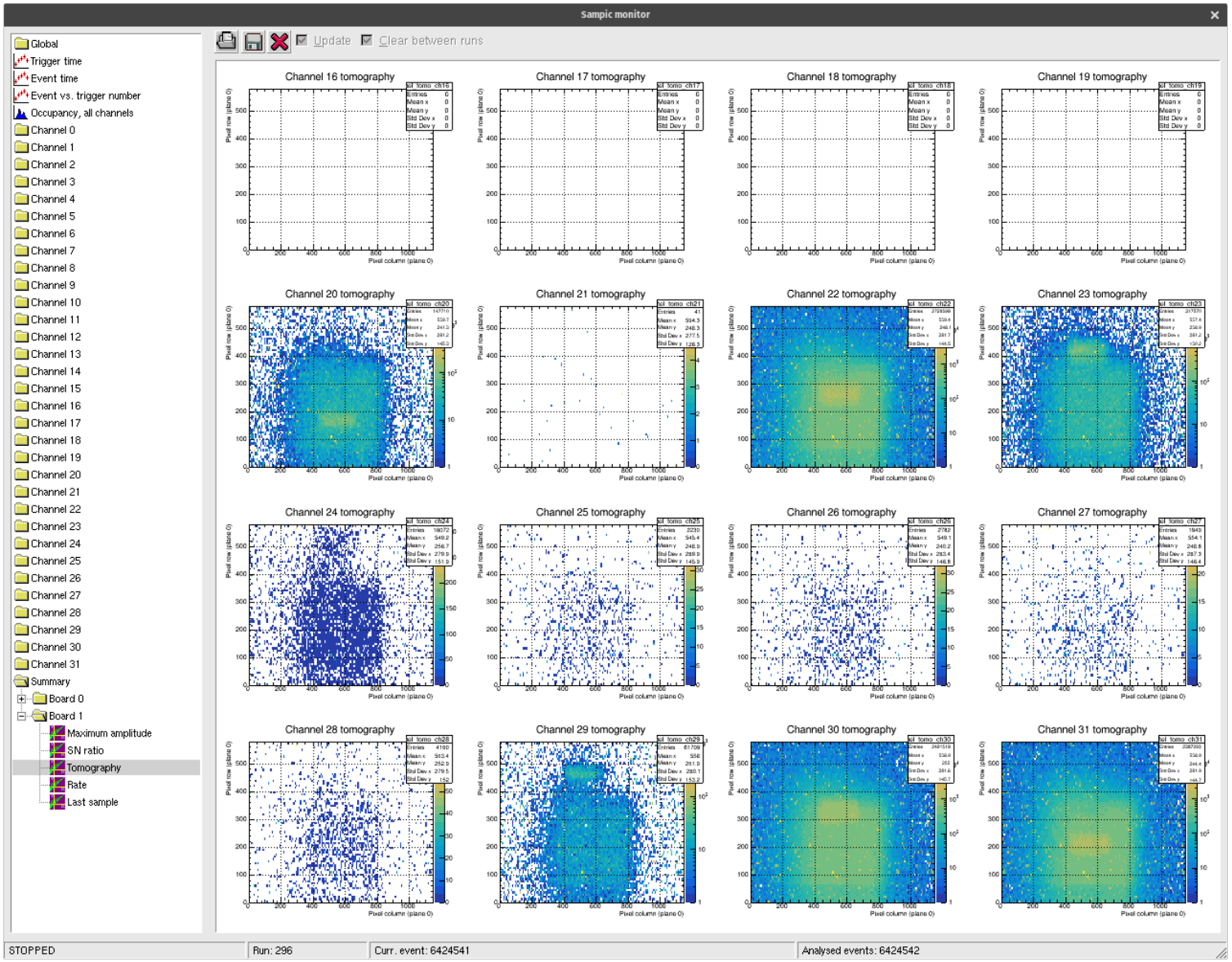
New **base class** for the definition of a GUI monitor

- standard ROOT `TApplication`, embedding EUDAQ2 TCP socket communication capability (commands/data stream/...)
- user configurable **"folder hierarchy"** for all monitored variables
  - each **single monitor**, or **summaries of multiple vistars** (directory view) can be displayed
  - allows combination of monitors from different folder into global summaries (e.g. single monitor for all channels)
- allows to output a hierarchised ROOT file containing all monitors after integration over full run
- **"playback mode"**: asynchronous loading of RAW EUDAQ files, or monitor output ROOT files
  - allows emulating the data collection stage in offline mode, and **reproducing the monitoring** under different conditions (binning, ranges, ...)
  - designed as a tool for **follow-up developments** between two test beams campaigns
- not meant as a replacement for EUDAQ1&2's `StdEventMonitor`, and may be run in parallel
  - possible re-implementation into this new scheme, though...

A live example: Sampic event monitor (single-channel summary)



A live example: Sampic event monitor (global per-monitor summary)



“Hands-on”

Monitor declaration

A minimal working example: declare a monitor with a few generic ROOT objects as monitored collections

- e.g. a 1-dimensional histogram, a 2-dimensional graph, and a profile histogram

```
#include "eudaq/ROOTMonitor.hh" // base dependency
// all required ROOT dependencies
#include "TH1.h"
#include "TH2.h"
#include "TGraph2D.h"
#include "TProfile.h"

class MyROOTMonitor : public eudaq::ROOTMonitor {
public:
  MyROOTMonitor(const std::string& name, const std::string& runcontrol):
    eudaq::ROOTMonitor(name, "Ex0 ROOT monitor", runcontrol){}
  void AtConfiguration() override;
  void AtEventReception(eudaq::EventSP ev) override;

  static const uint32_t m_id_factory = eudaq::cstr2hash("MyROOTMonitor");
private:
  TH1D* m_my_hist;
  TGraph2D* m_my_graph;
  TProfile* m_my_prof;
};
```

Monitor implementation

Register the new monitor into the runtime database:

```
namespace{
  auto my_mon = eudaq::Factory<eudaq::Monitor>::Register<MyROOTMonitor, const std::string&,
        const std::string&>(MyROOTMonitor::m_id_factory);
}
```

Build any ROOT's `TObject`-derivative for monitoring, using the templated booking method from the `m_monitor` protected member inherited from `eudaq::ROOTMonitor` base class:

```
template<typename T, typename... Args> T* Book(const std::string& path, const std::string&
      name, Args&&... args);
```

Build all monitored objects at e.g. the `eudaq::ROOTMonitor::AtConfiguration` declaration:

```
void MyROOTMonitor::AtConfiguration(){
  m_my_hist = m_monitor->Book<TH1D>("Channel 0/my_hist", "Example 1D histogram",
    "h_example", "A histogram;x-axis title;y-axis title", 100, 0., 1.);
  m_monitor->SetDrawOptions(m_my_hist_2d, "lego");
  m_my_graph = m_monitor->Book<TGraph2D>("Channel 0/my_graph", "Example graph");
  m_my_graph->SetTitle("A graph;x-axis title;y-axis title;z-axis title");
  m_monitor->SetDrawOptions(m_my_graph, "colz");
  m_my_prof = m_monitor->Book<TProfile>("Channel 0/my_profile", "Example profile",
    "p_example", "A profile histogram;x-axis title;y-axis title", 100, 0., 1.);
}
```

Monitor implementation

Monitors populated through a callback at each standard event reception

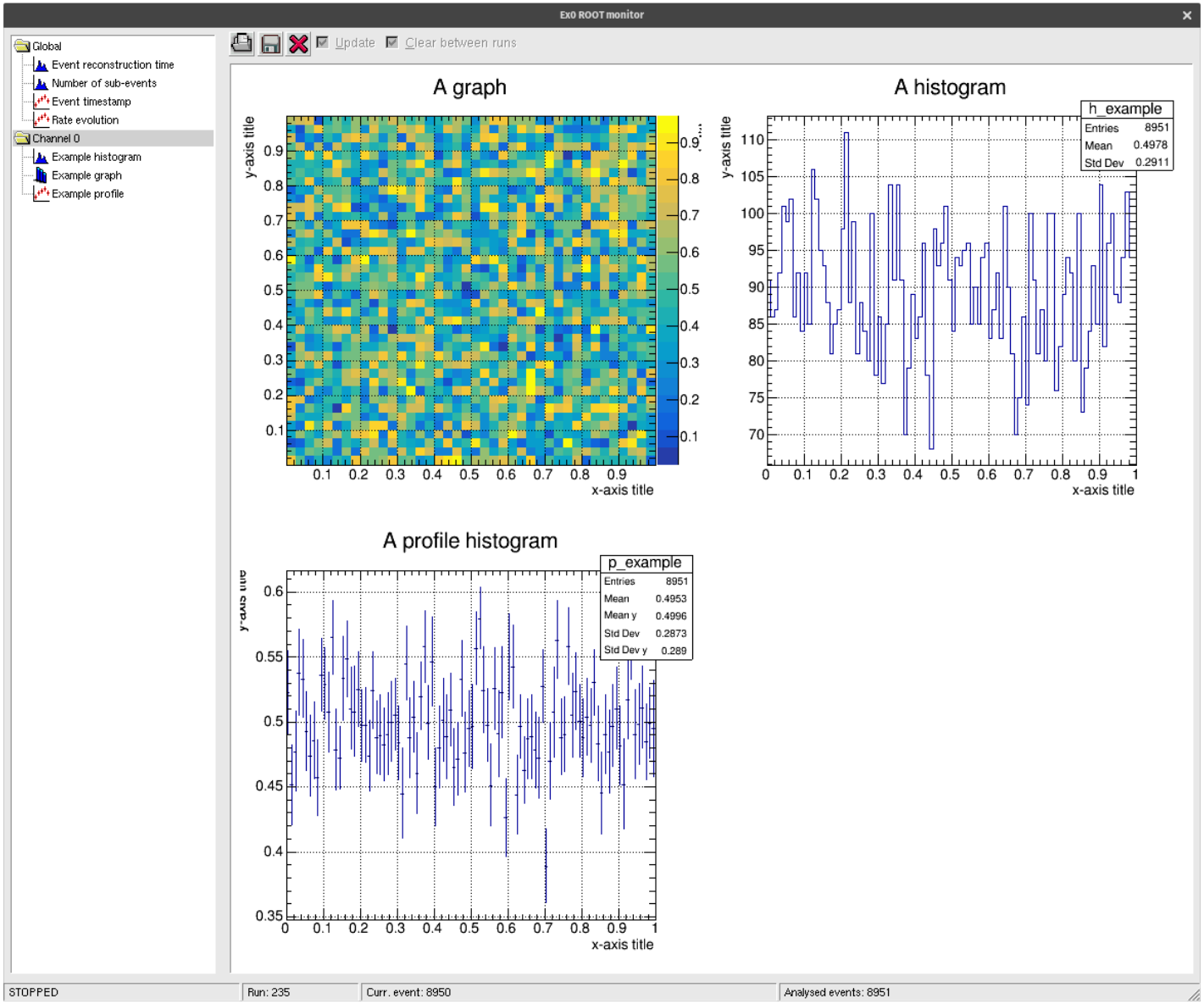
- if needed with proper casting to user-defined data format

```
void MyROOTMonitor::AtEventReception(eudaq::EventSP ev){
  auto event = std::make_shared<MyEventDataFormat>(*ev);
  // let there be a user-defined MyEventDataFormat, containing e.g. three
  // double-precision attributes get'ters:
  //   double GetQuantityX(), double GetQuantityY(), and double GetQuantityZ()
  m_my_hist->Fill(event->GetQuantityX());
  m_my_graph->SetPoint(m_my_graph->GetN(),
    event->GetQuantityX(), event->GetQuantityY(), event->GetQuantityZ());
  m_my_prof->Fill(event->GetQuantityX(), event->GetQuantityY());
}
```

Other overridable members:

- two global `void AtInitialisation()`/`void AtReset()` methods,
  - e.g. resetting accumulative variables, re-booking histograms, ...
- two "run-local" `void AtRunStart()`/`void AtRunStop()` methods
  - e.g. prepare and write to a per-run external dump file, R/W access to conditions database, ...

A sample monitor - output



Summary and next steps

- Integration of the CMS-TOTEM motherboard into the EUDAQ2 environment
  - Successfully tested last September at DESY-II test beam facilities, no synchronisation loss observed
- New tool developed for the fast monitoring of basic, and complex observables at test beams
  - Designed to fit EUDAQ users' needs, can be tailored for all data formats
  - GUI building as overhead, allows to concentrate on the monitors...

Work ongoing

- Integration of the DUT reconstruction in the EUTelescope data analysis framework for the synchronised MIMOSAs+DUT event content
- Helper for the definition of variables correlation monitoring (DUT $\leftrightarrow$ EUTelescope, DUT1 $\leftrightarrow$ DUT2, ...)
- "Trivial" monitors for EUDAQ's `StandardEvent` and `TTreeEvent`-compatible data formats
  - further motivation for users to develop converters in their workflow

The measurements leading to these results have been performed at the Test Beam Facility at DESY Hamburg (Germany), a member of the Helmholtz Association (HGF)