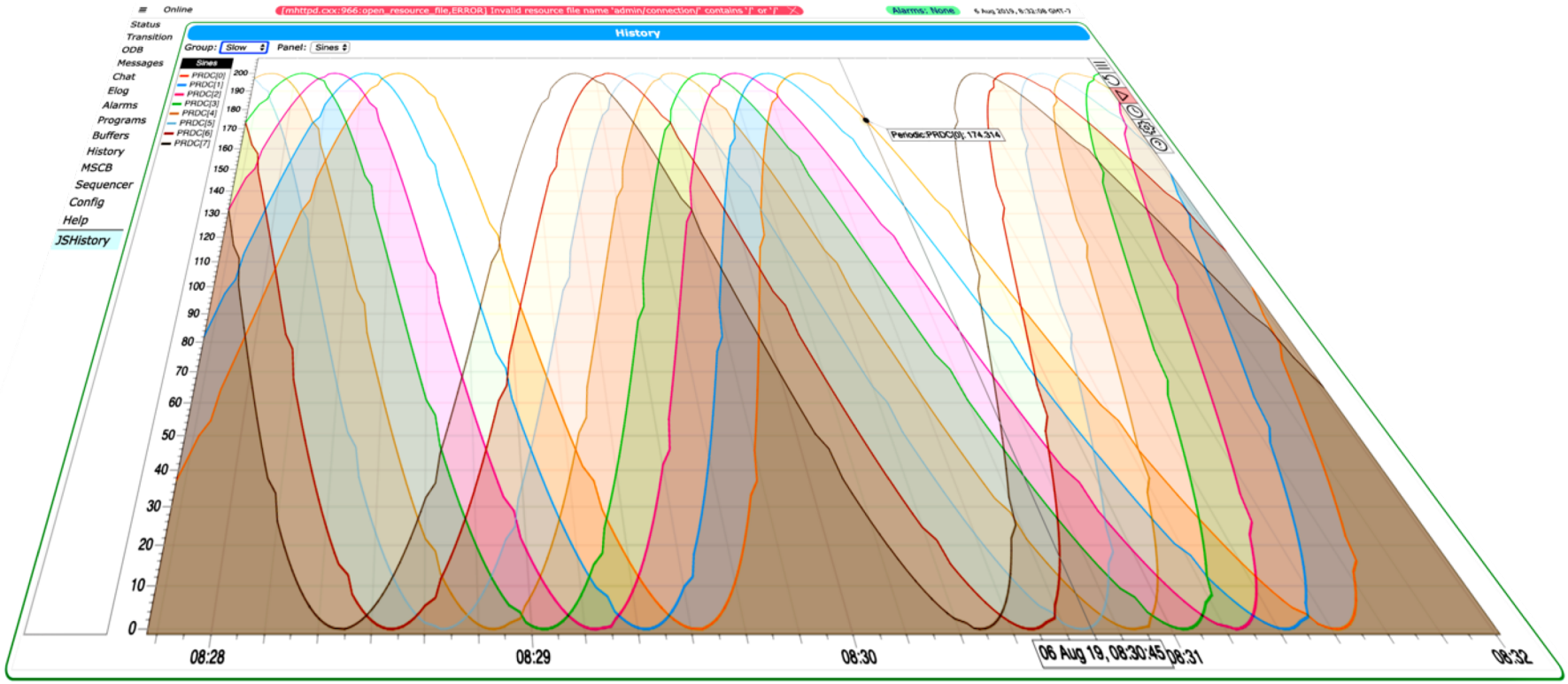


Stefan Ritt

MIDAS NEWS

Agenda

- Some news since last workshop at TRIUMF 2017
- Transition to CMake
- Review of new custom pages
- New interactive history plot
- Transition to C++ 11 (part one)

CMake

Building midas

- Traditionally, midas is built with
  - Makefile (UNIX)
  - Xcode project (Mac)
  - VS project (Windows)
- Proposal by Mu3e group in Mainz to move midas to CMake in May 2019
- By now we converted most things to CMake, but keep old Makefile for backup

CMake in a nutshell

CMakeLists.txt

```
add_library(midas STATIC midas.cxx odb.cxx ...)
target_include_directories(midas PUBLIC ${PROJECT_SOURCE_DIR}/include)

...

add_executable(mlogger mlogger.cxx ...)
target_link_libraries(mlogger midas)

add_subdirectory(progs)
add_subdirectory(examples)
```

target "mlogger" inherits include directory

CMakeLists.txt

```
add_executable(mhttpd ...)
```

CMakeLists.txt

```
add_executable(frontend ...)
```

Things which have changed

- Build midas (library, programs AND examples):
  $ mkdir build; cd build
  $ cmake .. ($ cmake3 ..)
  $ make
  $ make install

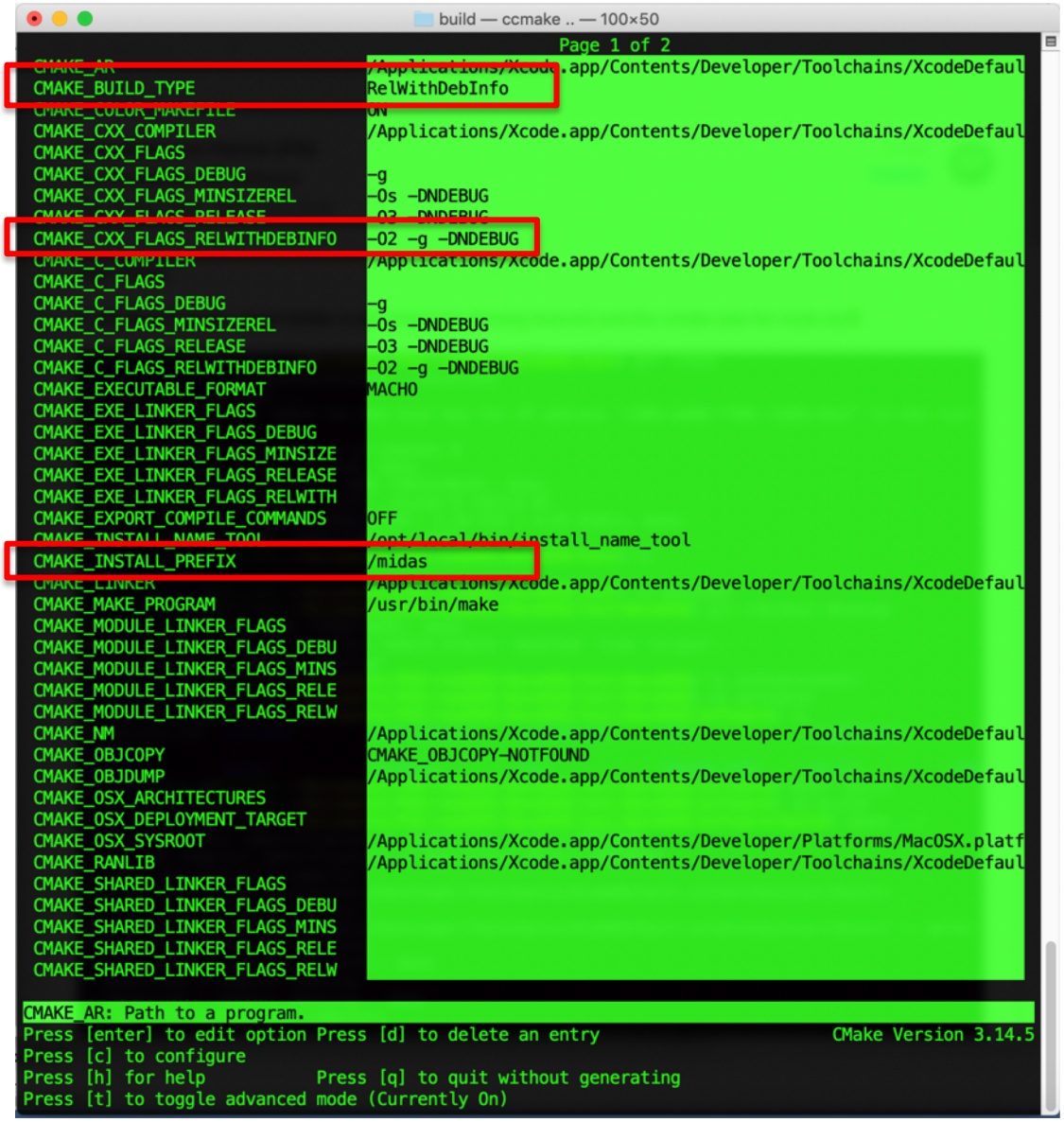
- Changed directory structure:
  src/ library sources
  progs/ midas programs
  build/ temporary build files
  lib/ midas library (was linux/lib)
  bin/ executables (was linux/bin)

- Install to **/usr/local** not supported any more

Some tips and tricks

- Multi-thread build:
  - export MAKEFLAGS=-j$((`nproc`+1)) [Linux]
  - export MAKEFLAGS=-j$((`sysctl -n hw.ncpu`+1)) [Mac]
  - improves build time by ~5x on 8-core machine
- Build type:
  - Default: "RelWithDebInfo" [-O2 –g]
  - $ make CMAKE_BUILD_TYPE=Debug [-g]
  - $ make CMAKE_BUILD_TYPE=Release [-O3]
  - Change permanently with "ccmake .."

Editing CMake cache with "ccmake .."



Building your frontend out-of-tree

```
cmake_minimum_required(VERSION 3.0)
project(frontend)

# Check for MIDASSYS environment variable
if (NOT DEFINED ENV{MIDASSYS})
   message(SEND_ERROR "MIDASSYS environment variable not defined.")
endif()

set(CMAKE_CXX_STANDARD 11)
set(MIDASSYS $ENV{MIDASSYS})

if (${CMAKE_SYSTEM_NAME} MATCHES Linux)
   set(LIBS -lpthread -lutil -lrt)
endif()

add_executable(frontend frontend.cpp)

target_include_directories(frontend PRIVATE ${MIDASSYS}/include)
target_link_libraries(crfe ${MIDASSYS}/lib/libmfe.a ${MIDASSYS}/lib/libmidas.a ${LIBS})
```

Building your frontend in-tree

```
add_subdirectory(your_frontend)
```

```
add_executable(frontend frontend.cpp)
target_link_libraries(frontend mfe midas ${LIBS})
```

Versioning with other packages

- In complex online environments, you need certain software versions to work with each other
  - online software
  - ROOT
  - perl 5
  - mysql
  - midas
- Versions evolve over time
- To go back in time, you have to manually combine right versions
- This can be automatized with git submodules

MEG software Wiki:

Minimum requirement of ROOT

from 1/Mar/2014	v5-34-17
from 27/Mar/2013	v5-34-00
from 1/Mar/2012	v5-32-00
from 1/Mar/2011	v5-30-00
from 1/Mar/2010	v5-26-00
from 18/Mar/2009	v5-22-00
from 1/Mar/2008	v5-18-00
from 1/May/2007	v5-14-00
from 1/Jan/2007	v5-12-00

Midas as a submodule

```
$ git submodule add git@butbucket.org:tmidas/midas
```

```
<your experiment>
    /online
    /analyzer
    /root
    /midas   →    fe0aa8d438a25157fb6
```

- Make sure your project compiles with current midas
- Committing your project commits a certain midas revision
- Checking out a specific review gets automatically the compatible midas version
- Compile recursively with CMake

```
$ git checkout 3ef0d3a
$ git pull --recurese-submodules
```

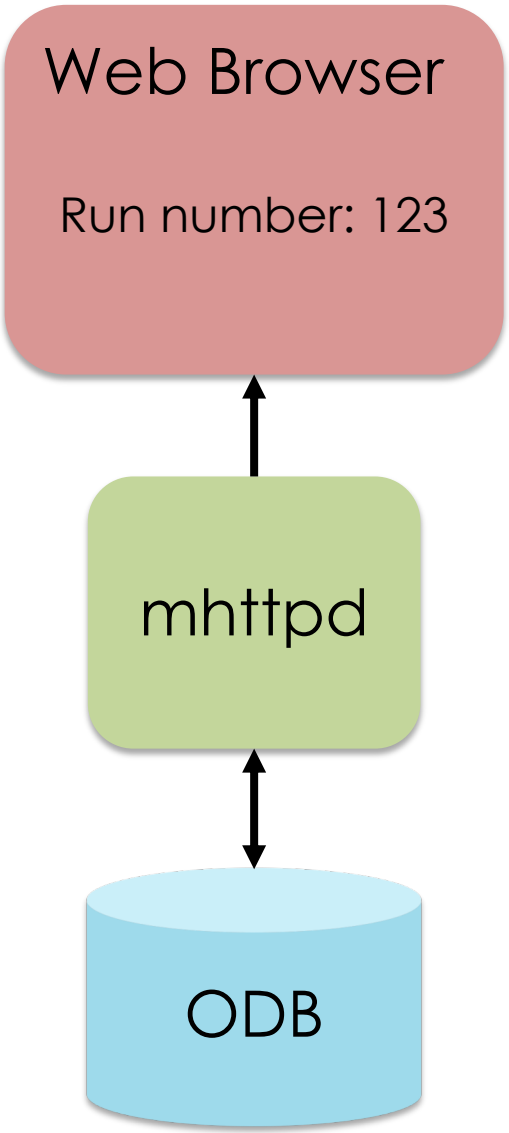
CMakeLists.txt

```
add_subdirectory(midas)
```

Custom Pages

Old Custom pages (reminder)

```
<html>

Run number: <odb path="/Runinfo/Run number">

</html>
```

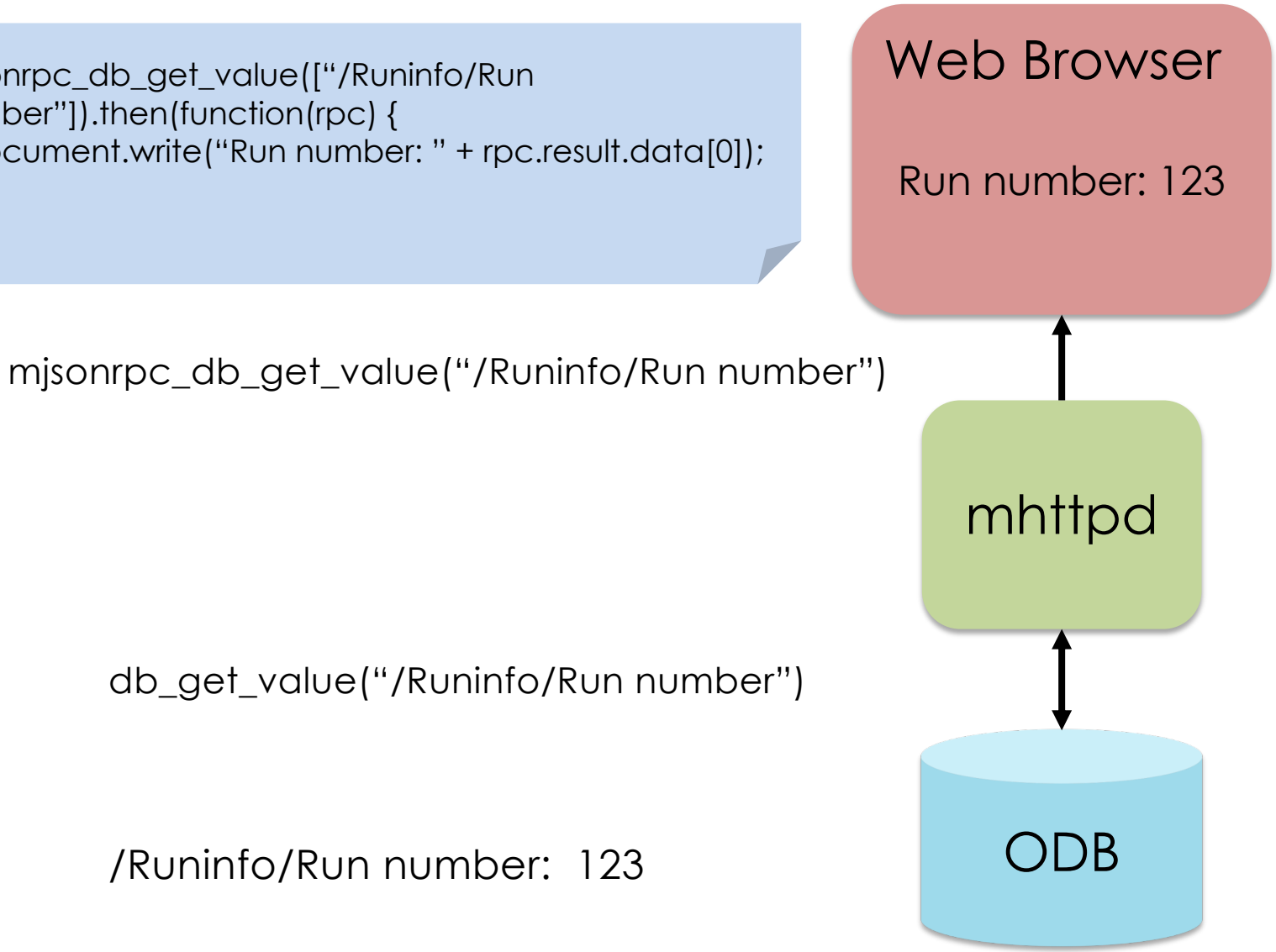


db_get_value("/Runinfo/Run number")

/Runinfo/Run number: 123

Old Custom pages (reminder)

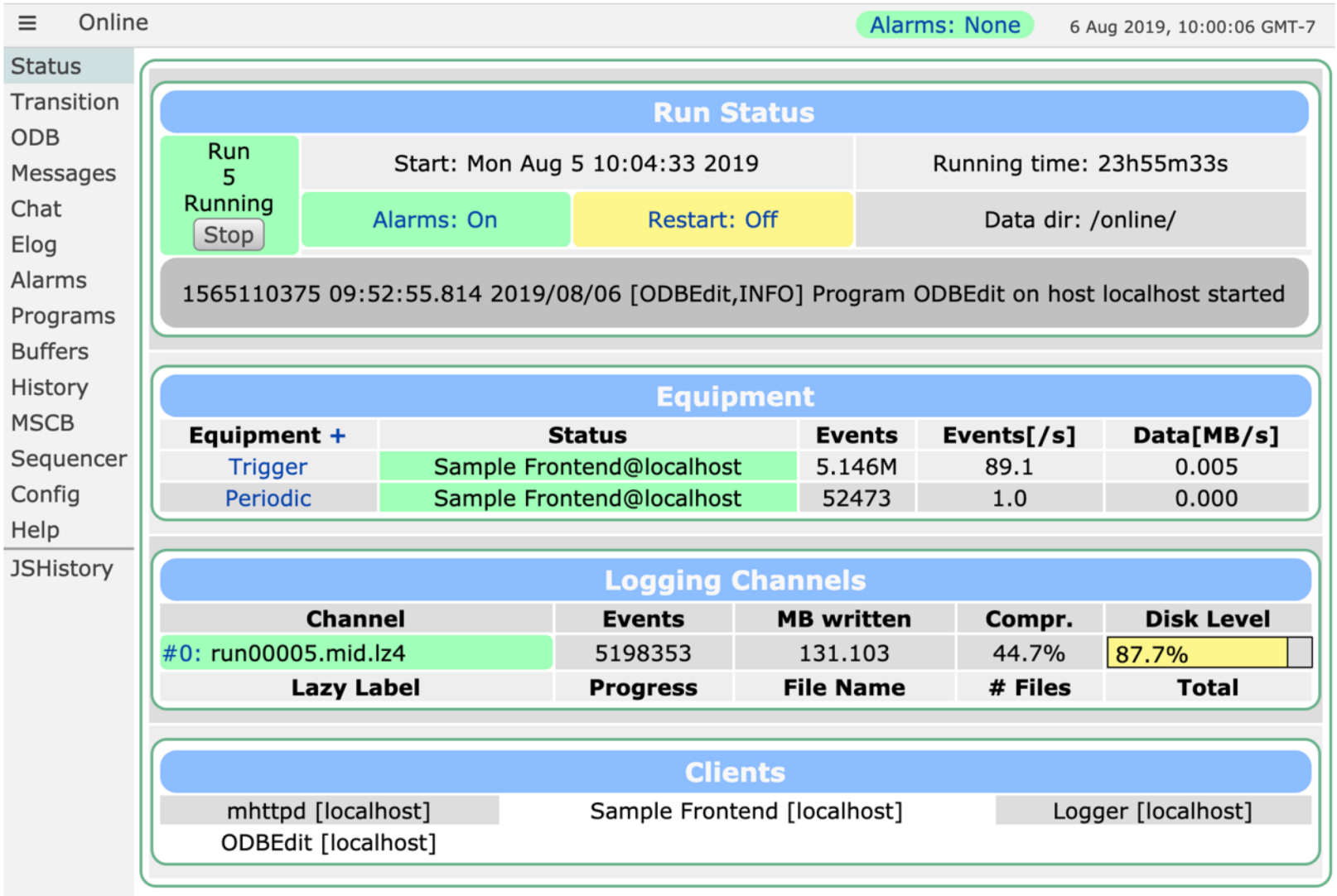
```
mjsonrpc_db_get_value(["/Runinfo/Run
number"]).then(function(rpc) {
   document.write("Run number: " + rpc.result.data[0]);
}
```



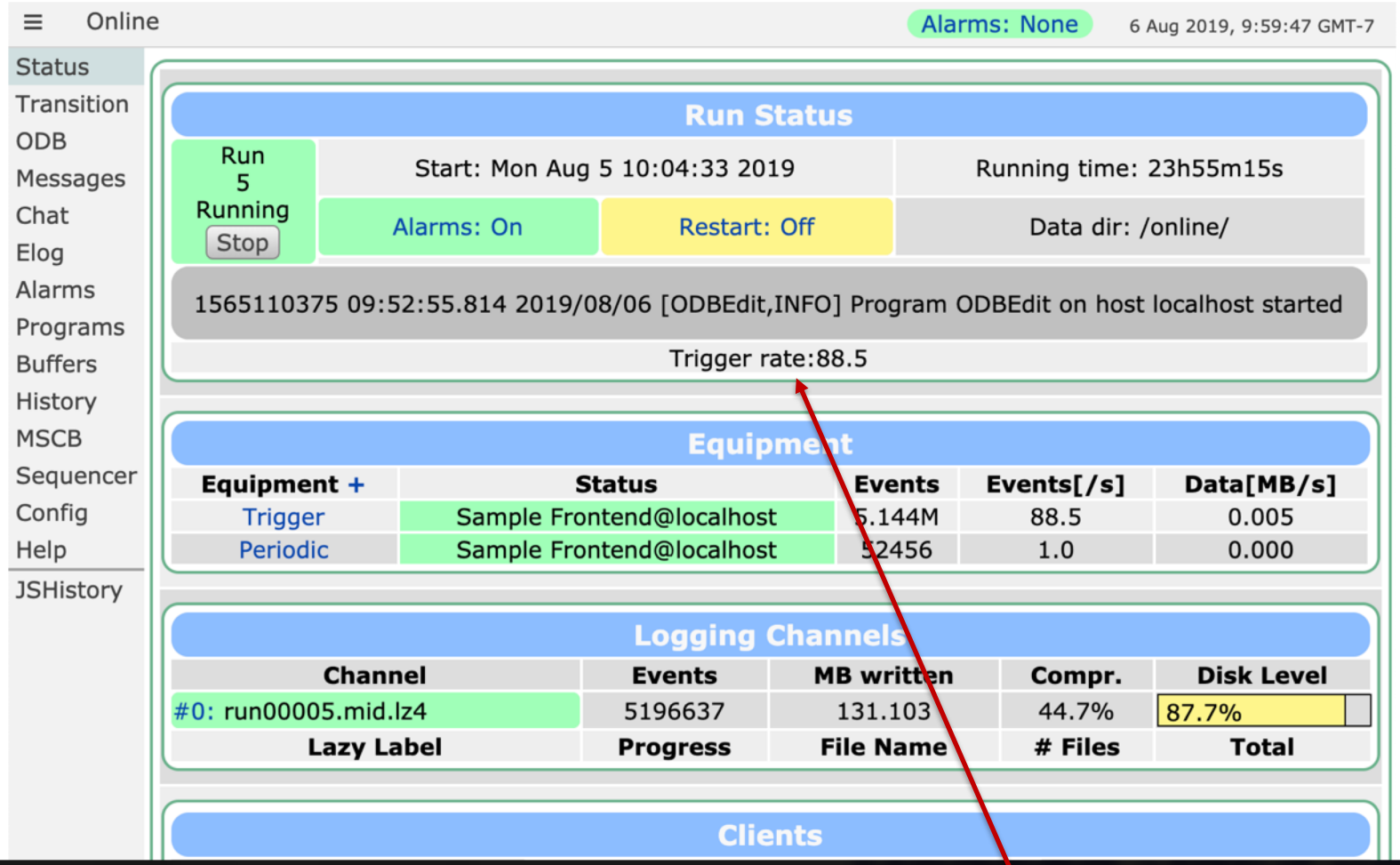
New Custom Pages

- Pages are independent of mhttpd:
  Changes do not require to re-compile mhttpd
- mhttpd becomes a simple file web server and RPC server to access midas functions from web pages
- Use full potential of HTML5: interactive web pages à la Google Mail or Google Maps
- "Standard" mhttpd pages can be customized easily

Standard Status Page

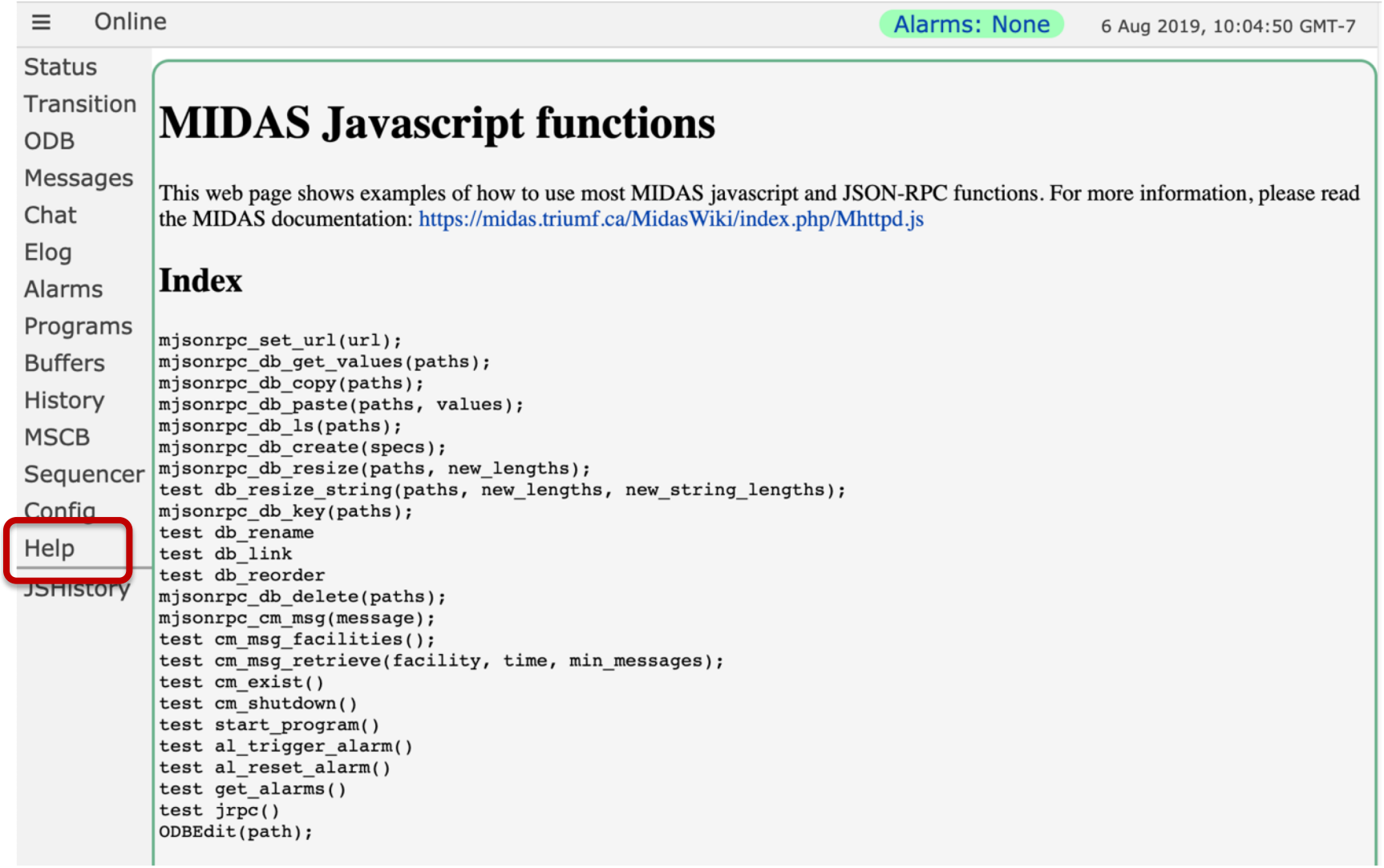


Modified Status Page




```
<tr>
  <td colspan="6" align="center">
    Trigger rate:<span name="modbvalue" data-odb-path="/Equipment/Trigger/Statistics/Events per sec." data-format="f1"></span>
  </td>
</tr>
```

mjsonrpc_xxx functions



modb* JavaScript Scheme

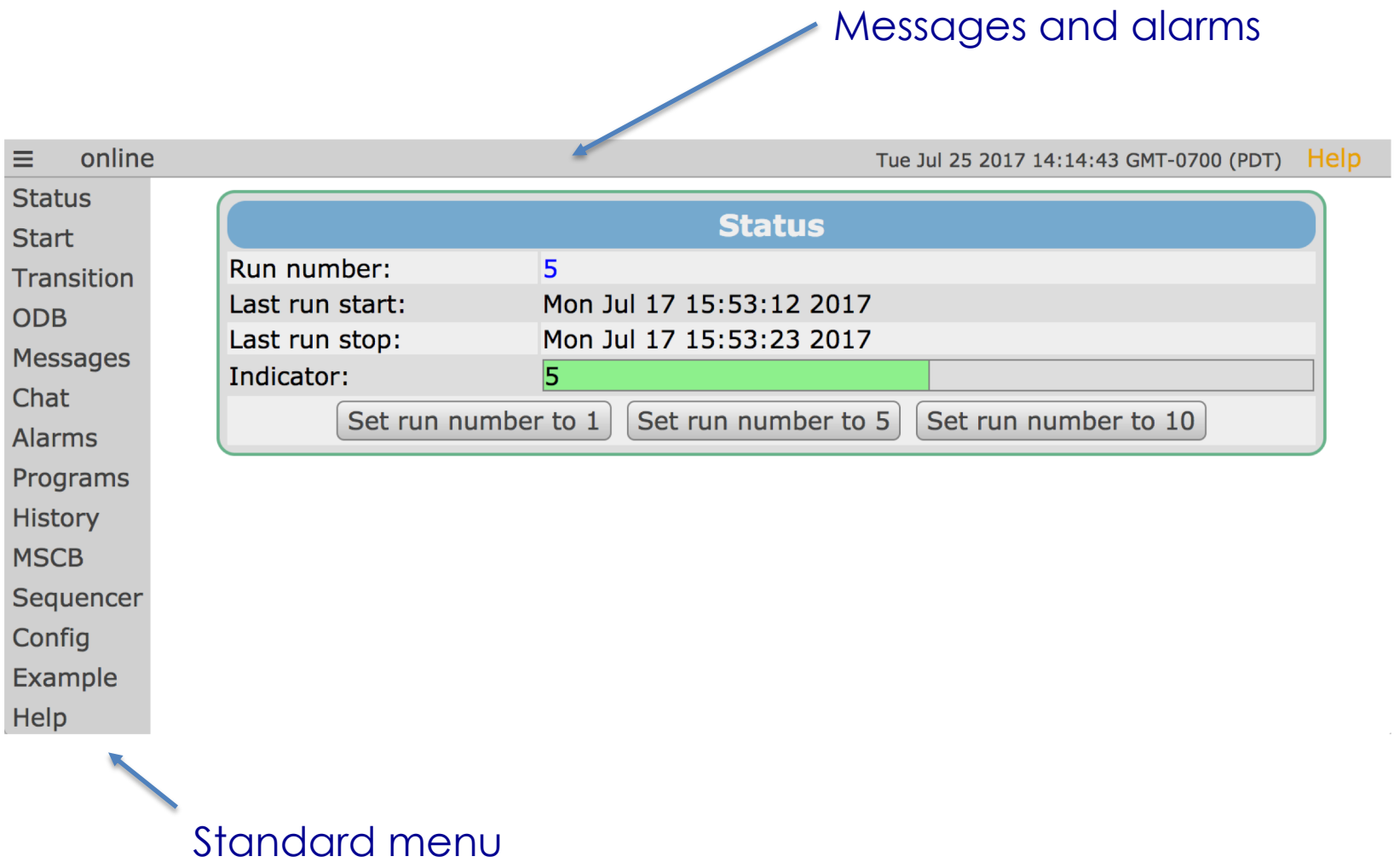
```
mjsonrpc_db_get_value(["/Runinfo/Run number"]).then(function(rpc) {
   document.getElementById('run_number').innerHTML = rpc.result.data[0];
}).catch(function(error) {
   mjsonrpc_error_alert(error);
});
```

```
<body class="mclass" onload="mhttpd_init('myPage');>

…

<div name="modbvalue" data-odb-path="/Runinfo/Run number"></div>
```

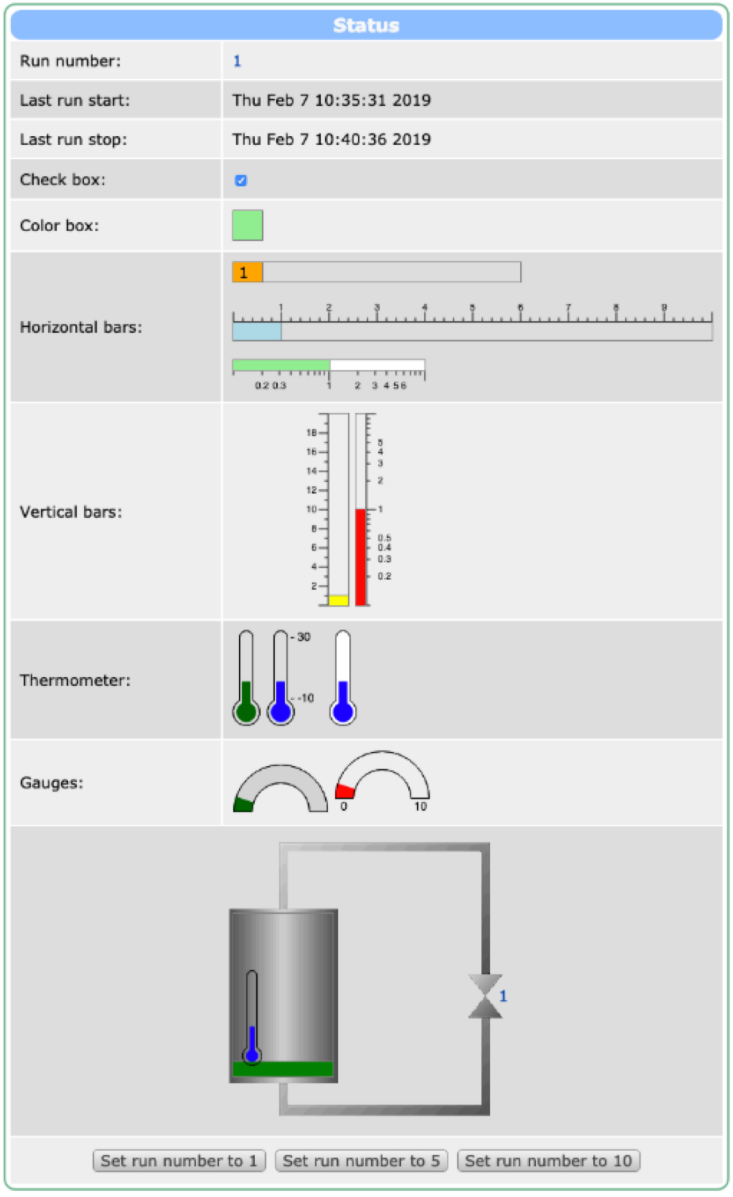
Complete example (reminder)

```
<body class="mcss" onload="mhttpd_init('Test', 1000)">

<div id="mheader"></div>
<div id="msidenav"></div>

<div id="mmain">
  <table class="mtable">
    <tr>
      <th colspan="2" class="mtableheader">Status</th>
    </tr>
    <tr>
      <td style="width: 200px">
        Run number:
      </td>
      <td>
        <div name="modbvalue" data-odb-path="/Runinfo/Run number" data-odb-editable="1"></div>
      </td>
      <td>
        <div name="modbbar" style="width: 500px" data-odb-path="/Runinfo/Run number" data-max-value="10"
             data-color="lightgreen"></div>
      </td>
    </tr>
    <tr>
      <td>
        <button name="modbbutton" class="mbutton" data-odb-path="/Runinfo/Run number" data-odb-value="1">Set run
          number to 1
        </button>
      </td>
    </tr>
  </table>
</div>
</body>
</html>
```

Resulting HTML page



https://midas.triumf.ca/MidasWiki/index.php/Custom_Page



modbvalue

modbcheckbox
modbbox

modbhbar + mhaxis

modbvbar + mvaxis

modbthermo

modbgauge

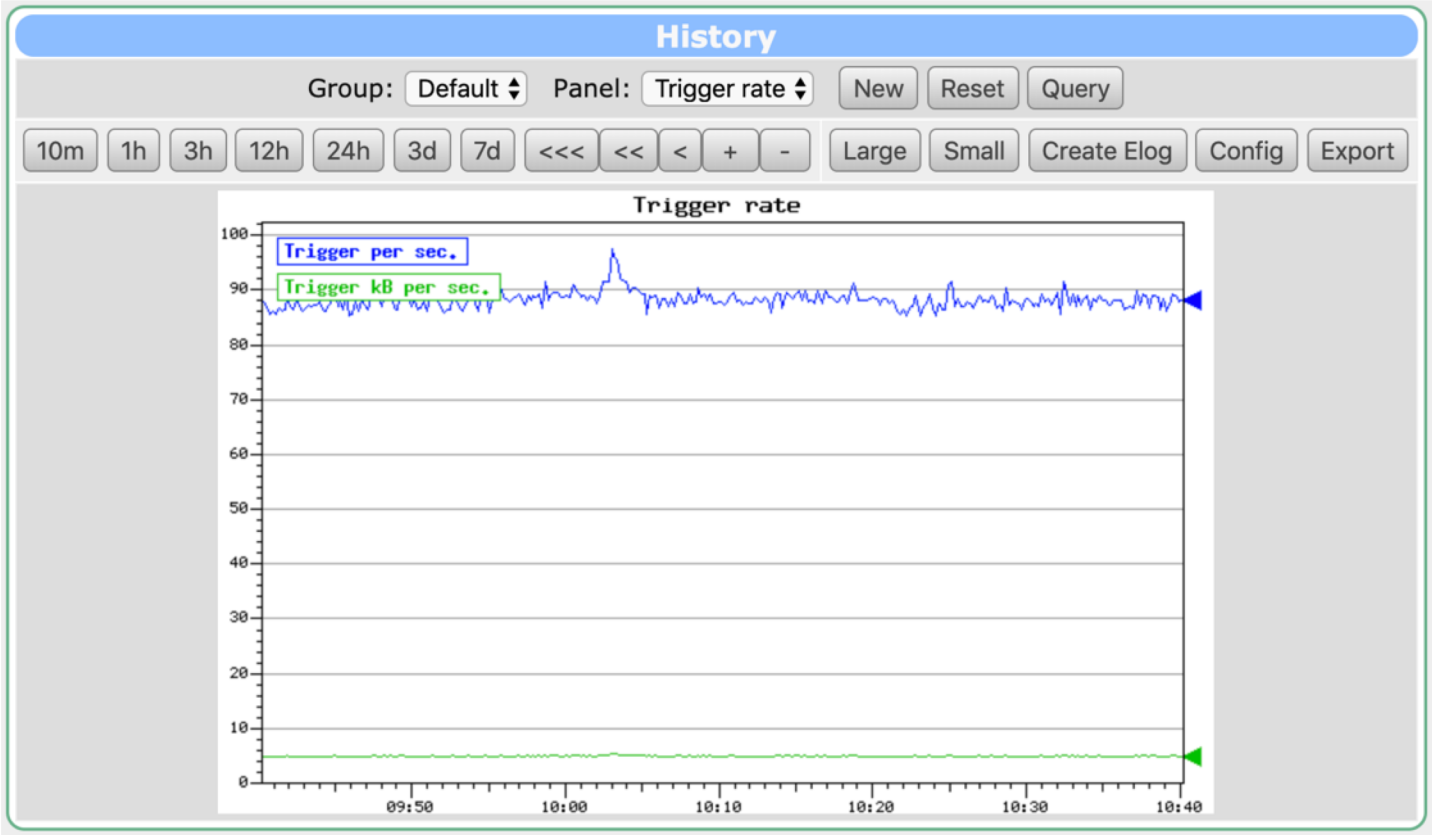
modbthermo + modbvbar
on top of image

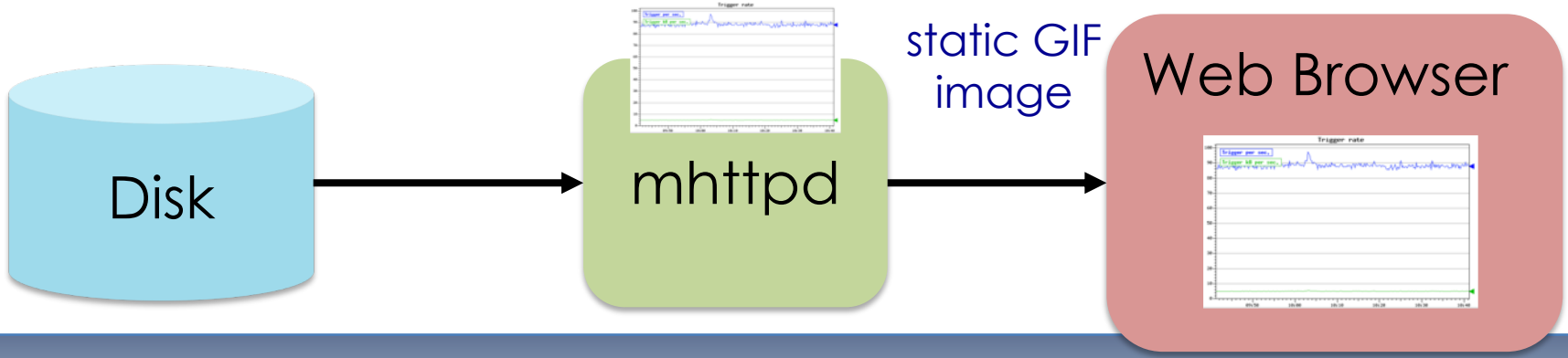
modbbutton

Conversion of standard midas pages

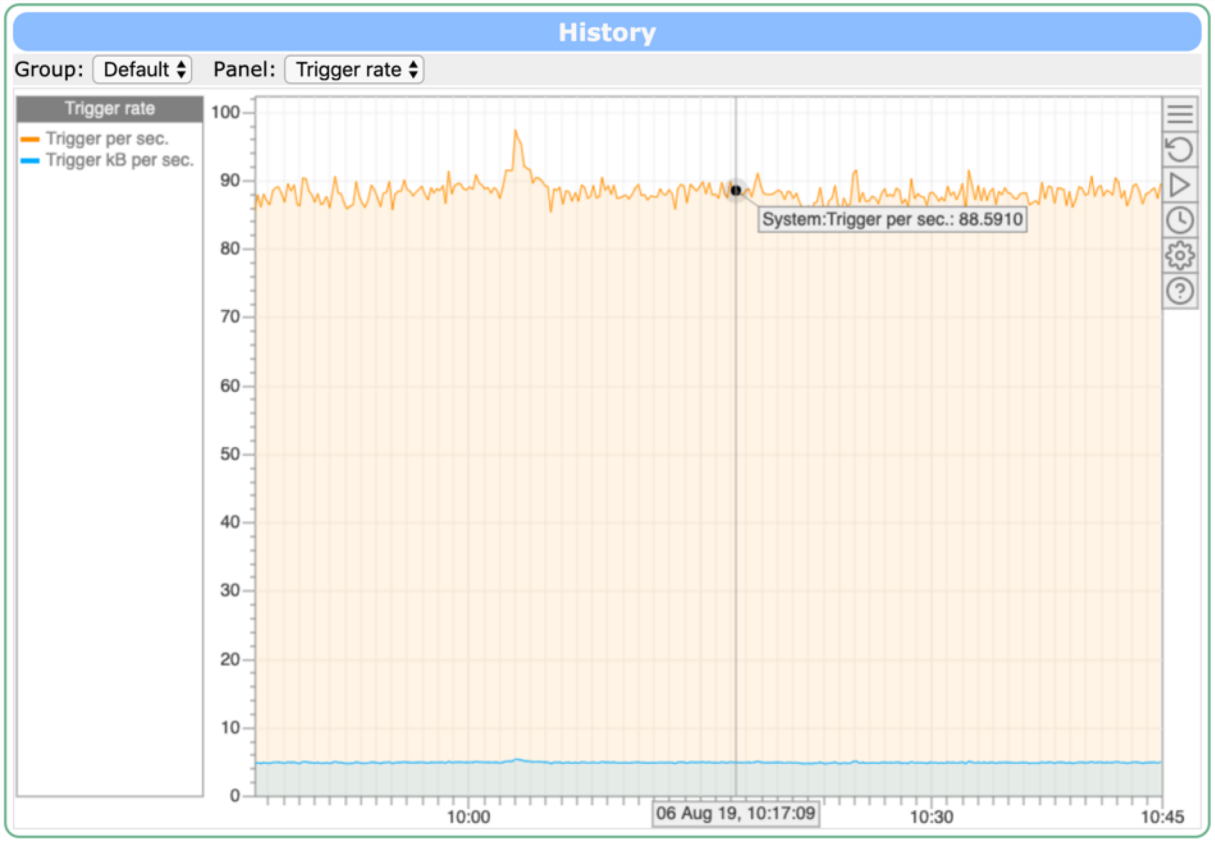
Page	Status
Status	OK (Shouyi Ma)
Transition	OK
ODB	TBD
Messages	OK
Chat	OK
Elog	OK
Alarms	OK
Programs	OK
Buffers	New (KO)
History	WIP
Sequencer	TBD
Config	OK
Help	TBD

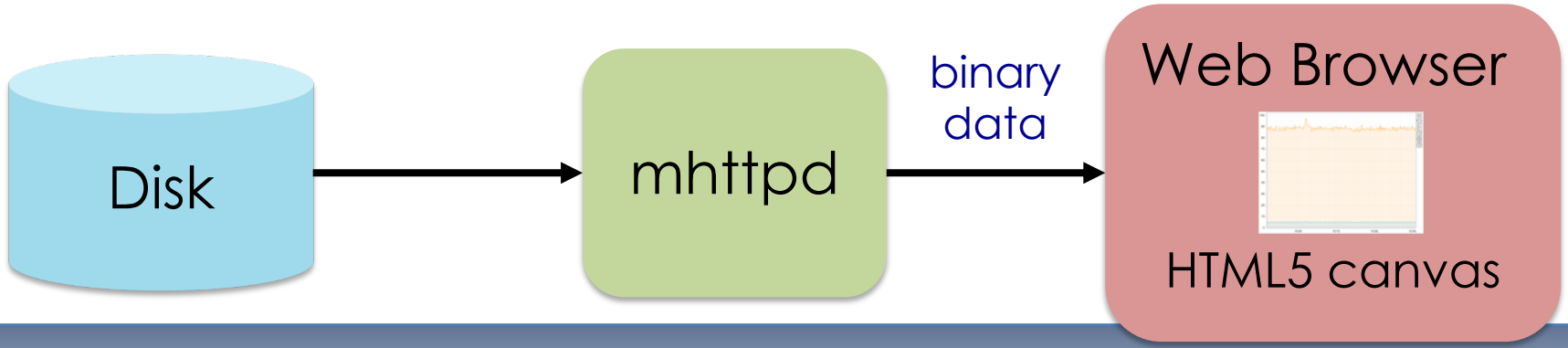
Old History


History
Group: Default
Panel: Trigger rate
New
Reset
Query
10m
1h
3h
12h
24h
3d
7d
<<<
<<
<
+
-
Large
Small
Create Elog
Config
Export
Trigger rate
Trigger per sec.
Trigger kB per sec.
100
90
80
70
60
50
40
30
20
10
0
09:50
10:00
10:10
10:20
10:30
10:40


Disk
mhttpd
static GIF image
Web Browser

New History


History
Group: Default
Panel: Trigger rate
Trigger rate
Trigger per sec.
Trigger kB per sec.
100
90
80
70
60
50
40
30
20
10
0
System:Trigger per sec.: 88.5910
10:00
06 Aug 19, 10:17:09
10:30
10:45


Disk
mhttpd
binary data
Web Browser
HTML5 canvas

Interactivity

- Live scrolling
- Panning and zooming (à la Google Maps)
- Inspect values with cursor
- Dynamic loading of data
  - Update only pulls new data points since last update
  - Un-zooming pulls old data on request
  - Switching to binary data (JS typed arrays) for speed improvements
- Custom page:

```
<div data-name="mjshistory"
 data-group=”Default"
 data-panel=”Trigger rate"></div>
```

Midas and C++

Switch to C++

- Around June 2019
  midas has been switched to C++
- Should we settle on C++ 11 ?
- Some minor changes in user code:
  https://midas.triumf.ca/elog/Midas/1526
- Converted most device drivers, few are missing (mainly type casts)
- This offers new possibilities → next talk
- This talk: C++ interface to ODB

Long standing problem

ODB

```
/Equipment/Trigger/Common
        Event ID                1
        Trigger mask            0
        Buffer                  SYSTEM
        Source                  0
        Format                  MIDAS
        …
```

C program

```
typedef struct {
        int event_id;
        int trigger mask;
        char buffer[32];
        int source;
        char format[32];
        …
} COMMON;

COMMON c;
c.event_id = 1;
```

```
[/Equipment/Trigger/Common]
Event ID = WORD : 1
Trigger mask = WORD : 0
Buffer = STRING : [32] SYSTEM
Type = INT : 2
Source = INT : 0
Format = STRING : [8] MIDAS
Enabled = BOOL : y
```

- Need ASCII string to create ODB entries
- Use db_get_record() to link C structure to ODB
- Issues: order of variables, size of strings, …

New Idea

- Inspired by Niels Lohmann: JSON for modern C++ https://github.com/nlohmann/json

```
#include <nlohmann/json.hpp>

using json = nlohmann::json;

int main()
{
   json j = {
     { "pi", 3.141 },
     { "happy", true },
     { "answer", {
        { "everything", 42 }
     }},
     { "name", "midas" },
     { "list", {1, 0, 2} }
   };

   j["pi"] = 3.14156;
   std::cout << j["pi"] << std::endl;
}
```

- Just one header file
- Vanilla C++11
- JSON object almost feel like native C++ literal
- Use JSON like an STL container
- Automatic type deduction

New C++ ODB API "mdata"

```
#include <mdata.hpp>

using mdata = midas::mdata;

int main()
{
   cm_connect_experiment(…);

   mdata d("/Equipment/Trigger/Common");

   std::cout << d["Format"] << std::endl;

   d["Event ID"] = 1;

   d["Format"].addListener([](mdata &d) {
      std::cout << d << std::endl;
   });
}
```

"d" contains a local copy of ODB subtree

send data to ODB

hot-links via lambda

Some additional thoughts

- Do not break current C API, just add new one on top

```
#include <mdata>
```

- Let compiler deduce data types

```
d["value1"] = 1.23;   // double TID_DOULBE
d["value2"] = 1.23f;  // float  TID_FLOAT
```

- Nested and array data

```
mdata d("/Equipment/Trigger");
d["Common"]["Event ID"] = 1;
std::string s = d["Settings"]["Names"][15];
```

Some additional thoughts

- Bunch write operations

```
d.startTransaction();
d[“a”] = 1;
d[“n”] = “hello”;
d.commitTransaction();
```

- Serialization to JSON files

```
ofstream f;
f.open(“example.json”);
f << d << endl;
f.close();
```

- Access ODB dumps and databases

```
mdata d(“file:run123.mid”);
mdata d(“couchdb:midas.triumf.ca/db”);
```

- Any other idea?