Web technologies in experiments

Sergey Linev

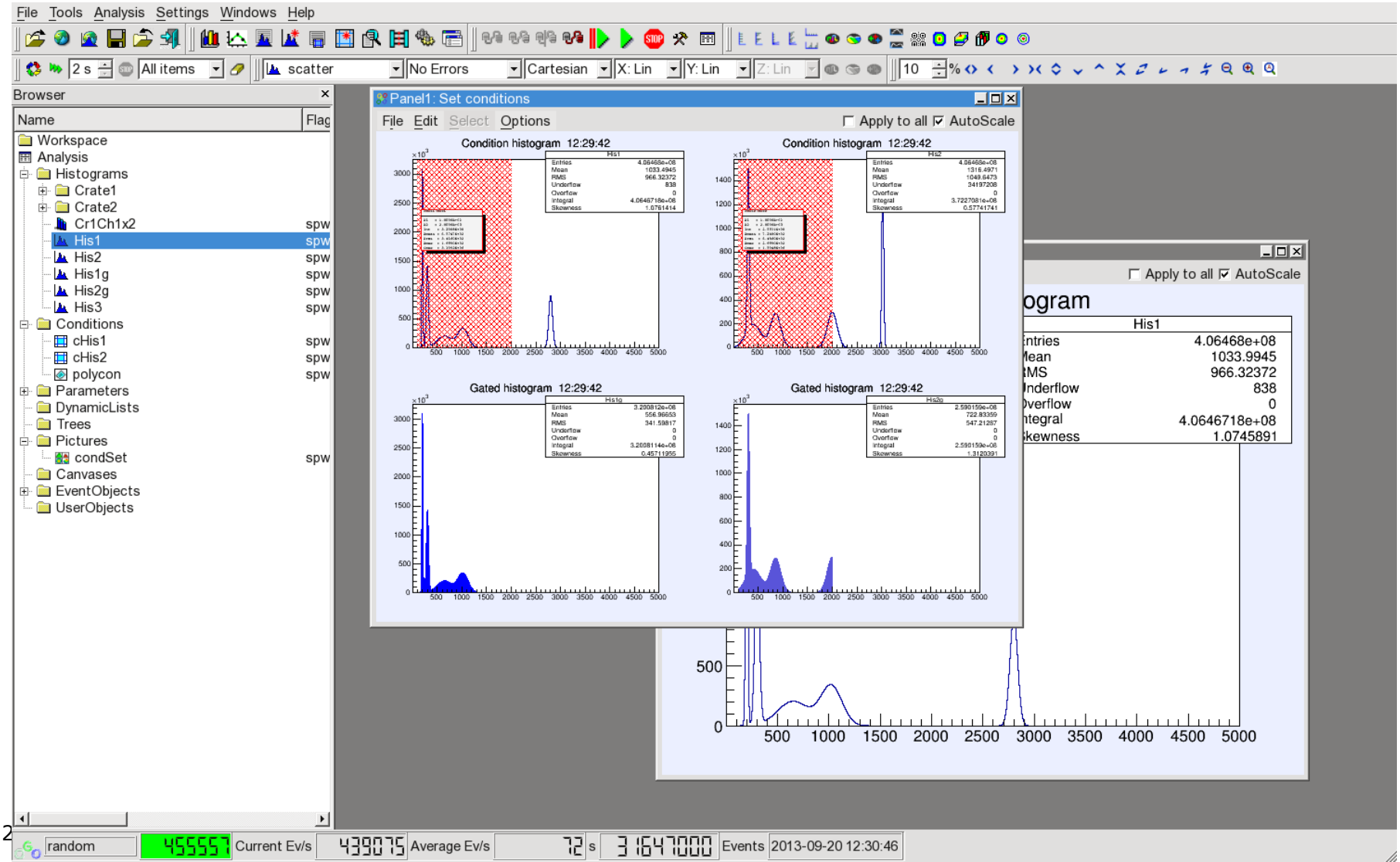
Common systems / Experiment Electronic

24.09.2013

Outlook

- JSRootIO project
- http in DABC
- Several use cases
- How it works
- Plans, ideas, questions

Prehistory – go4 project



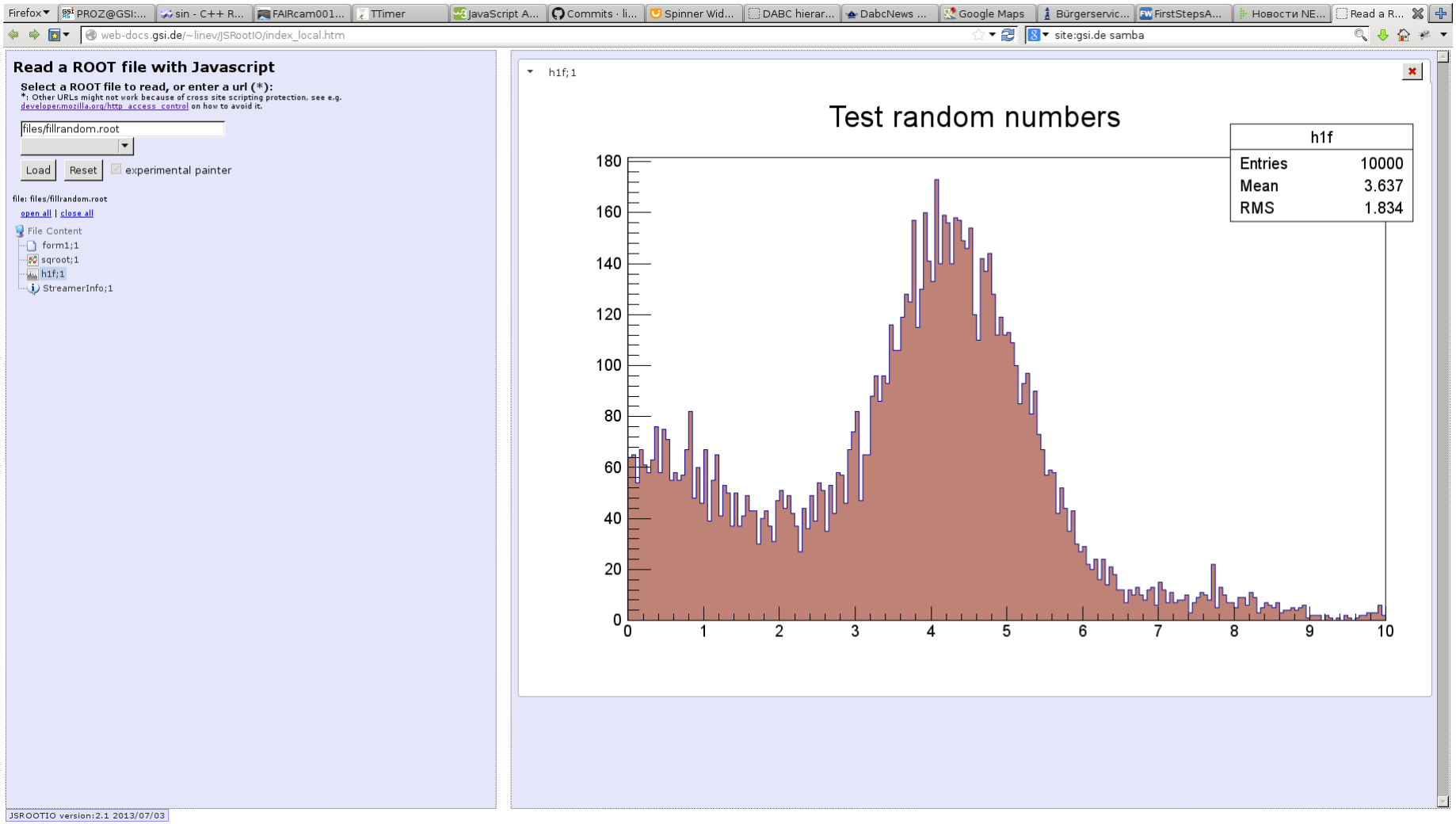
Web in experiments?

- Web provides different protocols to exchange data between application and user:
  - http, ftp, https, …
- There are many ways to represent user data in web browsers:
  - html, css, xml, JavaScript, ...
- Via web browser one potentially could provide user interface from any computing device to:
  - DAQ, control system, online/offline analysis, ...
- Are there ready solutions for that?

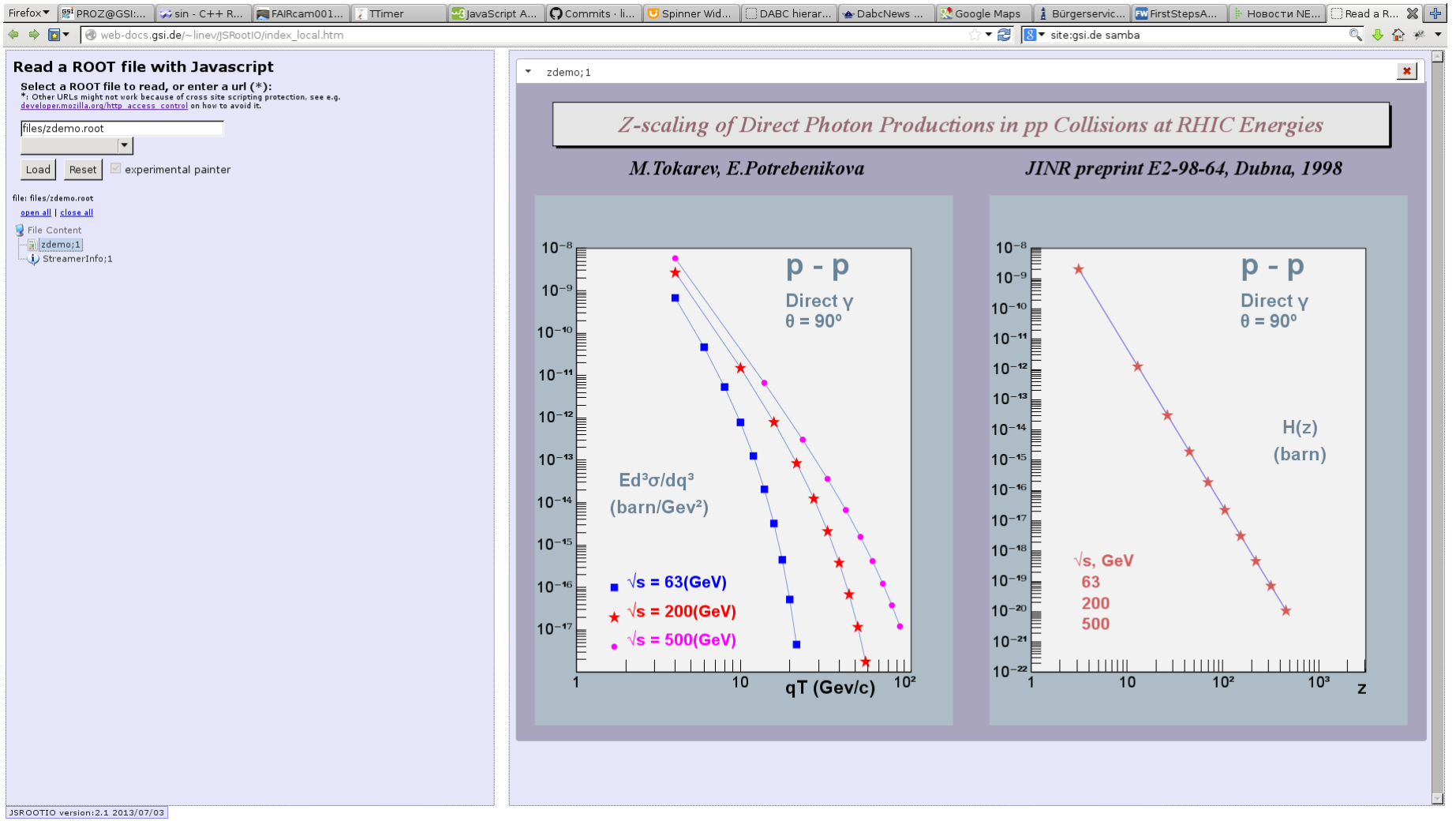
JSRootIO project

- Very new development in ROOT
  - http://root.cern.ch/drupal/content/trevolutionjs
- Written on JavaScript, works in all major browsers
  - IE, Firefox, Opera, Safari, iOS, Android, …
- Allows to draw many kinds of ROOT objects
  - TH1, TH2, TH3, TProfile, TGraph, THStack, TCanvas
- Demonstrator:
  - http://web-docs.gsi.de/~linev/JSRootIO/index_local.htm
- Main aim – display objects from ROOT files
- Advantage compare to GIF/PNG displays
  - manipulation with objects view (change of draw style, update stat box) can be done directly in browser

JSRootIO screenshot



JSRootIO screenshot



JSRootIO for online?

- File-based – pure offline
  - difficult to use for online tasks
  - not really scales for big systems
- Much more benefits if will run online
  - one would see live results from running experiment from any place of the world
- One need special web-server, which provides data online to JSRootIO

http::Server in DABC

- Based on mongoose http-server
  - https://github.com/valenok/mongoose
- Provide web (http) access to:
  - published hierarchical structures in xml form
    - used for navigation in browser
  - published records (plus history) in xml form
    - arbitrary widgets for graphical representation
  - binary buffer with ROOT-streamed objects
    - JSRootIO to decode and display such objects
  - arbitrary binary data, generated in user code

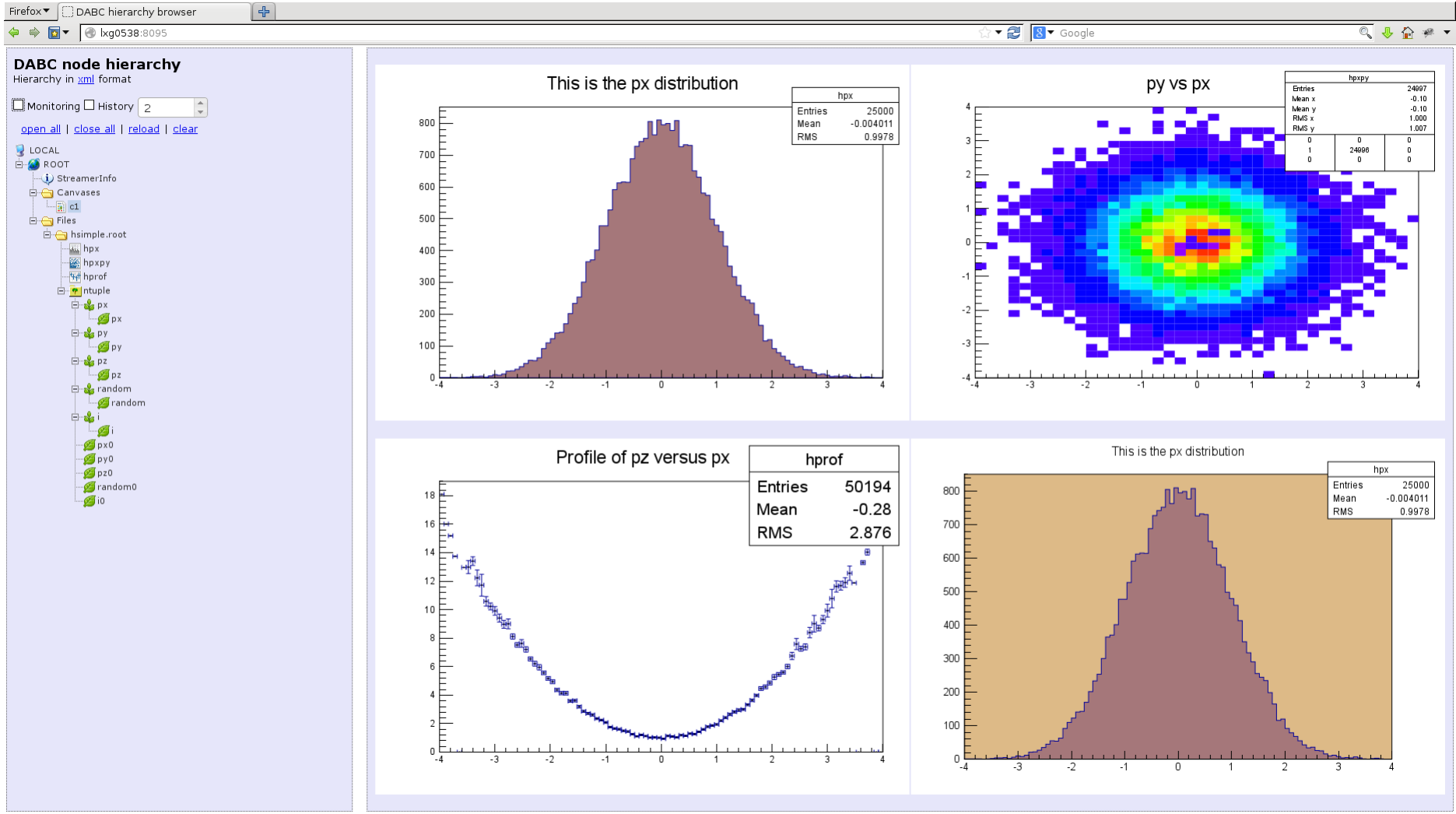
ROOT in DABC

- Main motivation – deliver ROOT objects to browser without intermediate files
- ROOT plugin in DABC streams **arbitrary objects** into binary zipped buffers
- JSRootIO unpack binary data and can display histogram/graph content

Use case – ROOT analysis monitoring

- Monitoring of any ROOT-based analysis
  - DABC 'sniffs' created ROOT objects and represents them in hierarchical view in web browser
  - All files/histograms/canvases can be displayed/monitored
  - **No any** change in analysis code is required
- Just start server at any moment of time
  - `root [0] DabcRoot::StartHttpServer(8095)`
  - `root [1] .x $ROOTSYS/tutorials/hsimple.C`

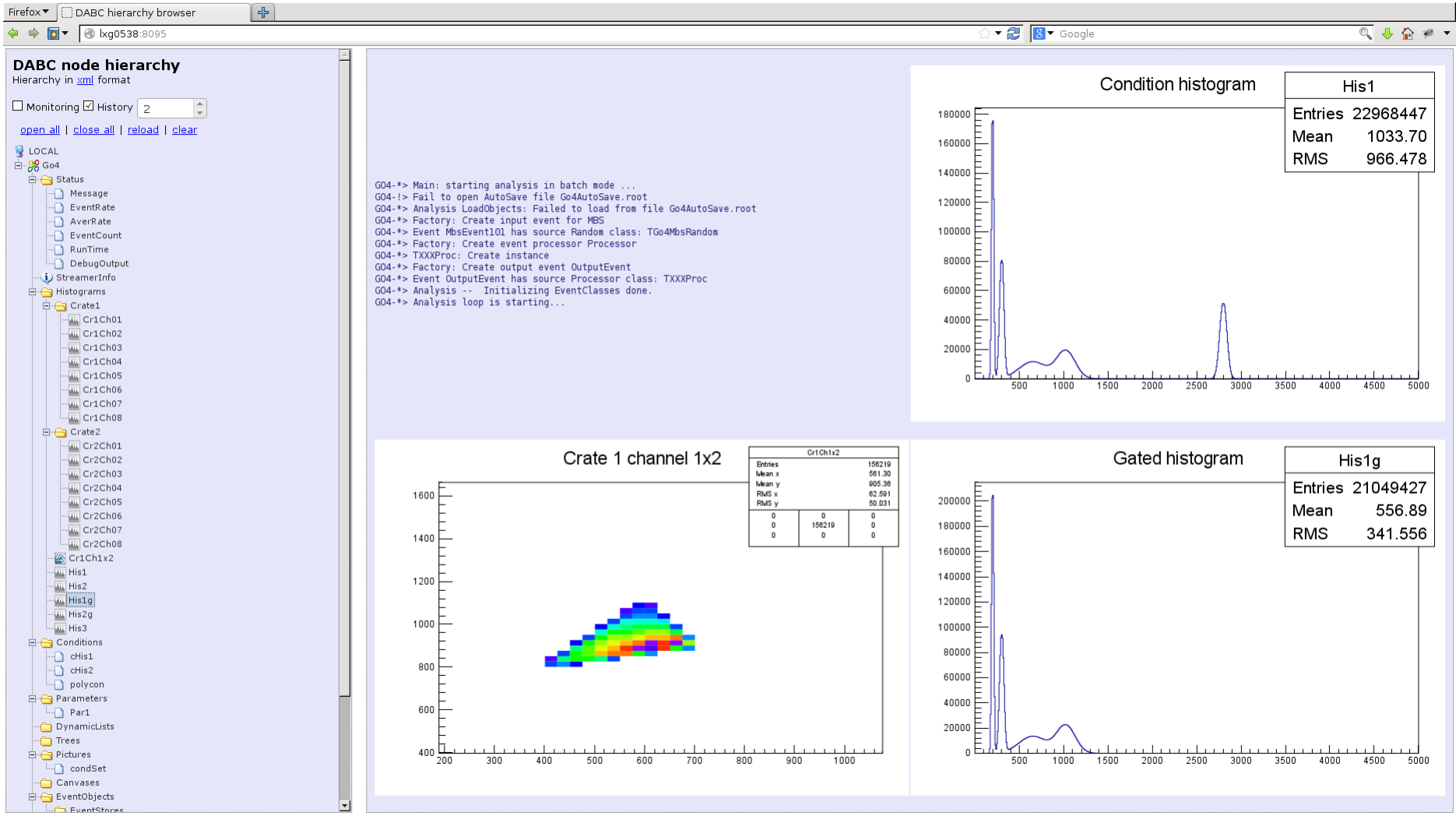
hsimple.C in web browser



Use case – go4analysis monitoring

- Monitoring of any go4-based analysis
  - go4 registers all user objects in folders
  - all these objects are accessible via web server
  - **No any** change in analysis code is required
- Just start analysis with "-http port"
  - `[shell] go4analysis -random -http 8095`
- Available in repository, will be released soon

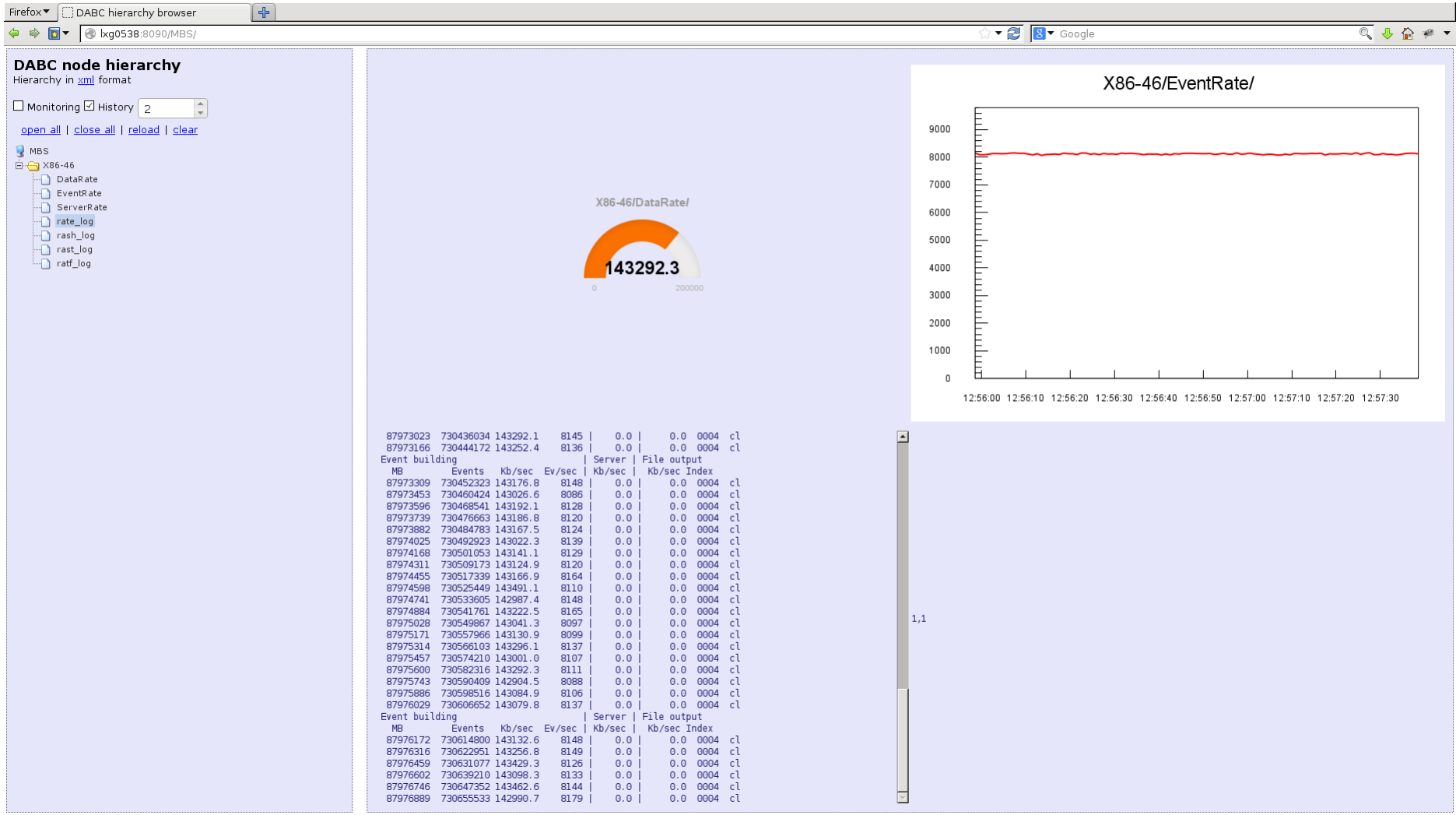
Go4ExampleSimple in web browser



Use case – MBS monitoring

- Plugin for MBS provides log information from MBS status record
- Output like MBS “rate” command
- Plus several ratemeters are delivered separately
- One could monitor any number of MBS nodes simultaneously

Use case – MBS monitoring



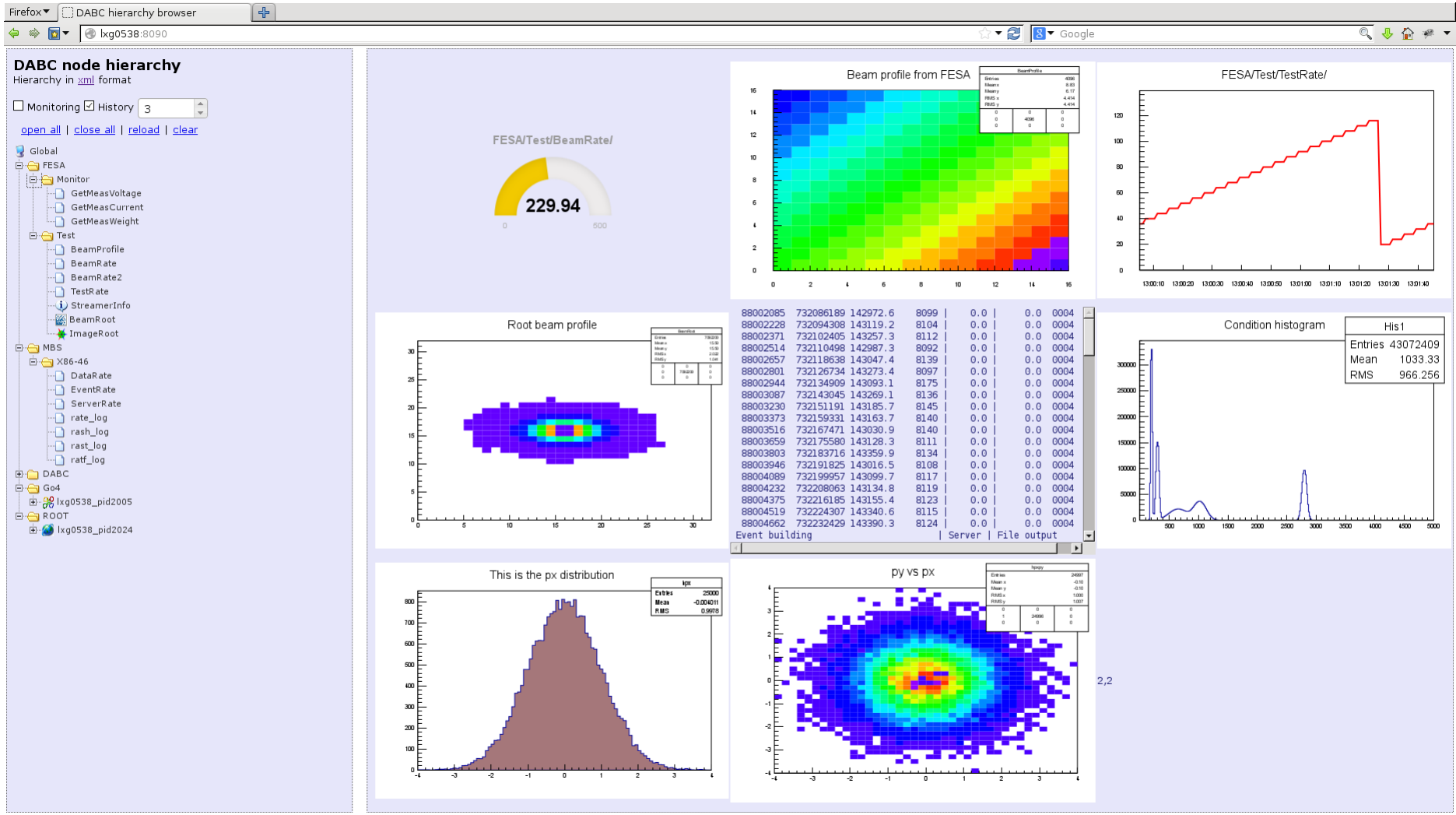
Use case – FESA monitoring

- Special DABC plugin provides access to **selected** FESA services
- Via subscription interface changes in FESA services signaled to the framework
- All available fields of FESA service automatically mapped into DABC hierarchy fields
- At the moment simple text printout used to display values in the browser
- Any custom display can be easily implemented

Use case – monitoring middleware

- Gathering data from very different systems together
  - MBS – GSI DAQ system
  - FESA – accelerator control system
  - ROOT-based analysis
  - EPICS – slow control system
  - DIM – yet another control system
  - any other experiment-specific data
- Deliver gathered data in unified form to web browser

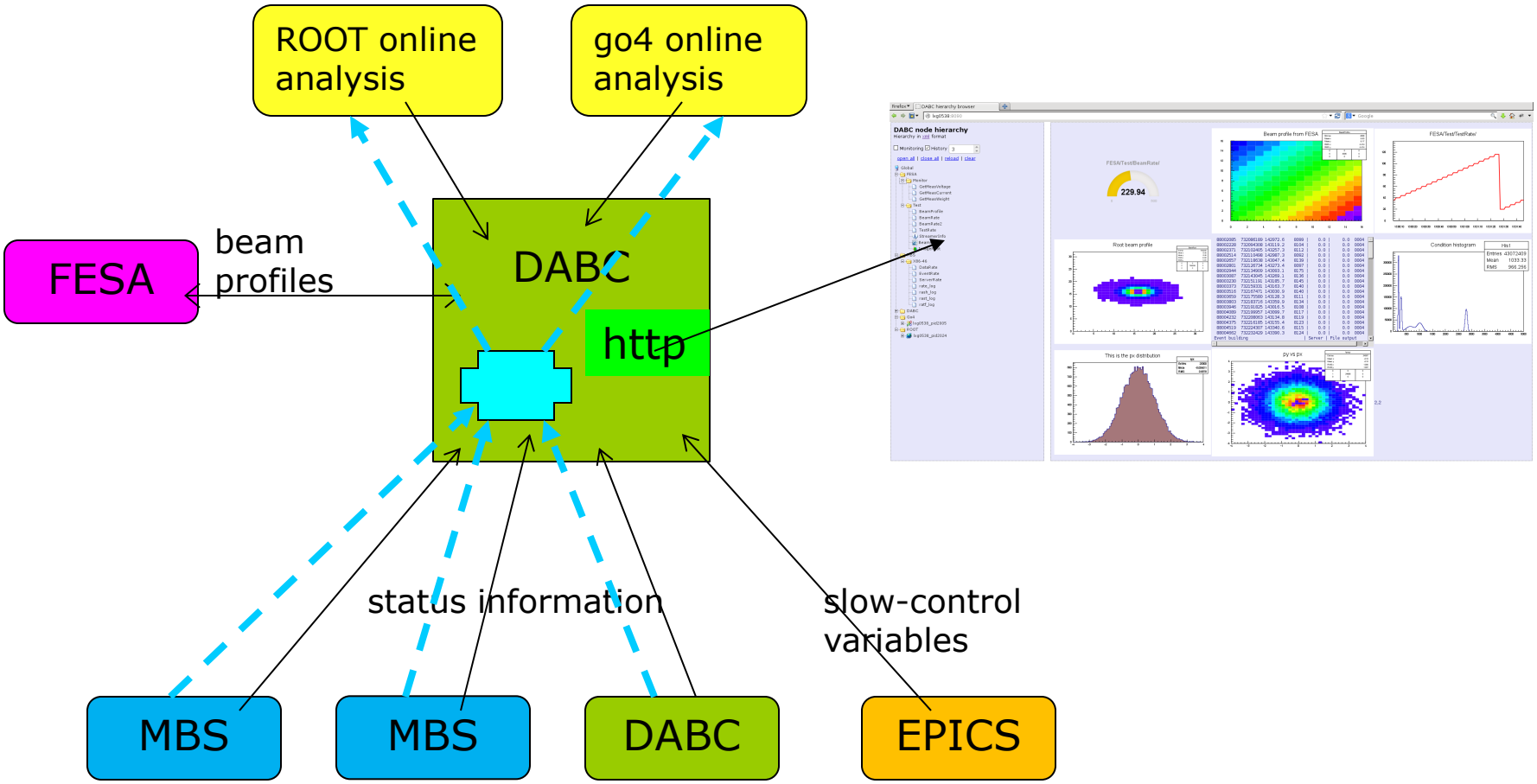
Use case – monitoring middleware



Use case – DAQ and slow-control middleware

- DABC is framework for building DAQ systems
  - many different combinations were used in CBM tests
  - integration of MBS with USB, Ethernet, Optic, ... readout
- DABC can aggregate different data streams, combine them, write to files
- DABC provides same kind of transport/stream servers as MBS
- When necessary, analysis can run in DABC context
- Produced in DABC data can be delivered to web browser, but also to any external framework via socket, temporary files, ...
- Slow-control will remain as is

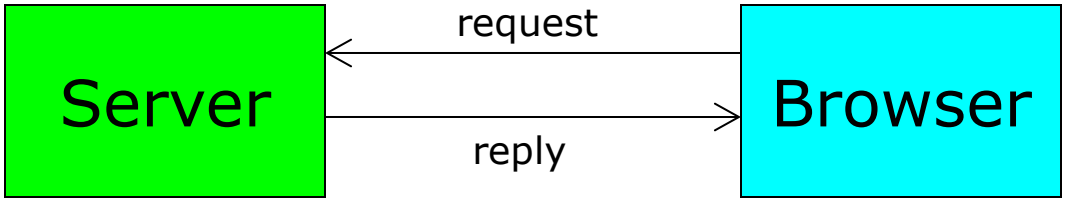
Use case – DAQ and slow-control middleware



Benefits

- Unified interface for different components
  - available on any computer, tablet, smartphone
- No any extra work for ROOT and Go4
  - simple installation routine for dabc
- Can be immediately used for small setups
  - no any complex configurations
- Smooth transition to larger setups
  - minimal effort to run master node
- Integration with other tools
  - EPICS, MIDAS, LabView provide web interfaces

How it works?



- On the server side
  - DABC process with http::Server
- On the browser side
  - JavaScript plus normal HTML

What is DABC

- Multithreaded environment
- Software modules for user code
- Zero-copy data transport approach
  - full support of InfiniBand/10GE VERBS
  - also socket implementation
- Integration with MBS
- Plugins for DIM, EPICS, HADAQ, FESA
- DAQ for CBM test beams (till Nov 2012)

Redesign/improvement of JSRootIO

- Makes it possible to update objects drawing without full clear and redraw
- Performance optimization
- Many new features:
  - context menu
  - comfort zooming – very like to native ROOT
  - stat box creation and update
  - autozoom – shows non-zero content
  - correct treatment of logarithmic scales
  - …

How it works – publish data

- Code is implemented as subclass of dabc::Worker or dabc::Module
- In most practical cases dabc::ModuleAsync is used, where different event-processing routines are defined
- Code runs in assigned thread, several modules can share the same thread
- User creates and publish hierarchical structure of arbitrary complexity:

```
{
  …
  Hierarchy dev1 = top.CreateChild(“Device1”);
  dev1.CreateChild(“Pressure”);
  dev1.CreateChild(“Temperature”);
  Hierarchy dev2 = top.CreateChild(“Device2”);
  dev2.CreateChild(“Current”);
  dev2.CreateChild(“Voltage”);

  Publish(“/Detector/Devices”, top);
  …
}
```

- **Detector**
  - **Devices**
    - **Device1**
      - **Pressure**
      - **Temperature**
    - **Device2**
      - **Current**
      - **Voltage**

How it works – modify data

- In simplest case values are read periodically from device and set to appropriate elements

```
void UserModule::ProcessTimerEvent(unsigned)
{
   double press(1e5), temp(22.5), curr(0.1), volt(3.3);

   // by any means read values from devices

   top.FindChild(“Device1/Pressure”).Field(“value”).SetDouble(press);
   top.FindChild(“Device1/Temperature”).Field(“value”).SetDouble(temp);
   top.FindChild(“Device2/Current”).Field(“value”).SetDouble(curr);
   top.FindChild(“Device2/Voltage”).Field(“value”).SetDouble(volt);

   top.MarkChangedItems();
}
```

How it works – version control

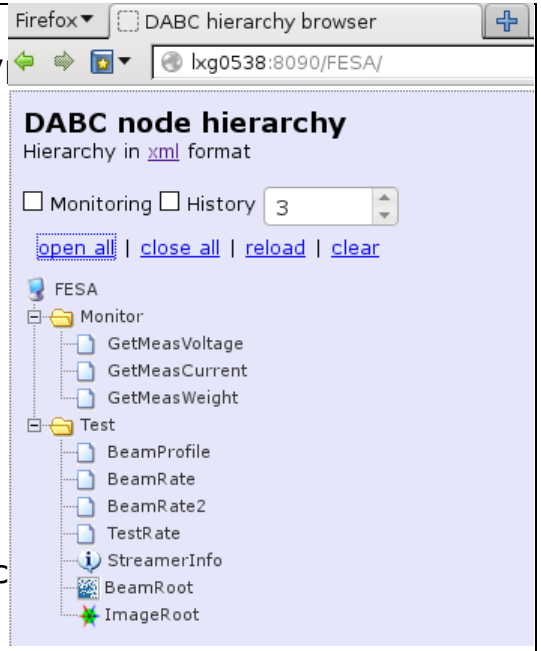
- Each entry in hierarchy has 64-bit version number
- When any field in entry is changed, new version number is set
- Version number is always provided when entry or hierarchy is requested

- Using version number one could:
  - identify if element was changed since last request
  - get history relative to previous request
  - optimize storage of data

How it works – hierarchy request

wget http://lxg0538:8090/FESA/h.xml

```
<?xml version="1.0"?>
<dabc version="2" xmlns:dabc="http://dabc.gsi.de/xhtml" path="/
<FESA>
  <Monitor dabc:producer="dabc://lxg0538:1237/fesa_monitor">
    <GetMeasVoltage dabc:history="100"/>
    <GetMeasCurrent dabc:history="100"/>
    <GetMeasWeight dabc:history="100"/>
  </Monitor>
  <Test dabc:producer="dabc://lxg0538:1237/fesa">
    <BeamProfile dabc:hash="74647" dabc:kind="FESA.2D"/>
    <BeamRate dabc:kind="rate"/>
    <BeamRate2 dabc:history="100" dabc:kind="rate"/>
    <TestRate dabc:history="100" dabc:kind="rate"/>
    <StreamerInfo dabc:hash="32" dabc:kind="ROOT.TList"/>
    <BeamRoot dabc:hash="7464700" dabc:kind="ROOT.TH2I" dabc
    <ImageRoot dabc:kind="image.png"/>
  </Test>
</FESA>
</dabc>
```



How it works – item request

wget http://lxg0538:8090/FESA/Test/TestRate/get.xml

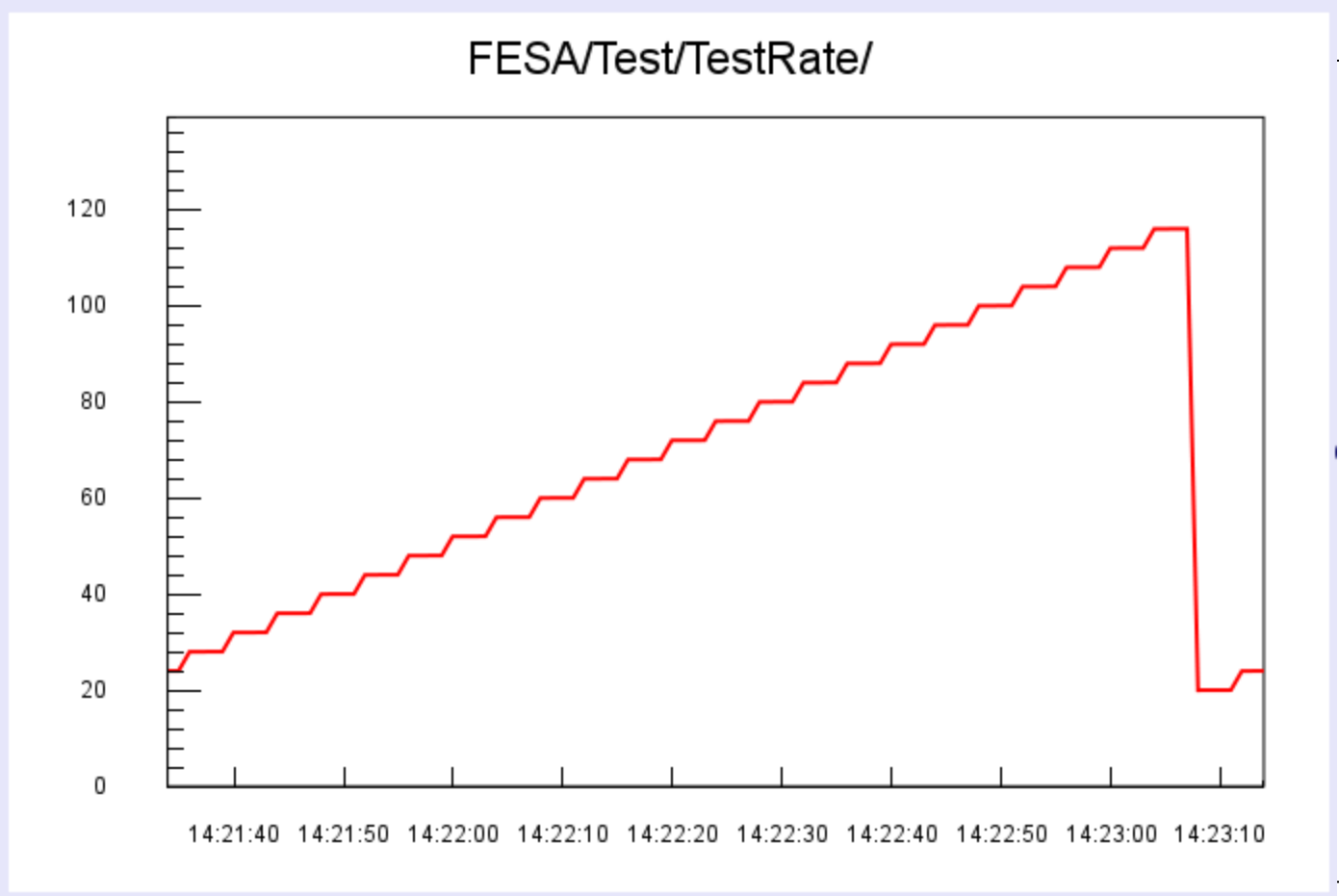
server item request

```
<Reply xmlns:dabc="http://dabc.gsi.de/xhtml" itemname="/FESA/Test/TestRate" dabc:version="75177">
  <get dabc:history="100" dabc:kind="rate" time="2013-09-20T12:12:43.874Z" value="96.00"/>
</Reply>
```

How it works – history request

wget http://lxg0538:8090/FESA/Test/TestRate/get.xml?history=20

```
<Reply xmln
<get dabc:hi
  <history ga
    <h time='
    <h time='
    <h time='
    <h time='
    <h time='
    <h time='
    <h time='
    <h time='
    <h time='
    <h time='
    <h time='
    <h time='
    <h time='
    <h time='
    <h time='
    <h time='
    <h time='
    <h time='
    <h time='
  </history>
</get>
</Reply>
```



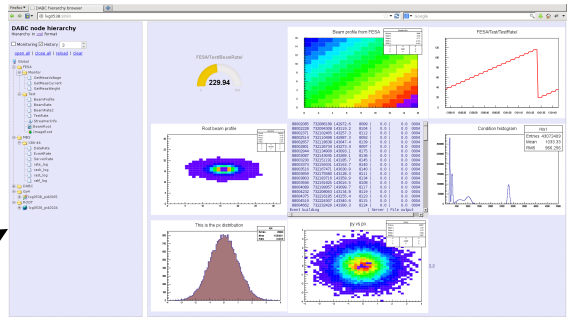
Current state

- Main concepts are proved and working
  - web server integrated in DABC
  - hierarchical structure organization
  - preliminary API is defined
  - distributed systems, gathering data together
  - online version of JSRootIO
  - primary plugins for MBS, FESA, EPICS, DIM
  - any Go4 and ROOT analysis can be monitored

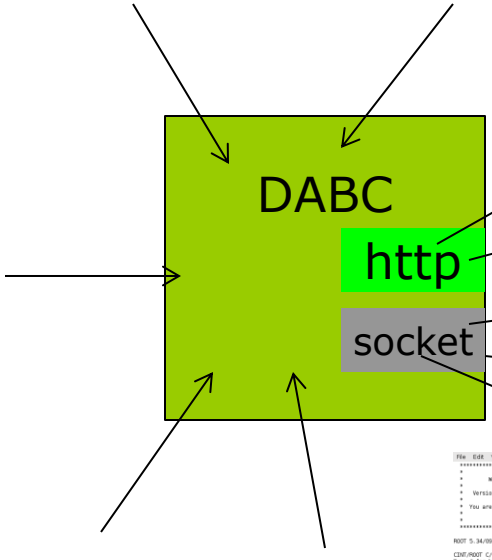
Much more to be done

- Recording of gathered data:
  - writing collected data in binary files
  - access to recorded data from browser
  - access to data from analysis
  - store data in SQL databases
- Access rights (users/passwords) management
- SOAP and WSDL support, to which extend?
- Alternative (non-http) browsers?

Alternatives to web browser

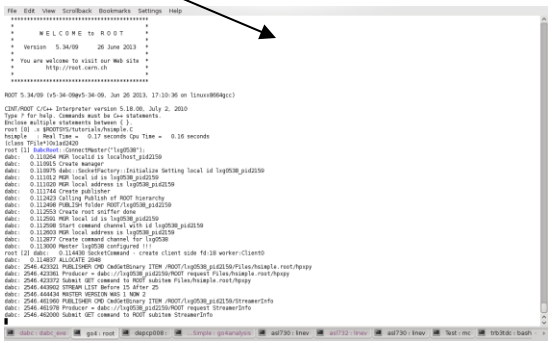


web browser

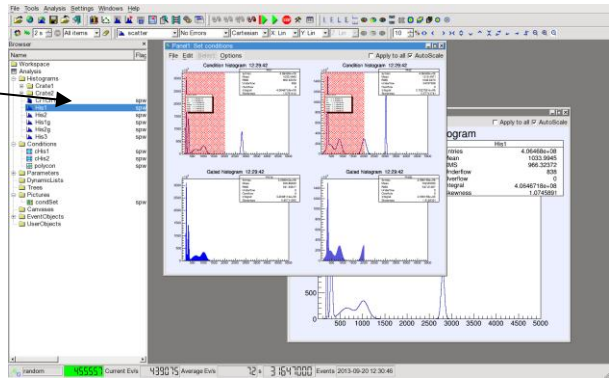


SOAP

DABC API



console-based tools



go4 gui with
native ROOT graphics

Questions to be discuss

- Data organization
  - is folders hierarchy proper model?
  - supported data types
    - int, double, boolean, date, string
    - 1-D arrays
    - something else?
  - date/time format
    - now JavaScript format is used (ms since 1970)
  - API functionality
    - get is implemented
    - get with N previous elements (history)
    - do we need asynchronous mode?

Questions to be discuss

- Control interface
  - do we need http-based control interface?
    - send commands
    - change parameters
  - do it with SOAP?
  - how flexible it should be?

Questions to be discuss

- Experiment-specific components
  - hardware/software kinds
  - required plugins
  - possible experimental setups
  - custom (binary) data support
  - integration with other web tools
- Custom web-browser elements
  - is generic GUIs good enough?
  - support of embedded elements?
  - user-specific elements

Conclusion

- Development at the start point
  - was started ~3 months ago
- Many things working already now
  - much more need to be implemented
- Feedback is required