# Data Quality Monitoring for High Energy Physics (DQM4HEP) Version 03-02-00

**R. Été, A. Pingault, L. Mirabito**

Université Claude Bernard Lyon 1 - Institut de Physique Nucléaire de Lyon / Ghent University

3 février 2016


# Overview

DQM4HEP : an online monitoring system for data quality

## Key points

- Event distributed system : server/client paradigm
- Set of interfaces for data analysis, adapted to DQM purpose
- Histogram distributed system
- Visualization interface (Qt GUI)
- Large scale process management
- IO support for any data type (opt. LCIO)
- Full size detector (ILD) to single prototype (CALICE)
- ELog interface

Set of interfaces inspired from CMS DQM system (monitor elements, collectors).

Application flow inspired from ALICE DQM system, AMORE (cycles).

# Overview

## DQM4HEP packages

One location : `https://github.com/DQM4HEP`

### The main package : DQM4HEP

Installation package for sub-packages (CMake).

Sub-packages :

- **dim** : Distributed Information Management (Delphi). Manage client/server communications
- **dimjc** : DIM Job Control (L. Mirabito). Process management using dim.
- **jsoncpp** : Json I/O for dimjc
- **streamlog** : logging library (used in ILCSOFT)
- **DQMCore** : Core part of the DQM system. Client/server interfaces, analysis, IO, run control interface, plugin management ...
- **DQMViz** : Qt visualization interfaces. Job control gui client, monitoring gui client, run control server gui (standalone).
- **LCIO** : Linear Collider IO. Build support for LCIO streamer

# Overview

## Installation

### Installation mode

Designed to be built **standalone** or using **ILCSOFT**.
Basic install requires ROOT.
Full install with DQMViz requires Qt and ROOT **compiled with *–enable-qt* option**.

Standalone mode :

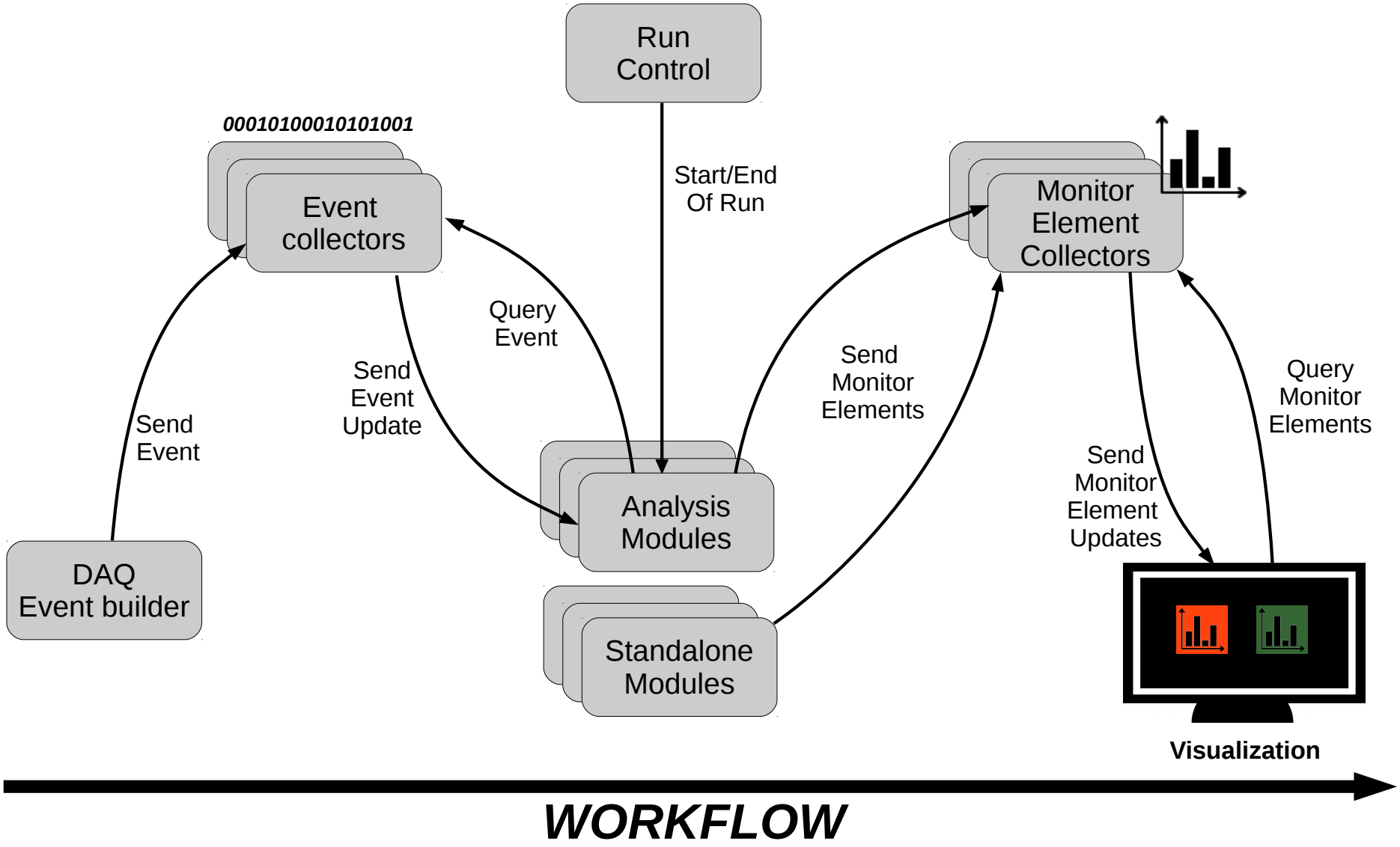
- Basic install : dim, dimjc, jsoncpp, streamlog, DQMCore
- Full install : + DQMViz, LCIO

ILCSOFT mode :

- Basic install : dim, dimjc, jsoncpp, DQMCore
- Full install : + DQMViz

# Architecture

## Global workflow



# Architecture

## Event collectors, client/server

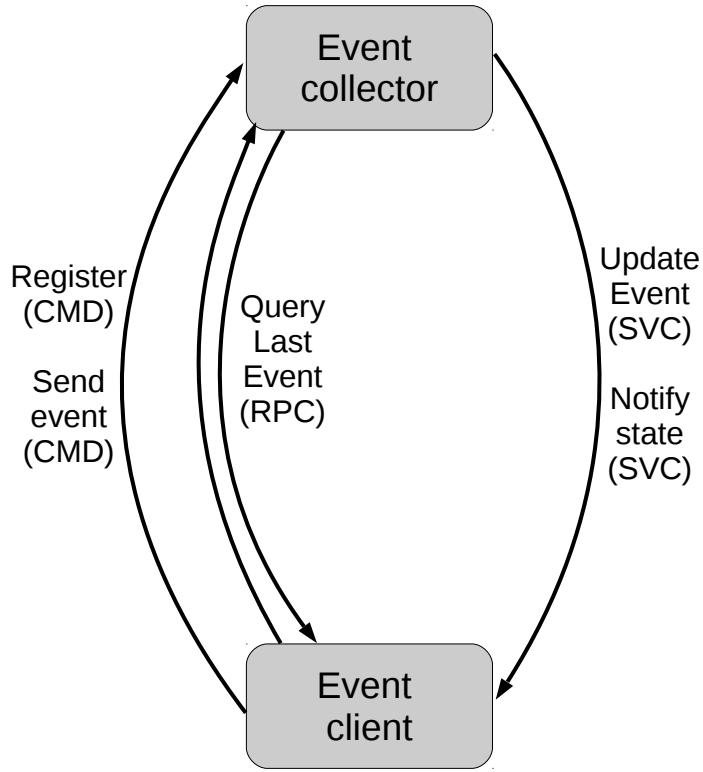
Event collector (`DQMEventCollector`) and
Event client (`DQMEventClient`)
linked via DIM (TCP/IP).

Receiver interface with 2 modes :

- on update
- on query

Sender interface with one unique command to send an event to the collector server

Use `dqm4hep_start_event_collector` to start a collector server.



# Architecture

## Module applications - analysis module

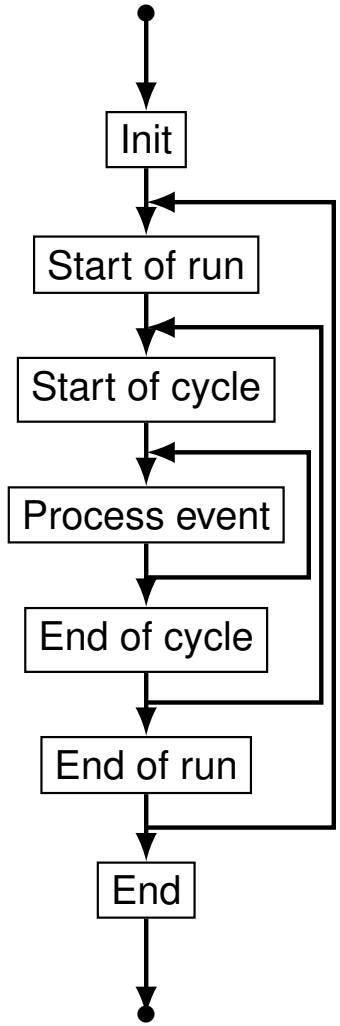
### Purpose

- Receive events from a collector server and process them
- Produce monitor elements (histograms, scalars, generic TObject)
- Follow the run control signals (SOR, EOR)

- **Init** : Initialize the application : load dlls, declare services, etc ... Wait for a SOR
- **Start of run** : start cycles loop, open archive
- **Start of cycle** : start a cycle of '*process event*'
- **Process event** : Process incoming event, fill monitor elements, etc ...
- **End of cycle** : send subscribed monitor elements, update archive (opt).
- **End of run** : Wait for SOR, close archive (opt).
- **End** : Clean and exit module.

To implement online DQM analysis, user must implement the `DQMAnalysisModule` interface. A shared library must be build and loaded in the application using the plugin system (see next slides).

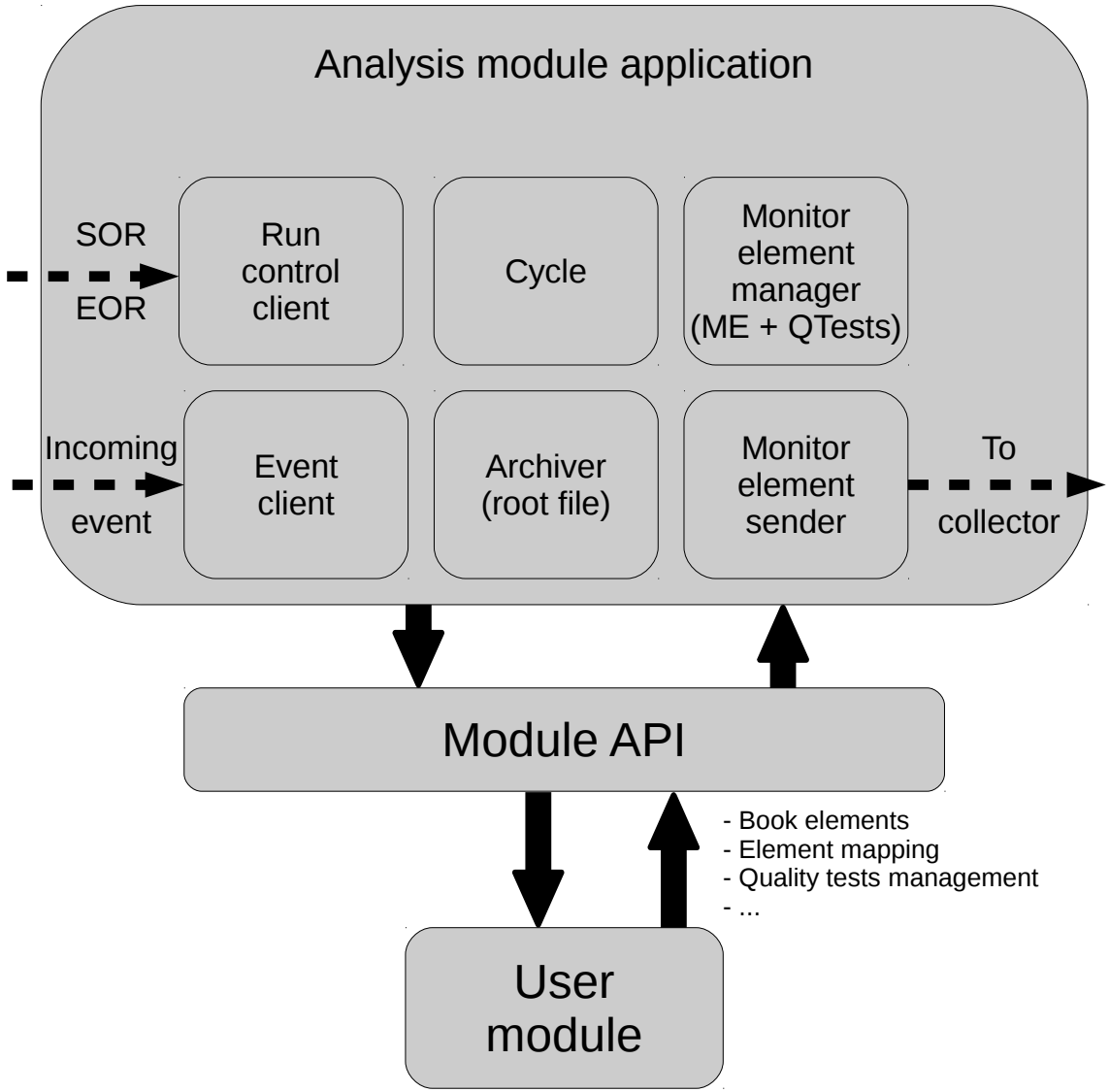
Use `dqm4hep_start_analysis_module` to start an analysis module.




Analysis module application flow

# Architecture

## Module API



# Architecture

## Module applications - standalone module

### Purpose

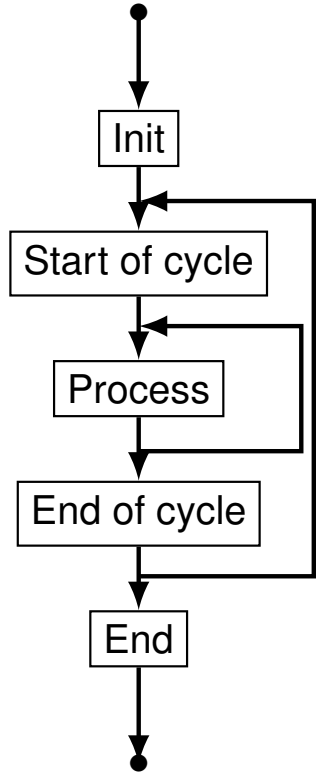
- No event reception
- No run signals
- Produce monitor elements (histograms, scalars, generic TObject)

- **Init** : Load dlls, init the module.
- **Start of cycle** : start a timer cycle of n seconds
- **Process** : call back function.
- **End of cycle** : collect monitor elements and send
- **End** : The application has received a signal to exit and the process ends.

To implement online standalone analysis, user must implement the `DQMStandaloneModule` interface. A shared library must be build and loaded in the application using the plugin system (see next slides).

Designed for *slow control - like* data processing.

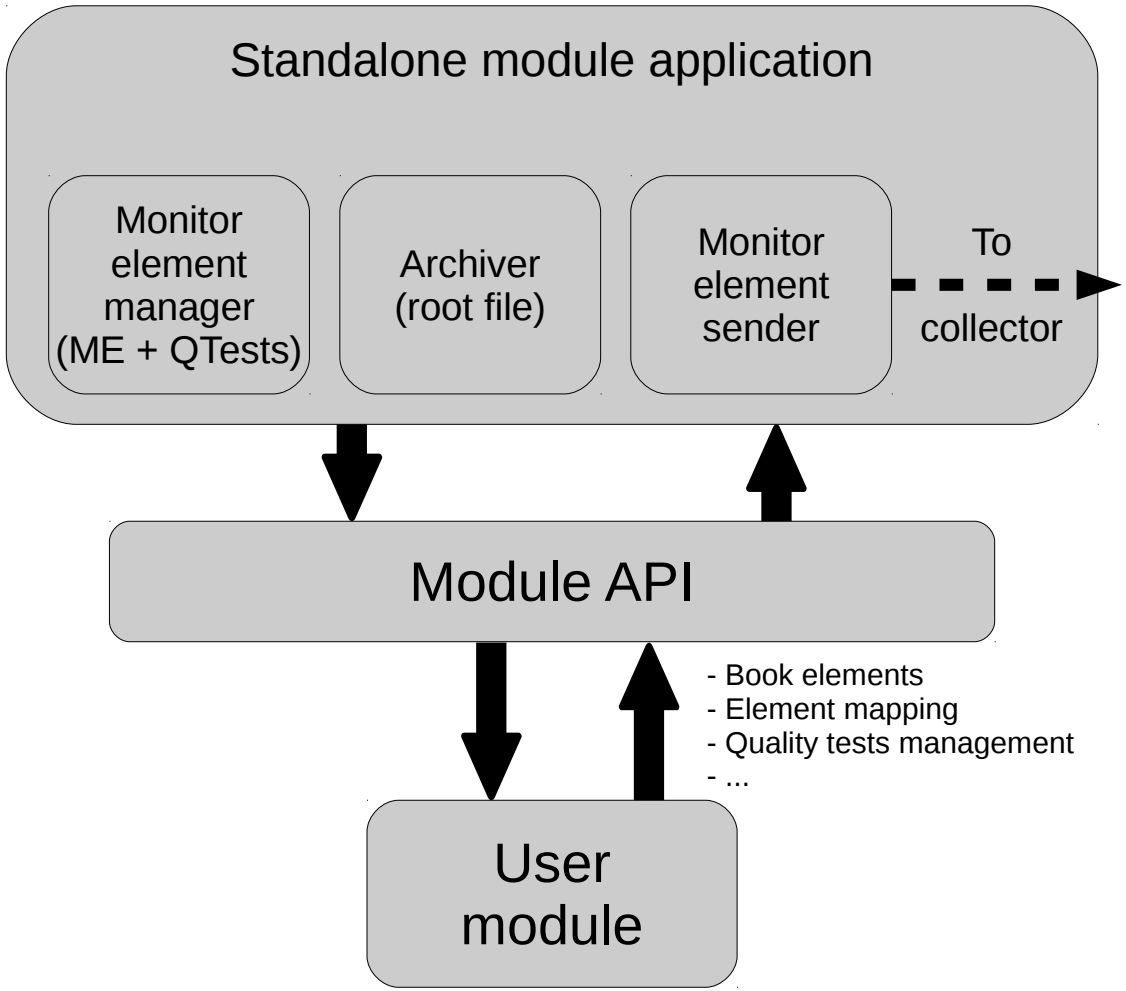
Use `dqm4hep_start_standalone_module` to start a standalone module.




Standalone module application flow

# Architecture

## Module API



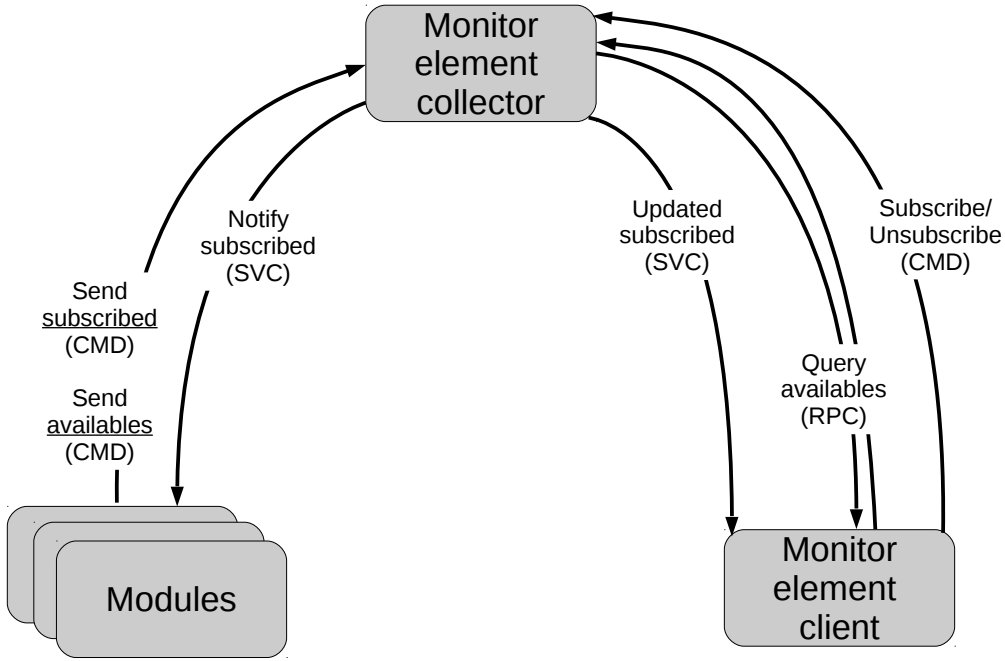
# Architecture

## Monitor element collector, client/server

Collect monitor elements from different modules.

Publish/subscribe paradigm with client side.

Collector stores the available list of elements (names) from the different modules and the elements that the clients have subscribed.




Client interface works on both update and query modes.

Use `dqm4hep_start_monitor_element_collector` monitor element collector.

# User interfaces overview

## Plugin system

The core part of the system provides a plugin system, massively used by the framework to handle user classes.

The `DQMPluginManager` singleton class manages the plug-ins provided by the system and the users. Plugins can be loaded at any time by loading shared libraries.

Example :

```
// single library loading
DQMPluginManager::instance()->loadLibrary("libMyClass.so");
// multiple libraries loading
StringVector libraries = { "libMyClass.so" , "libAnotherClass.so" }
DQMPluginManager::instance()->loadLibraries(libraries);
// using DQM4HEP_PLUGIN_DLL env var with ':' separation
// assuming export DQM4HEP_PLUGIN_DLL=./lib/libSuperClass.so:./lib/libDirtyClass.so
DQMPluginManager::instance()->loadLibraries();
```

In principle **any** user class can be plugged in the framework and retrieved inside applications.

# User interfaces overview

## Plugin system

User Example :

```
// MyClass.h
class MyClass
{
  public:
    MyClass(); // Default constructor mandatory !!
  private:
    int     m_attribute;
};
// MyClass.cc
DQM_PLUGIN_DECL( MyClass , "MyClass" ) // plug the class in the system

MyClass::MyClass() : m_attribute(0) {}
// ...
```

To get an instance of your class, use the plugin manager :

```
// ...
MyClass *pClass = DQMPluginManager::instance()->createPluginClass<MyClass>("MyClass");
// ...
```

For example, this functionality is used internally to get :

- user module implementations
- event streamers
- run control clients

# User interfaces overview

## Streaming interface

**The framework has no dependency on the type of data transferred over the network !**

For example, the streamer for LCIO is implemented and provided as a plug-in in the software. The type of data that is transferred over the network can be user defined.
Users have to implements the `DQMEventStreamer` interface :

```
class DQMEventStreamer : public DQMStreamer<DQMEvent>
{
public:
  // ...
  virtual StatusCode serialize(const DQMEvent *const pEvent, DQMDataStream *const pDataStream) = 0;
  virtual StatusCode deserialize(DQMEvent *&pEvent, DQMDataStream *const pDataStream) = 0;
  virtual StatusCode serialize(const DQMEvent *const pEvent, const std::string &subEventIdentifier,
        DQMDataStream *const pDataStream) = 0;
};
```

and plug the streamer class using the plugin mechanism. The `DQMDataStream` class provides functions to read and write raw buffers.

For an experiment with a simple event structure, it can be useful to define a raw event streamer and propagate the event through the DQM system without taking care about the network interface.

# User interfaces overview

## Module API

Data processing performed in **modules** (standalone or analysis).
Modules **book** monitor elements, **fill** them and **publish** them to a single collector.

A monitor element is a wrapper around a ROOT TObject with some additional attributes :
$\rightarrow$ Type, name, path (i.e "/Efficiency/Layer2/"), collector name, quality flag, reset policy, title, description, run number, quality test results.

The `DQMModuleApi` class provide a static interface to perform operations within the application :

- Monitor elements management (book, delete, reset, from xml)
- Directory structure management (mkdir, cd, ls, rmdir, pwd)
- Quality test management (register, add, remove, run, from xml)

Quality tests can be run on a particular monitor element to test the quality of the processed data (chi2, Kolmogorov, user defined).

Note that **QTest results are sent to the collector together with the monitor element** !

# User interfaces overview

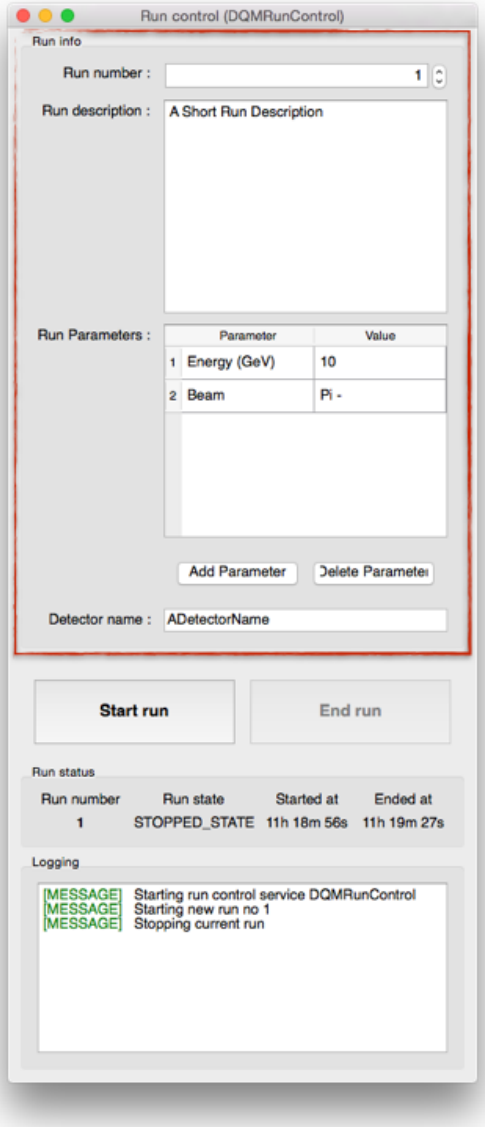
## Gui visualisation

Gui interfaces for DQM client developed :

- Run control, job control, online monitoring
- Written with Qt4 framework 
- Easily configurable with json and xml.

# User interfaces overview

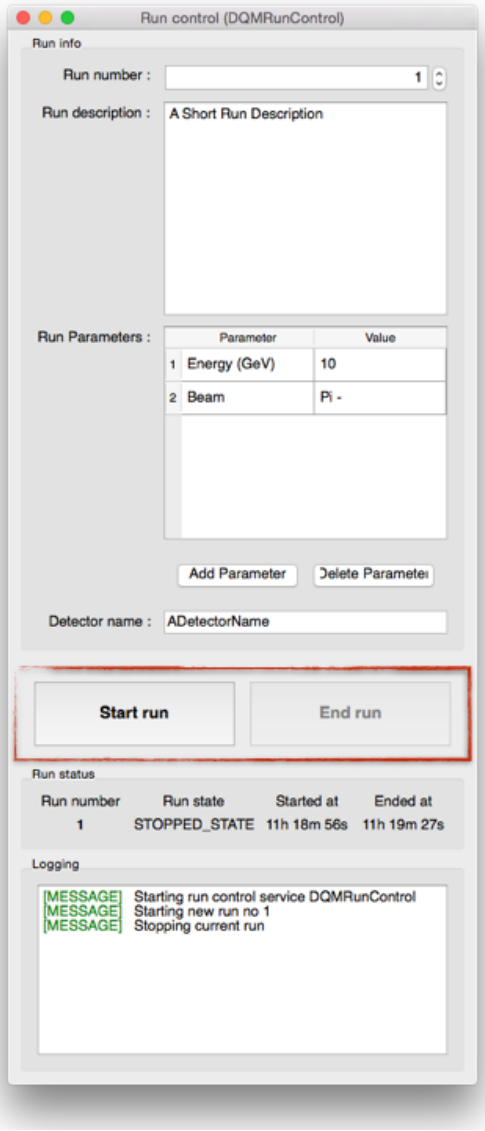
## Run Control GUI




- Parametrisation of run with run number, detector name, run description and parameters

# User interfaces overview

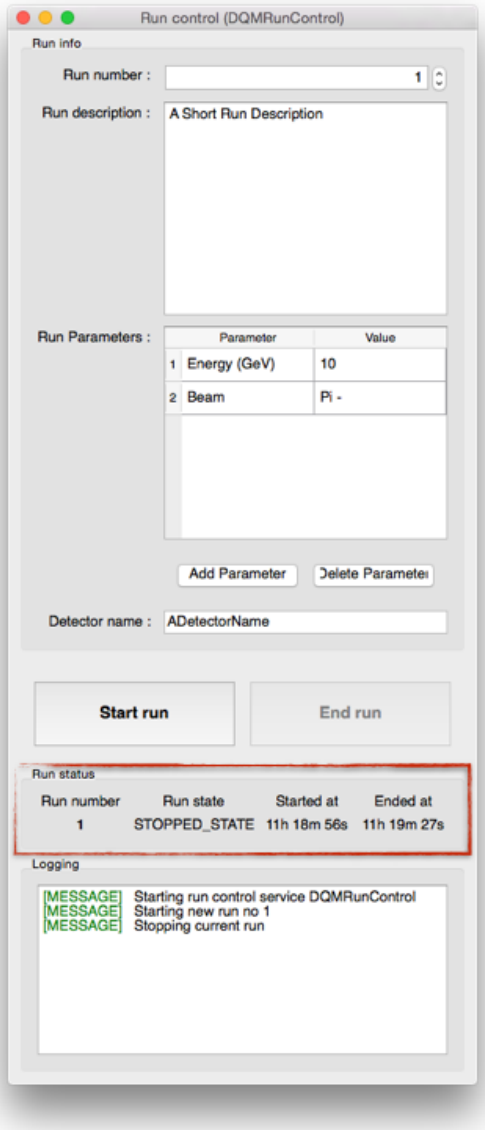
## Run Control GUI




- Parametrisation of run with run number, detector name, run description and parameters
- Send SOR and EOR signals

# User interfaces overview

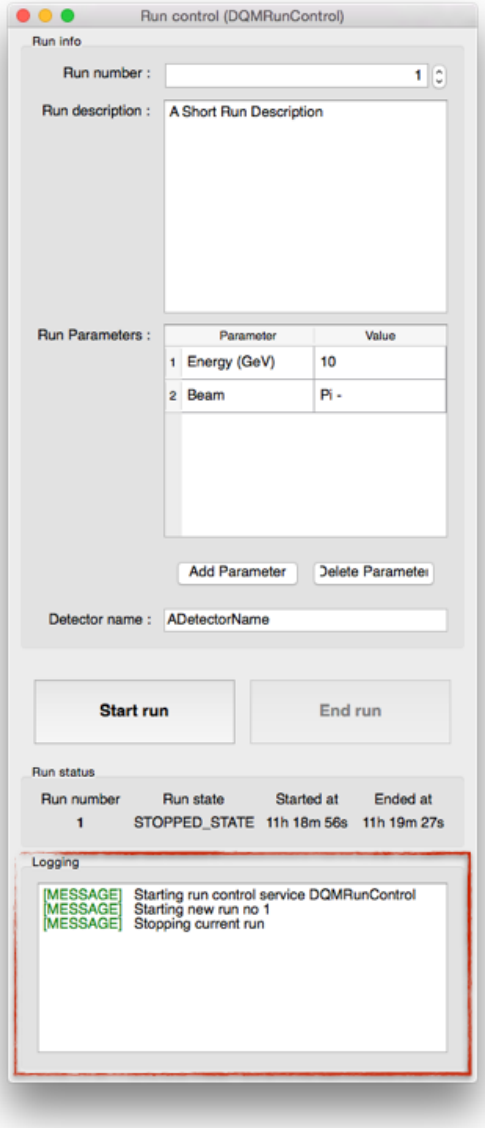
## Run Control GUI




- Parametrisation of run with run number, detector name, run description and parameters
- Send SOR and EOR signals
- Control run status ( State, Started/Stopped time )

# User interfaces overview

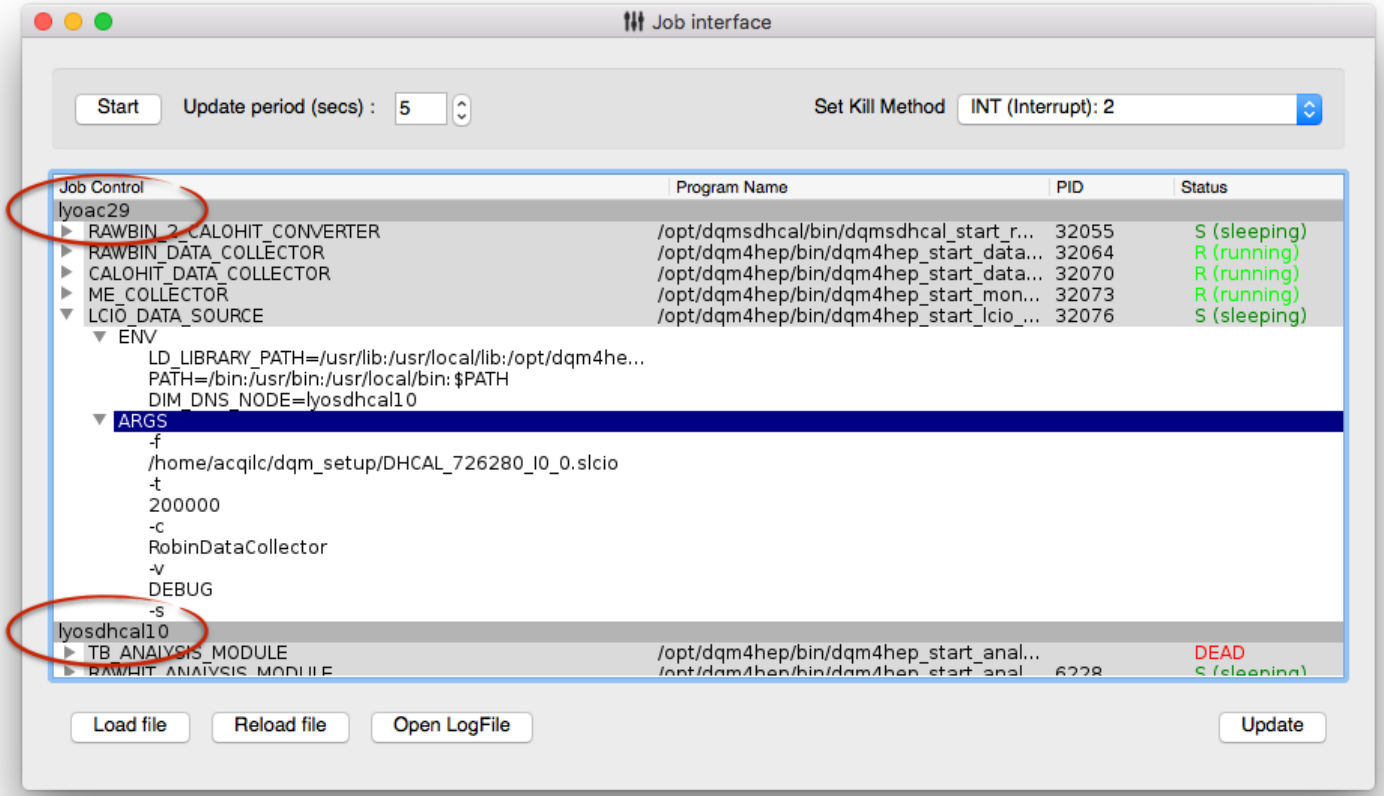
## Run Control GUI




- Parametrisation of run with run number, detector name, run description and parameters
- Send SOR and EOR signals
- Control run status ( State, Started/Stopped time )
- Every action is logged for easy information overview

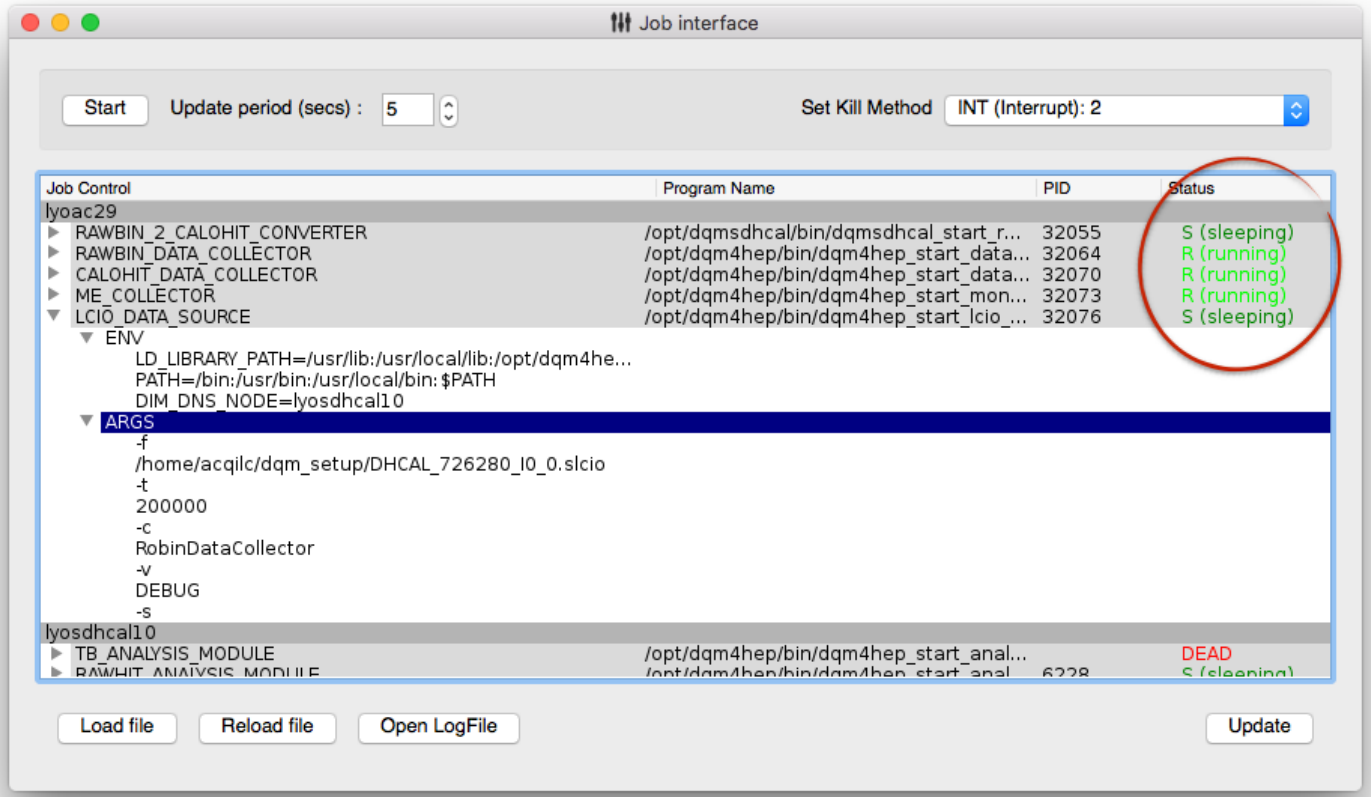
# User interfaces overview

## Job Control GUI



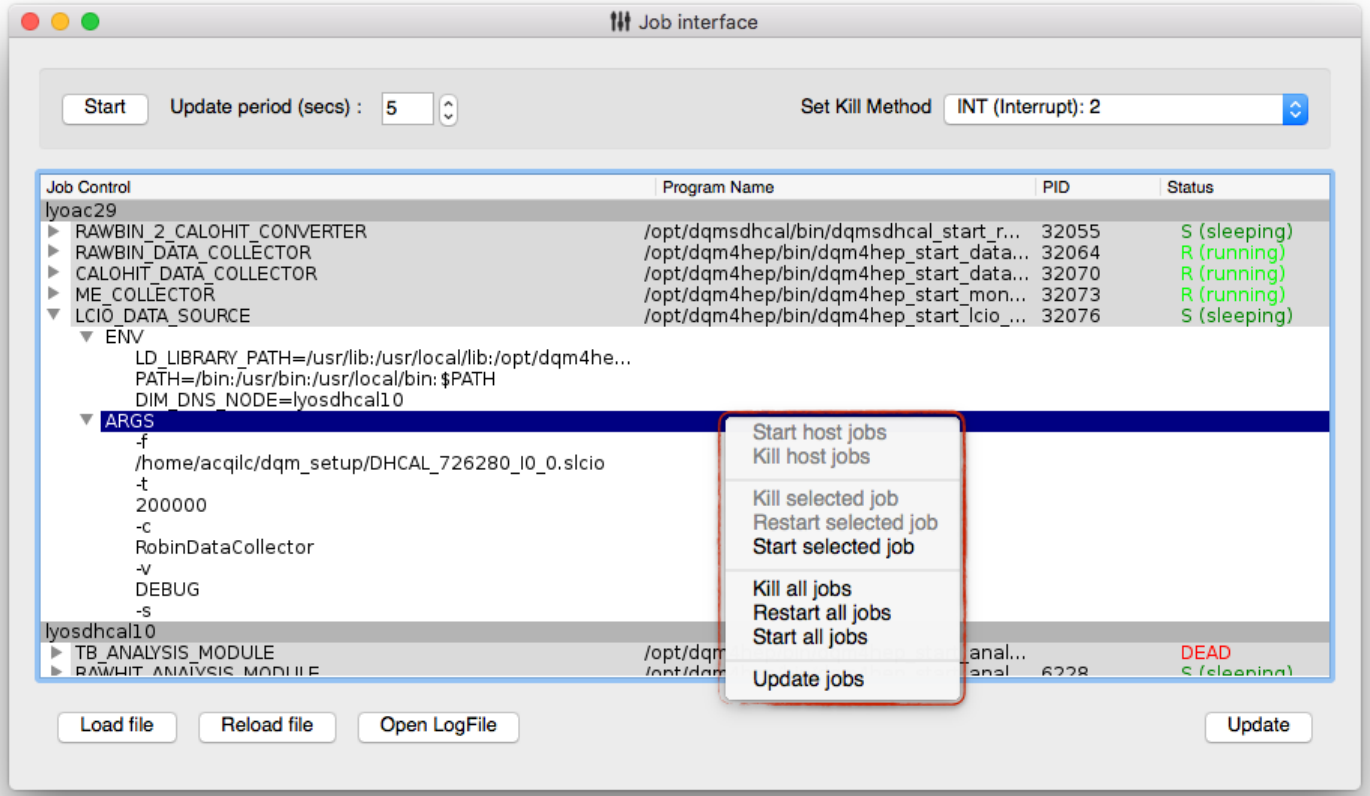
# User interfaces overview

## Job Control GUI



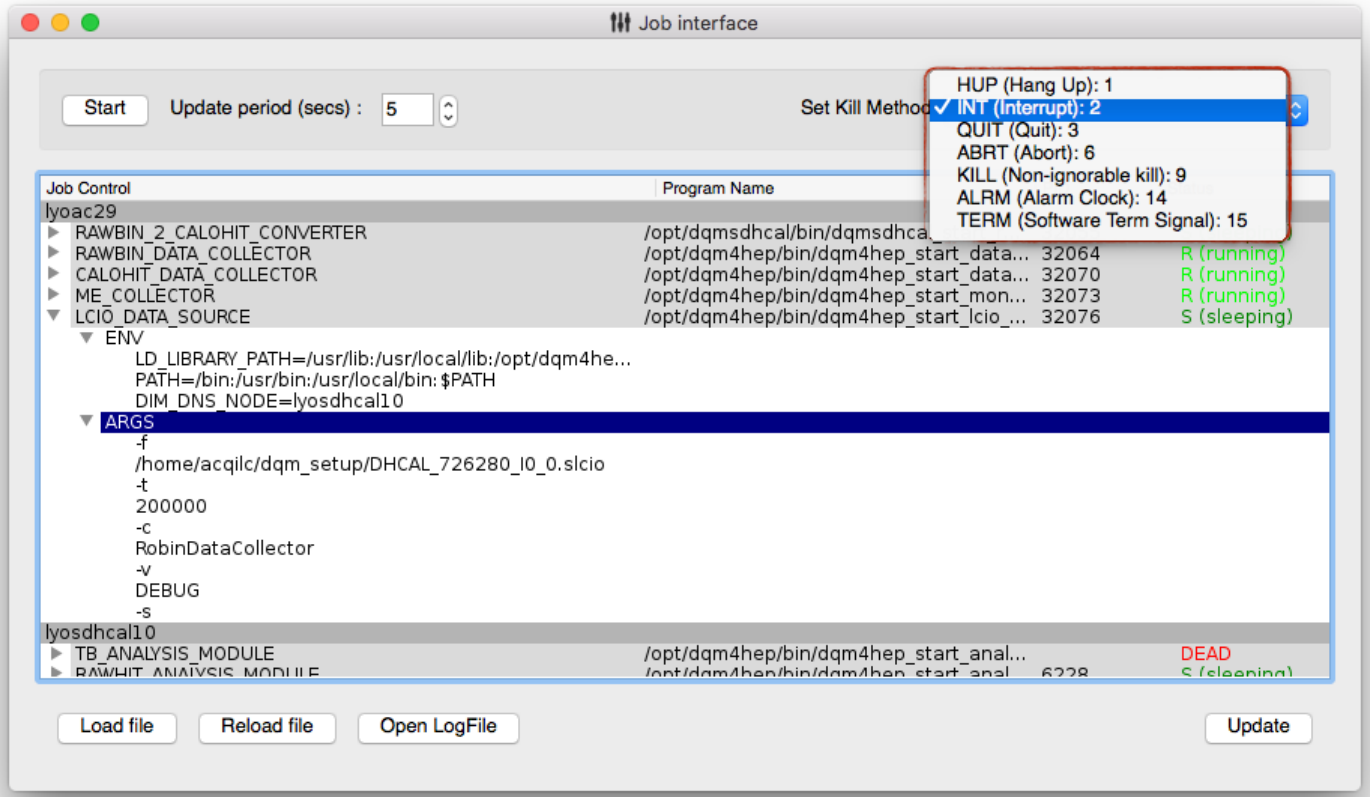
# User interfaces overview

## Job Control GUI



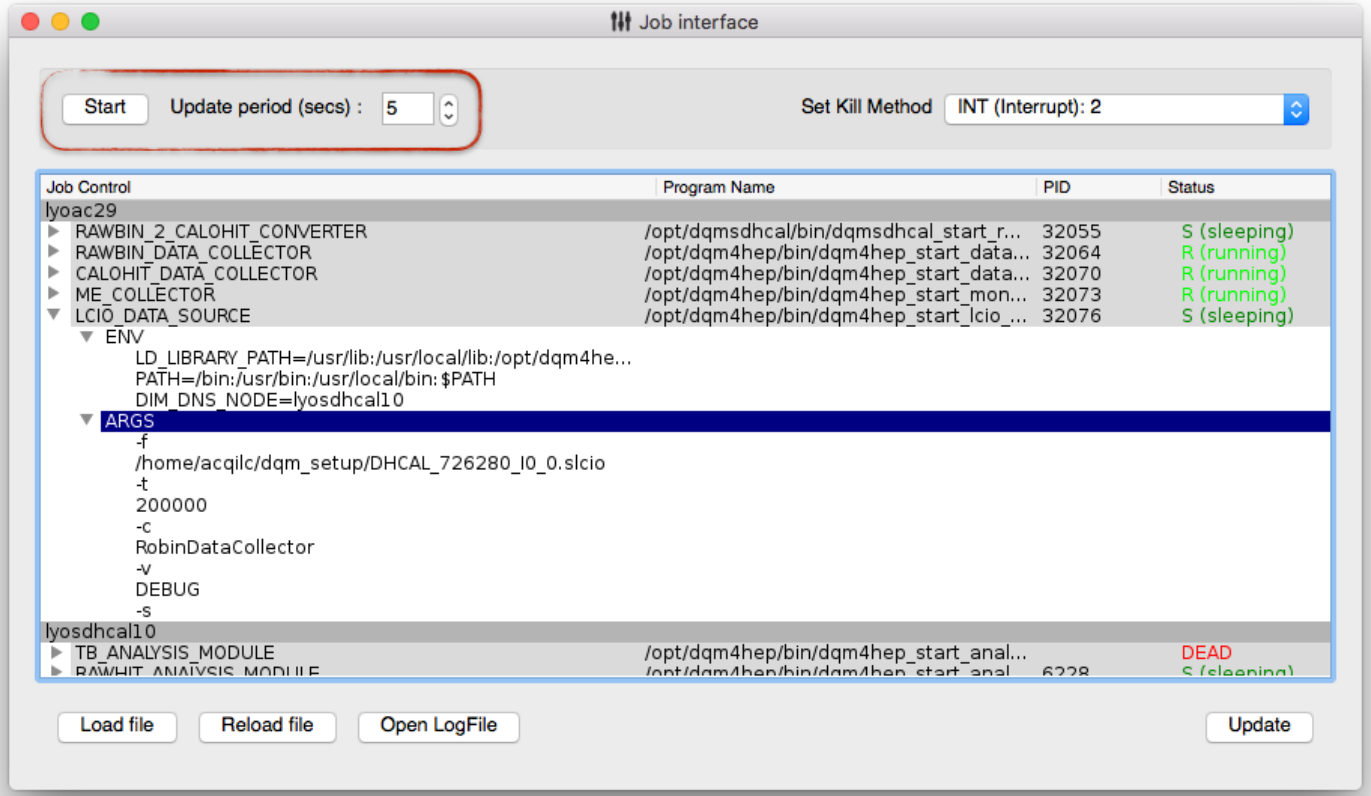
# User interfaces overview

## Job Control GUI



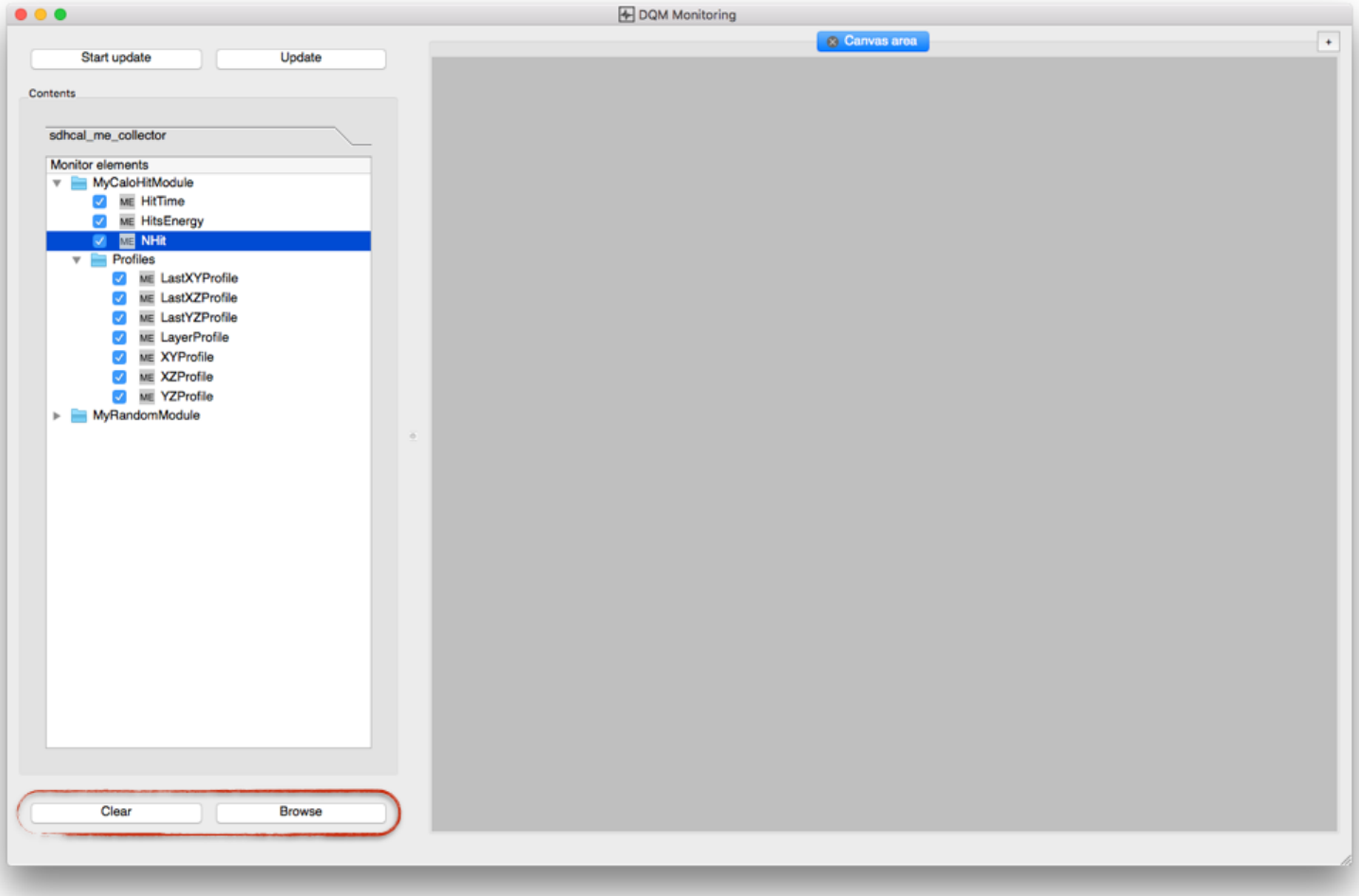
# User interfaces overview

## Job Control GUI



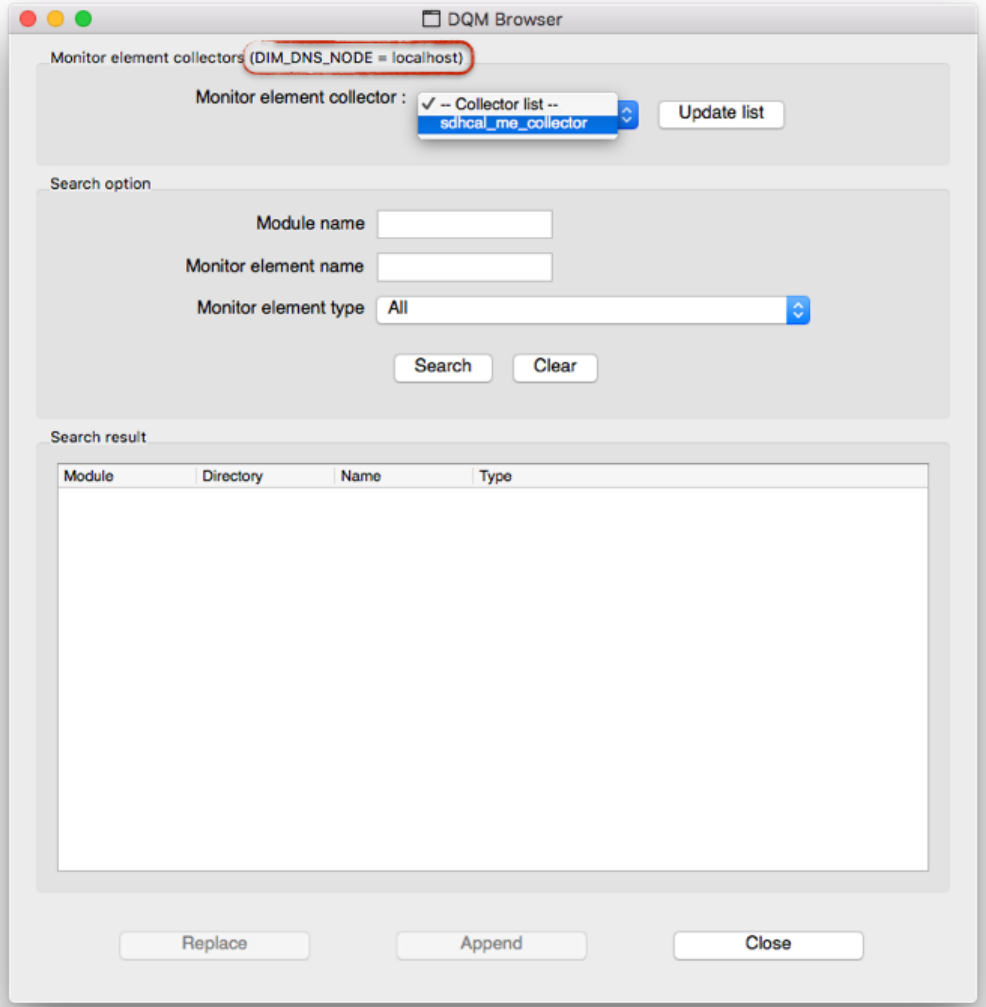
# User interfaces overview

## Monitoring Gui



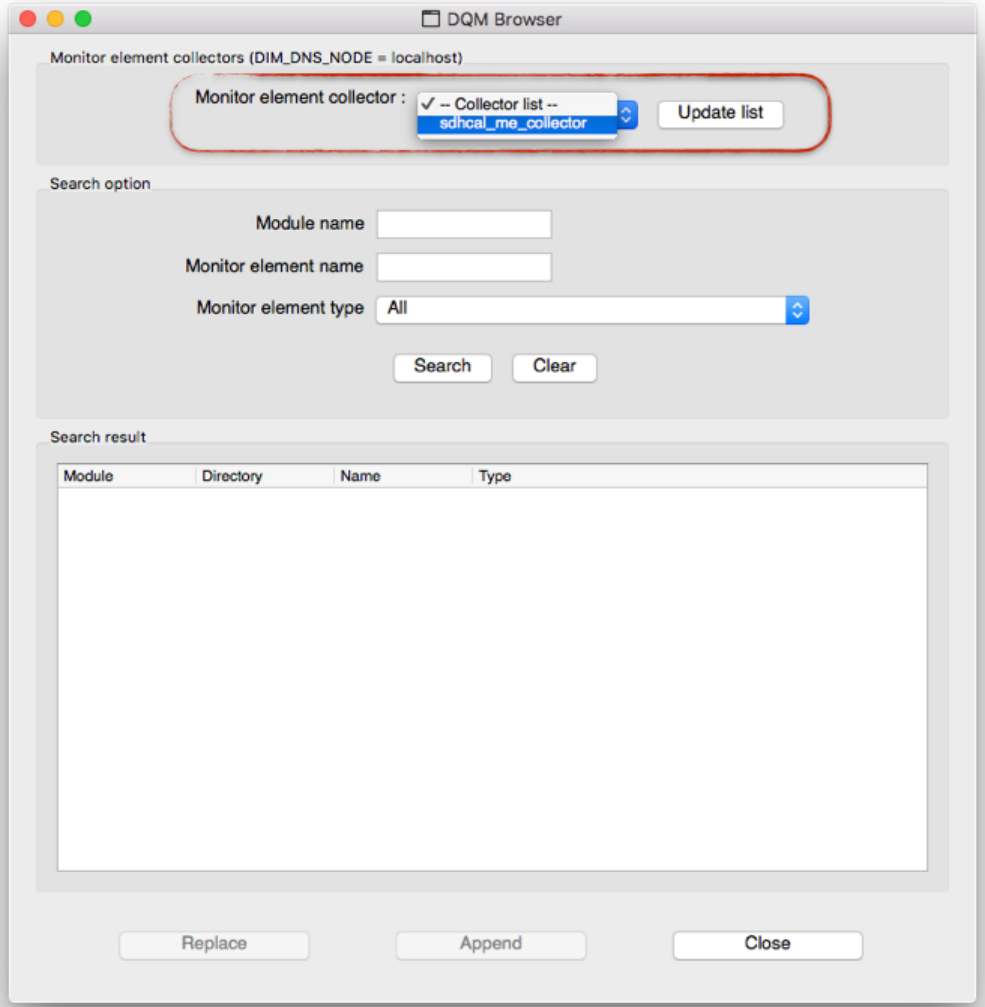
# User interfaces overview

## Monitoring Gui



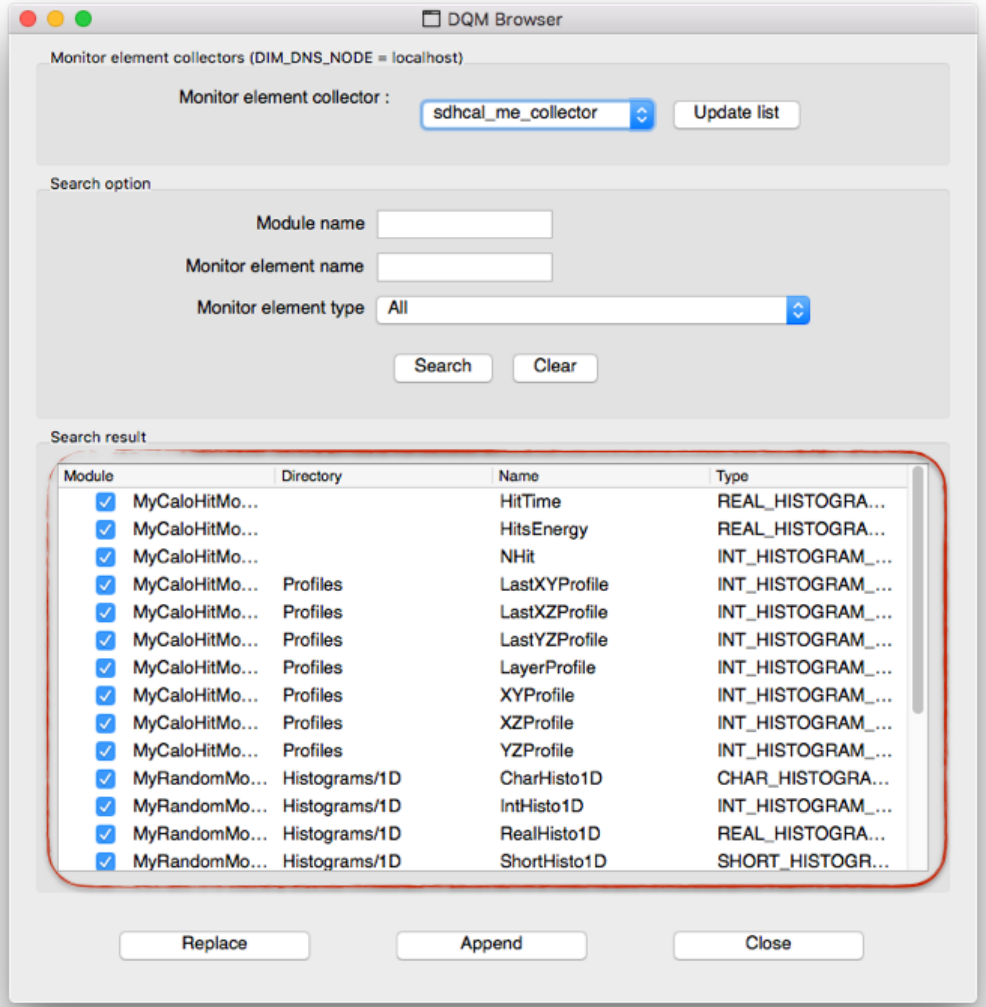
# User interfaces overview

## Monitoring Gui



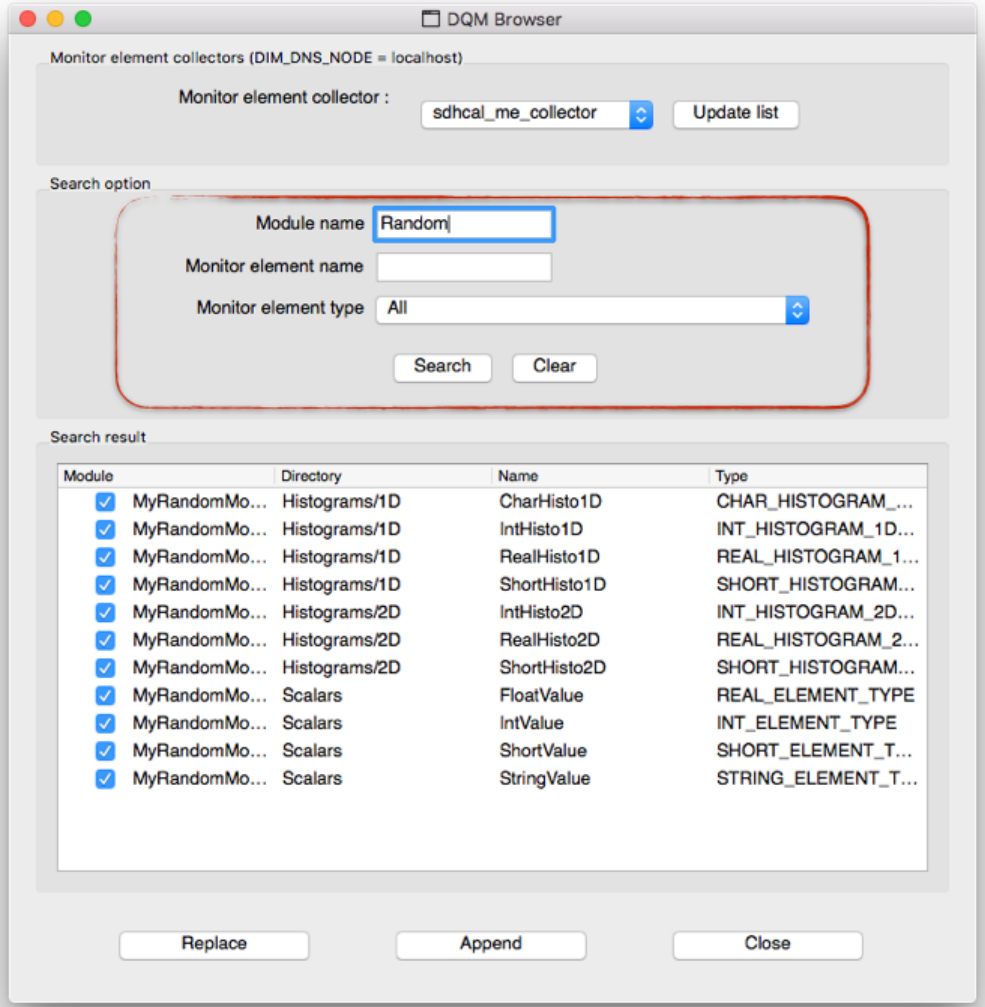
# User interfaces overview

## Monitoring Gui



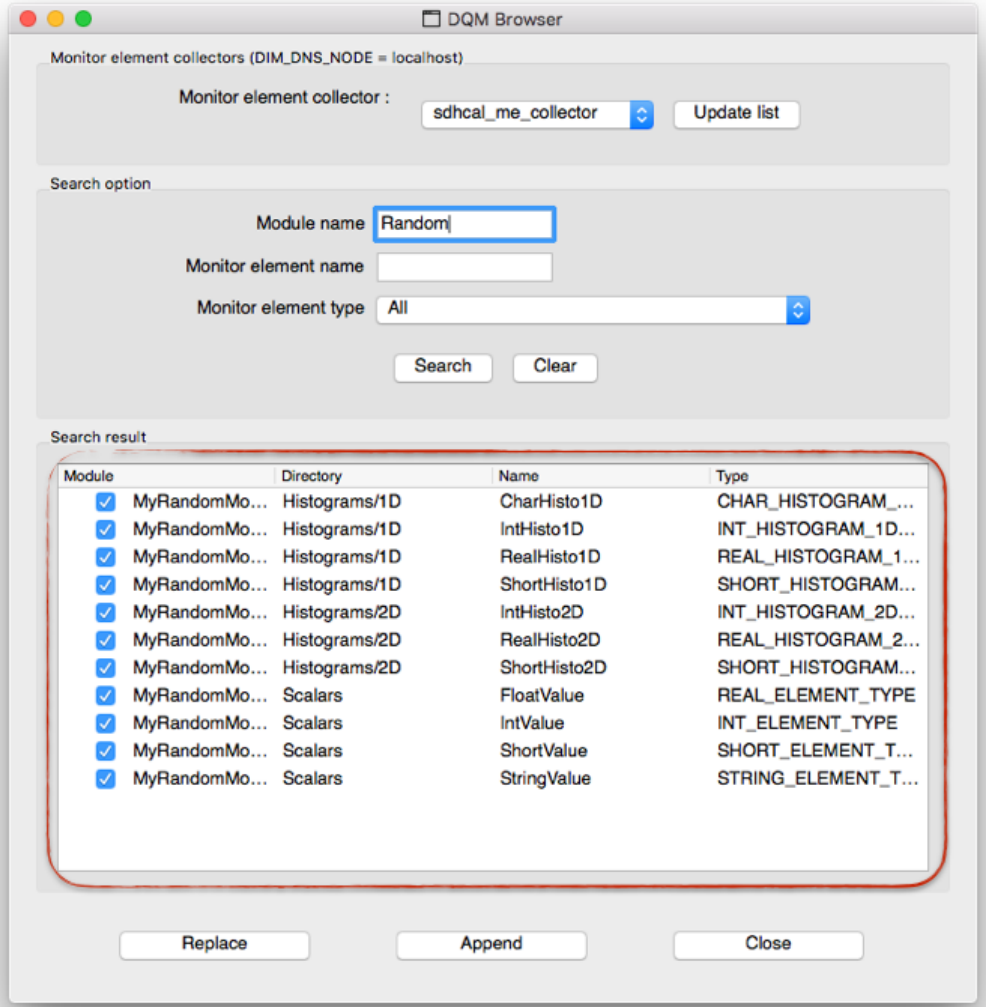
# User interfaces overview

## Monitoring Gui



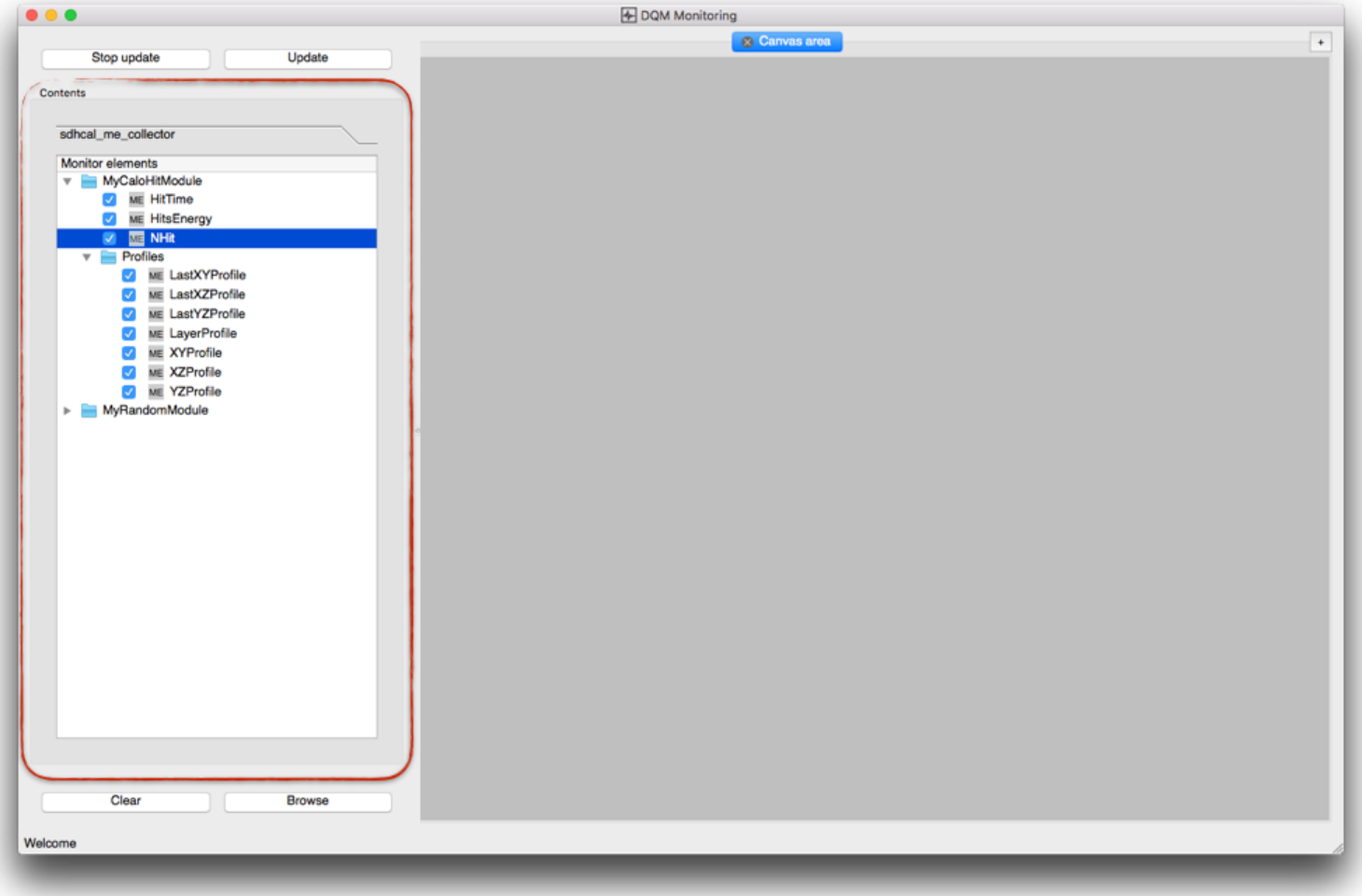
# User interfaces overview

## Monitoring Gui



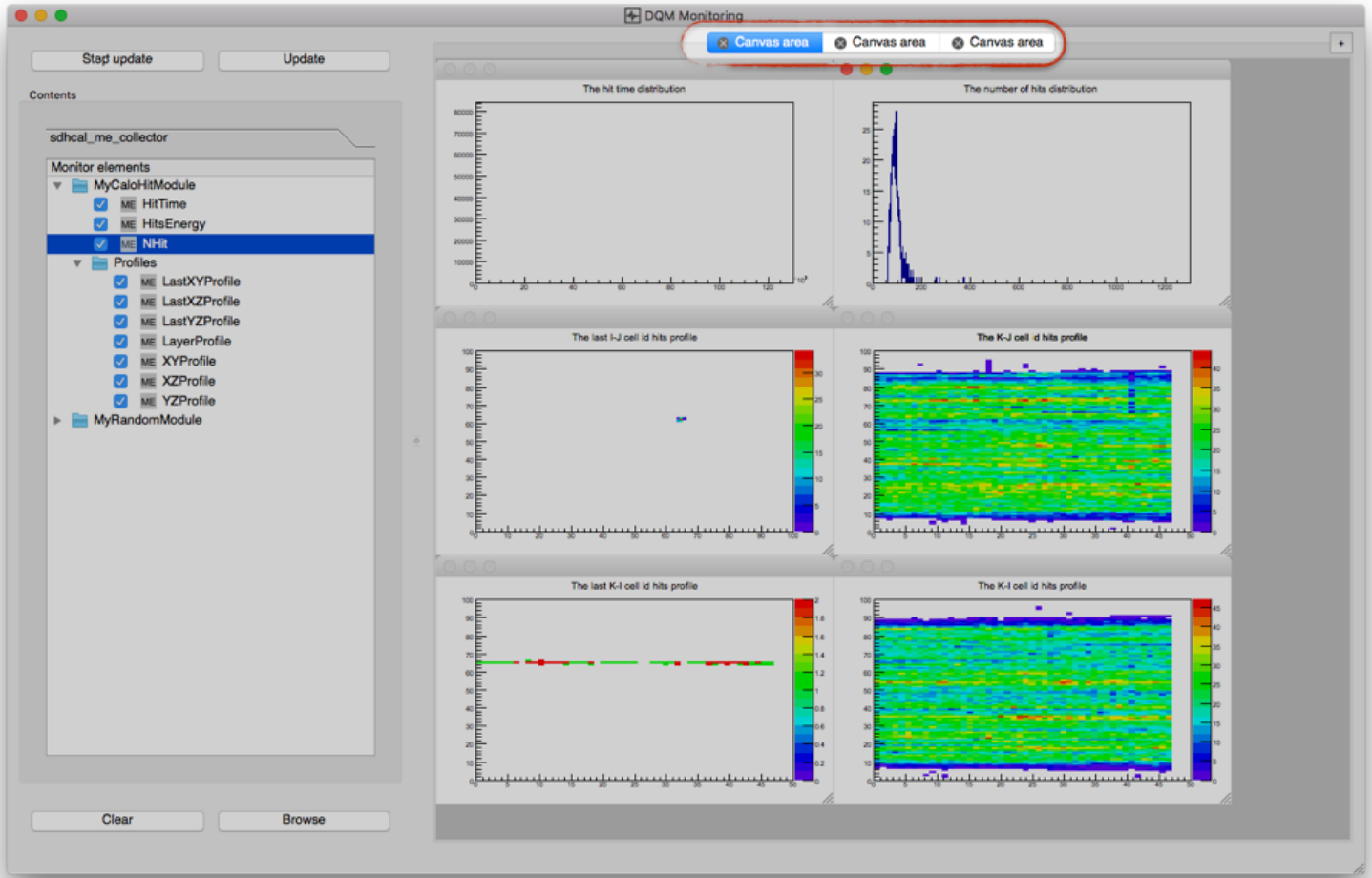
# User interfaces overview

## Monitoring Gui



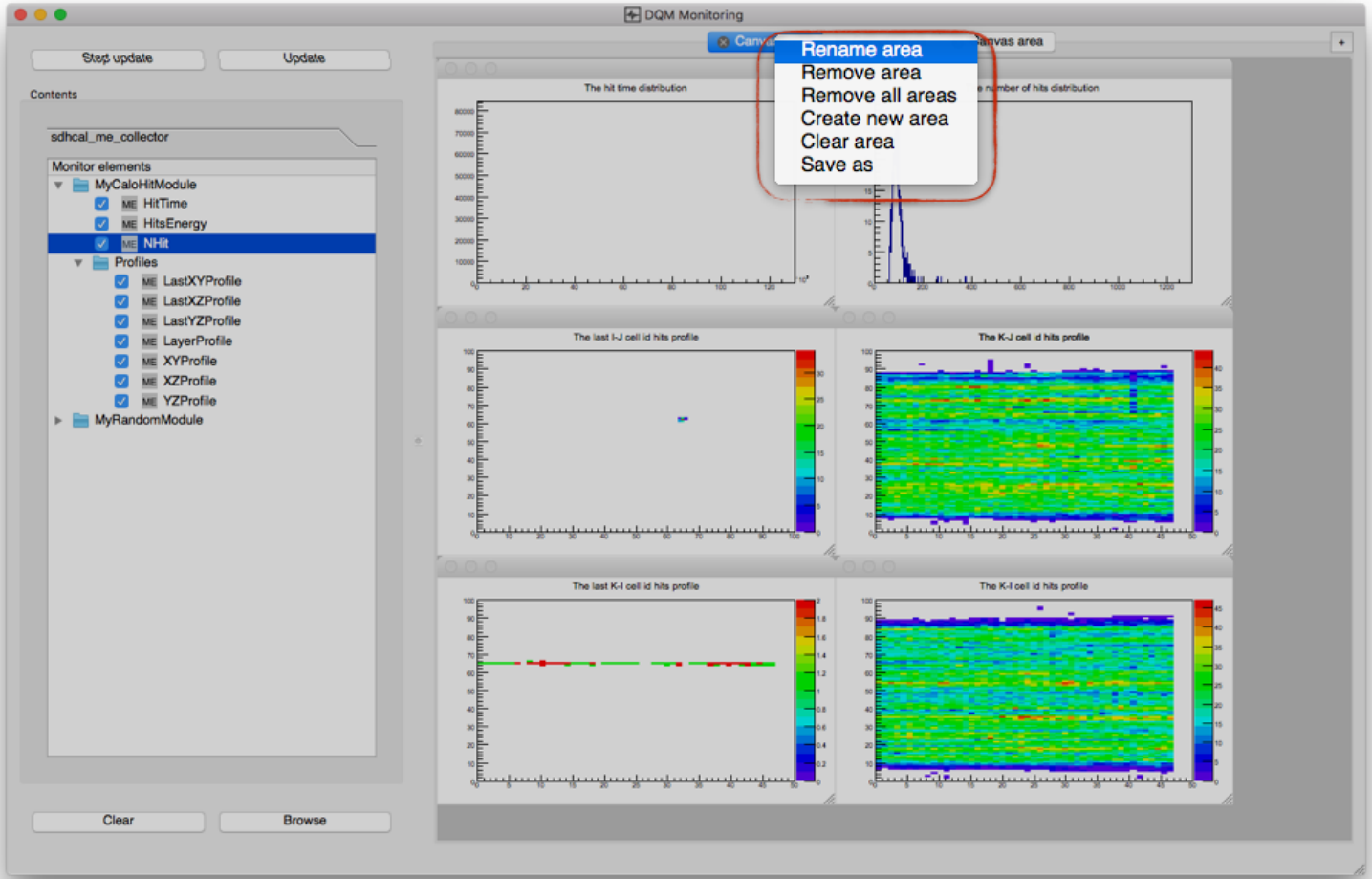
# User interfaces overview

## Monitoring Gui



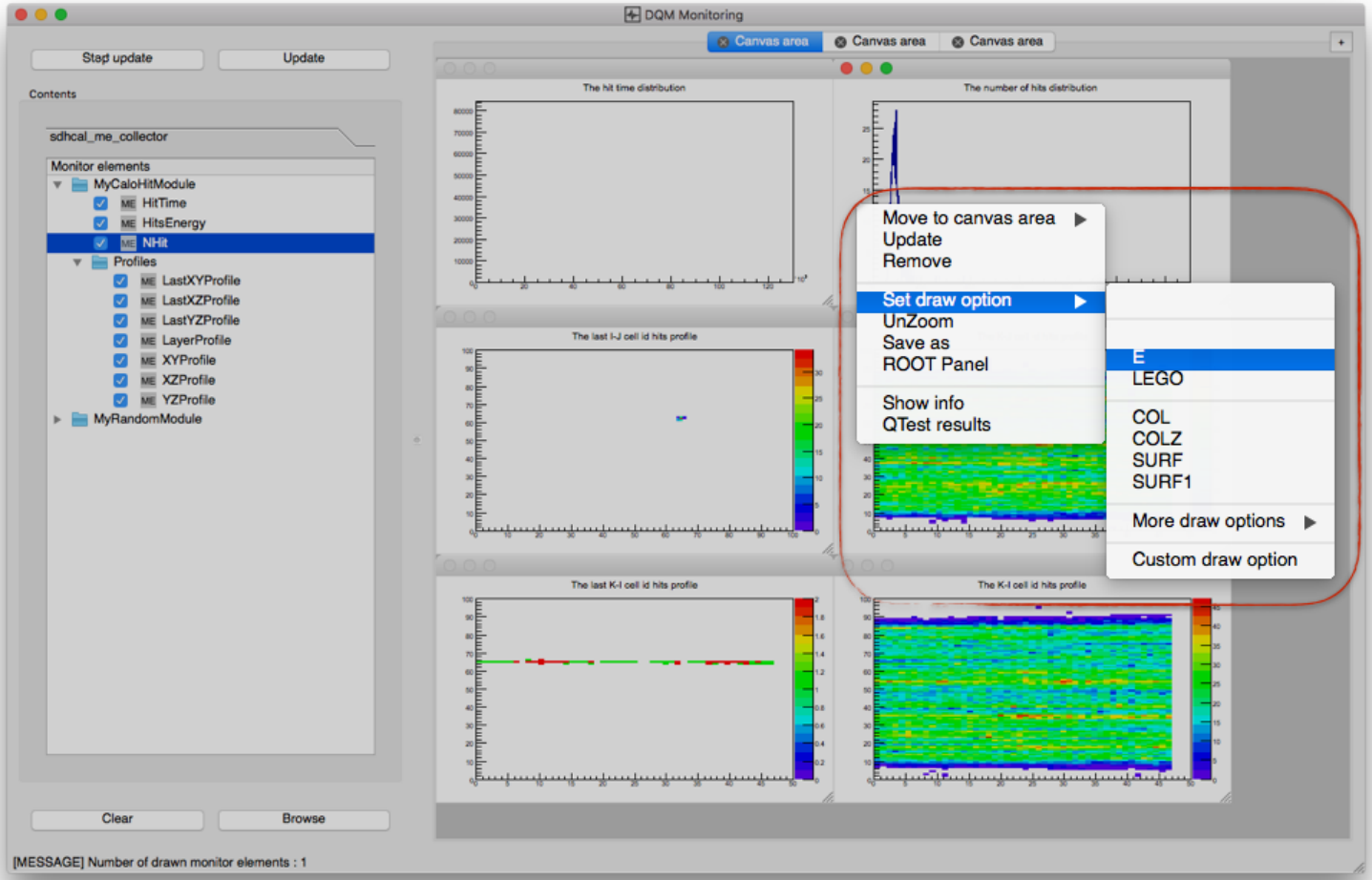
# User interfaces overview

## Monitoring Gui



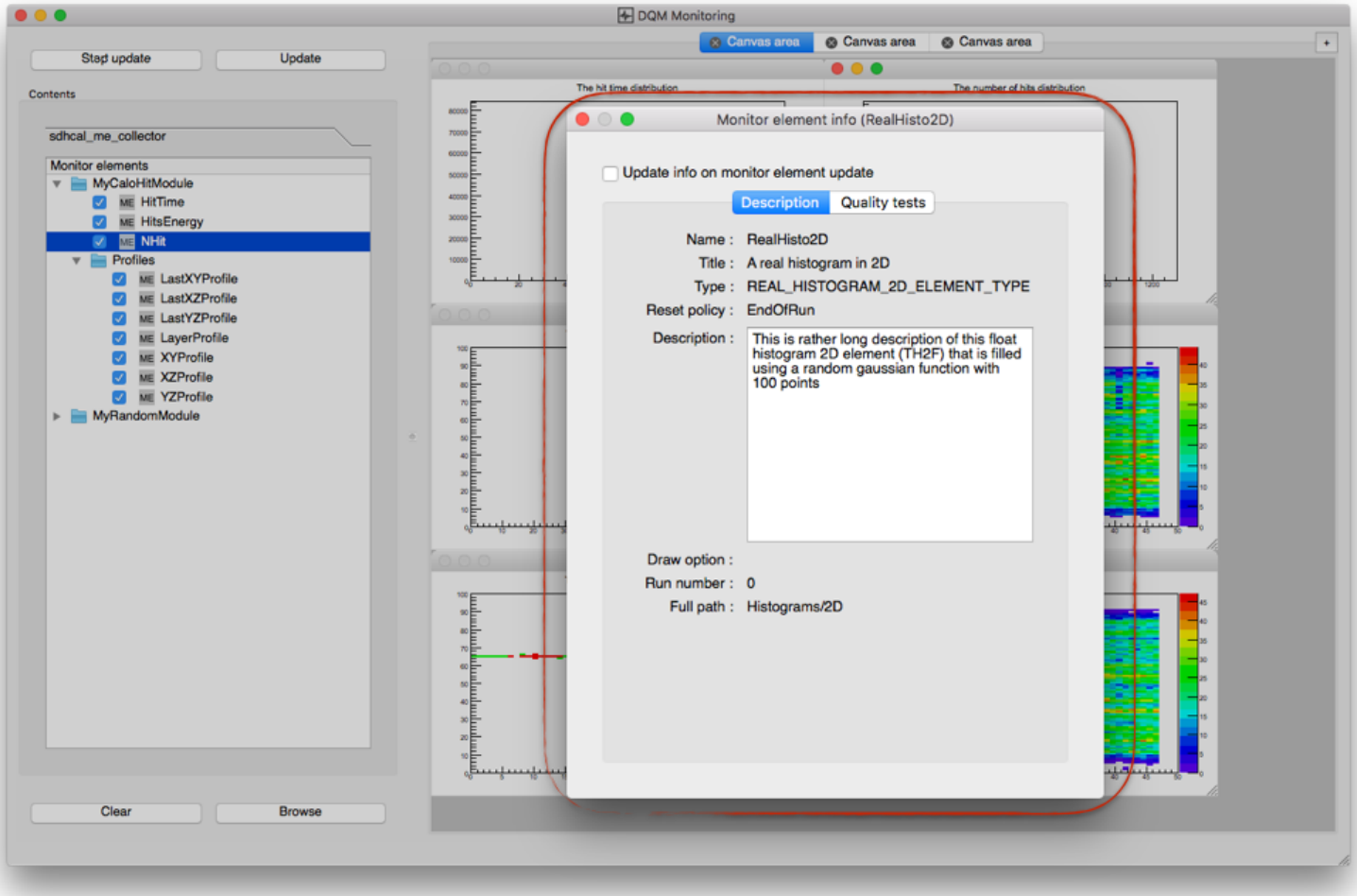
# User interfaces overview

## Monitoring Gui



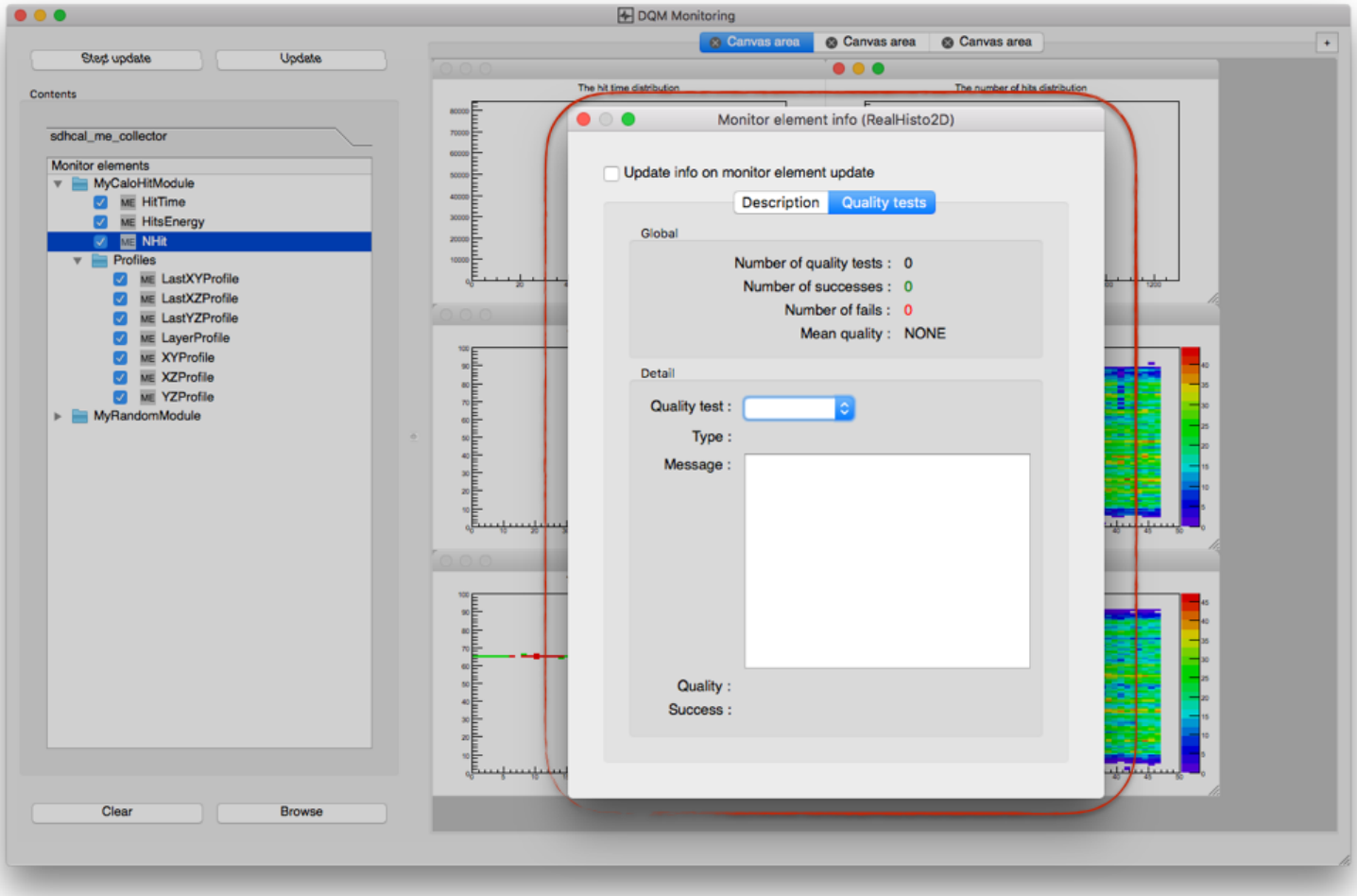
# User interfaces overview

## Monitoring Gui



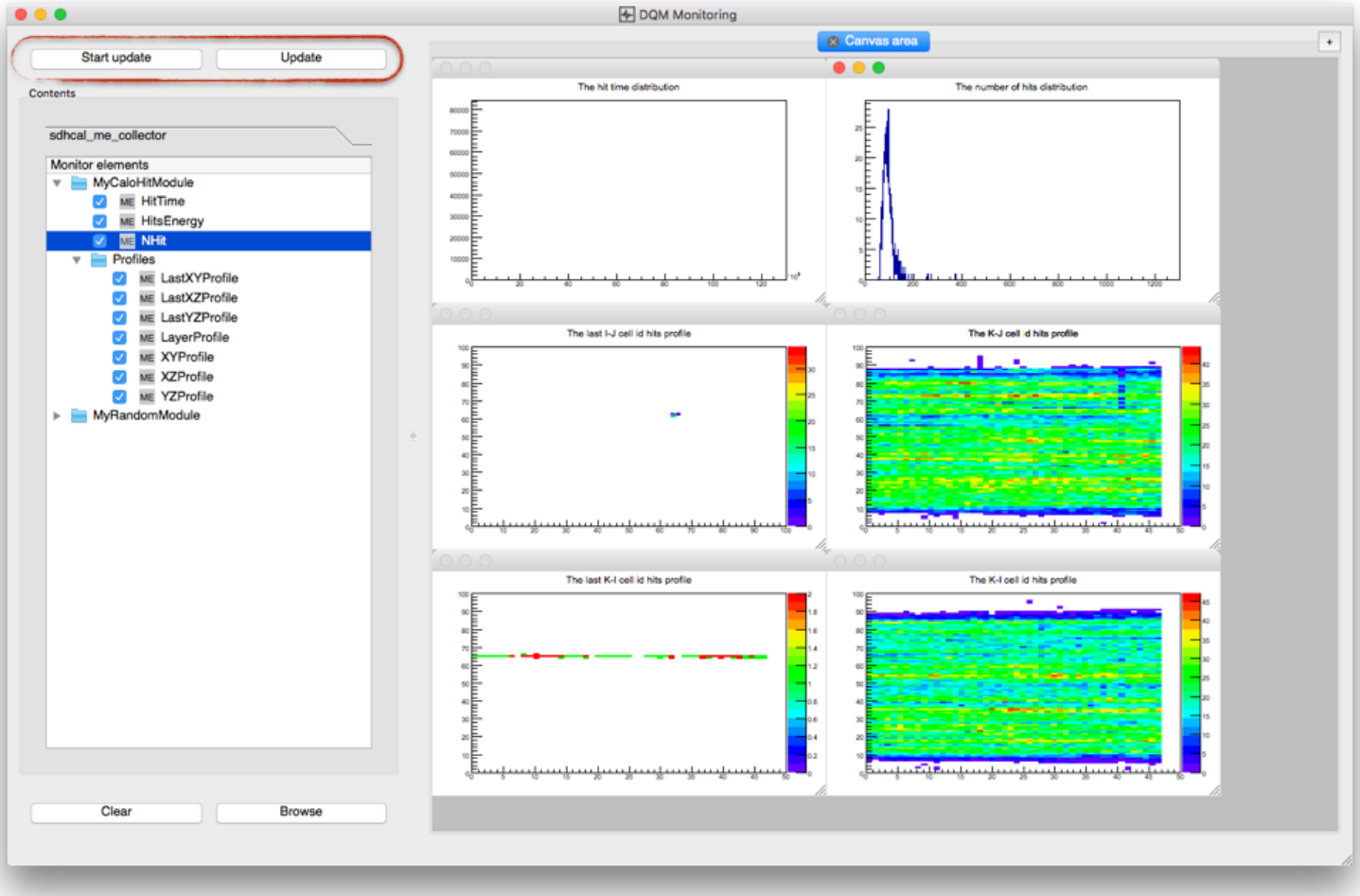
# User interfaces overview

## Monitoring Gui



# User interfaces overview

## Monitoring Gui



# User interfaces overview

## Conclusions and plans

### Conclusion and plans

- Network decouples/links different part of the system.
- Plugins (modules, data streaming) to configure and run the system
- Tools for data feed in the system from the DAQ (event client interface)
- GUIs to control/monitor the system
- Tests are OK but performance tests : timing, statistics, compression (network, streaming)
- ILCSOFT release ?

### Plans for SDHCAL

- Full implementation of SDHCAL DQM :
  - Gas system, HV, LV, T/P
  - Global detector (efficiency, multiplicity, nHit 1-2-3, rate)
  - Particle ID (counting, selection)
  - Particle specific modules (pion, electron, muons)
  - Particle flow (performance)
- Combined ECAL test-beam : combined DQM ?