Using MATLAB to script your MDs

G. Sterbini, D. Gamba

26 May, 2014

During MD, usually we need to perform repetitive tasks not needed in normal operation. The standard operation software is not designed for MD and using it in MD is not always comfortable. In the following we share our experience on using the JAPC classes to interface to MATLAB and to write easy-to-handle MD scripts.

Disclaimer I

We are neither CO nor JAVA expert: we are here mainly to share with you our doubts and mistakes.

Disclaimer II

We assume you were partially exposed to the CCC control software, MATLAB language and the object oriented programming.

Disclaimer III

When you change the machine settings you need to be aware of the impact of your actions.

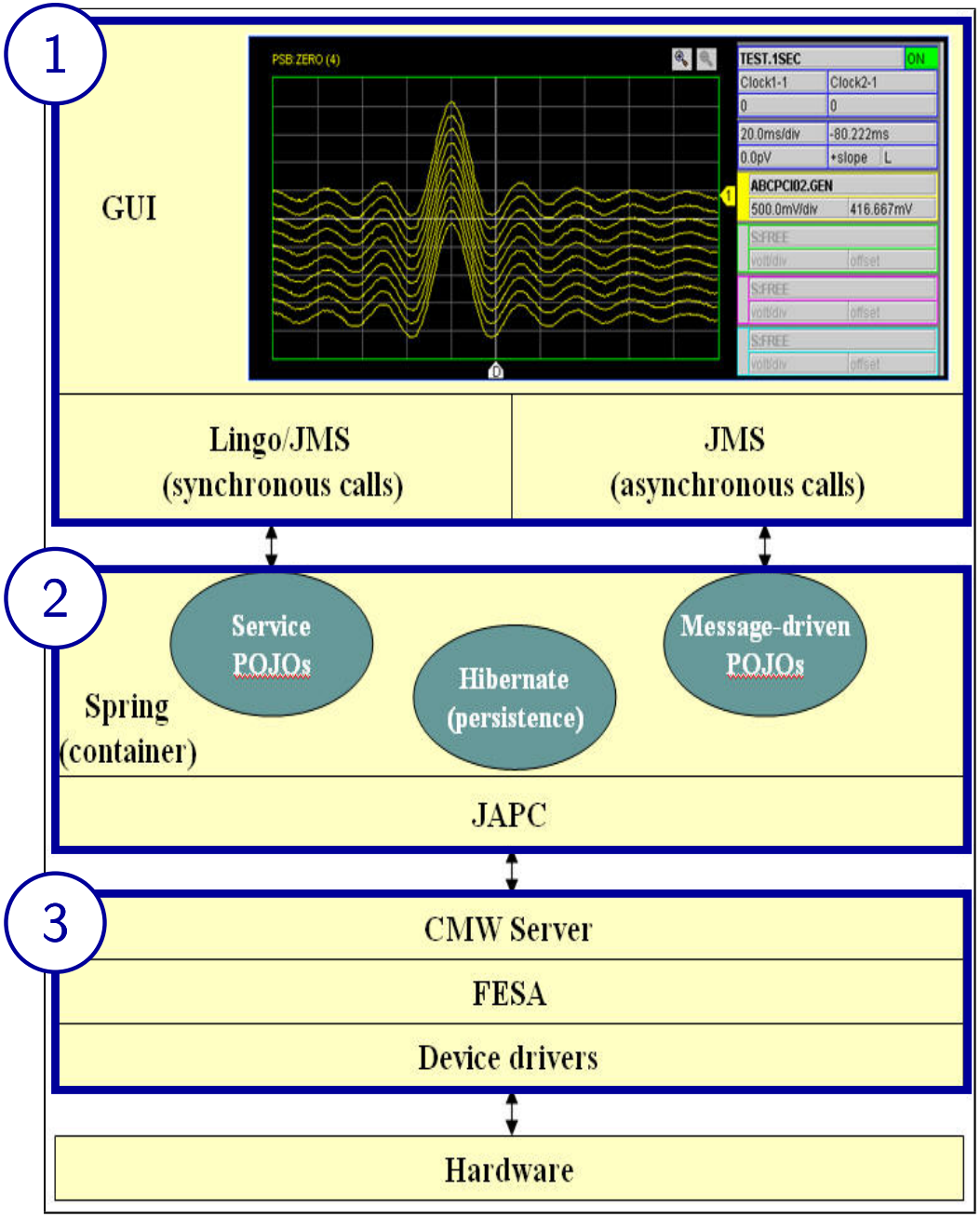
Outline

A common vocabulary I: CERN CO architecture

There are 3 levels for controlling the HW

1. Java applications launched by the Common Control Manager (CCM),
2. JAPC framework,
3. FESA classes.

In these slides we will show how to use MATLAB to work in the level 2 instead of the (usual) level 1.



A common vocabulary I: what is *JAPC*?

"JAPC is a framework to build Java applications that control accelerator devices. Its central concept is a parameter, which typically represents a controls value of an accelerator device (also known as "device property") but actually can provide access to everything accessible (equipments, databases, timing events, intermediate servers, simulation). Client programs can access JAPC parameters with set and get interactions, and they can wait to be notified of value changes using subscription."[1]

[1] From https://wikis/display/JAPC/Home

A common vocabulary II: what is a *selector*?

CERN machines work in multi-users mode where different cycles in the super cycle perform different tasks (produce different beams) for different users (Pulse-to-Pulse Modulation, PPM).

If we want to change the current of a quadrupole we have to specify for which machine and user we want to change the setting. This information is in the selector string in the format "*machine*.USER.*user_name*".

- i.e., "*CPS*.USER.*ZERO*"

The selector is also called "PLS line", "PLS condition", "User Cycle", "TGM cycle", just "cycle"...: the selectors dispatching is done via the telegram.
8 machines (ADE, CPS, ISO, LEI, LHC, PSB, SCT, SPS) can be present in the selector.

Each selector is mapped with a specific LSA user.

A common vocabulary III: what is a *parameter*?

A selector owns several parameters strings: a parameter is defined by a device, a property and a field the format "*device*/*property*#*field*".

- i.e., "*CL.QDA0415*/*AQN*#*value*"

You can find >2.5M of parameters at http://wwwpsco.cern.ch.

In reality, a lot of parameters can be set/get via the WorkingSet Launcher. From this application you can launch the different WorkingSets. You can open the knob and by clicking on the label on the knob labels you have the different parameter names.

There are also important devices that are not controlled on the workingSet but using specific application: i.e., wire scanners, OASIS, samplers. To know which parameters you have to control you have to contact the developer of the Java application or of the FESA class.

Plan

The challenge

Assuming that you have a Java class, the idea is to use the class in a scripting language. This would allow us to use the Java class written by the CO team to script your own MD.

Which scripting languages can instantiate a Java class?

- MATLAB naturally can load a Java class (it is written (also) in Java).
- Mathematica (JLink).
- Jython.
- Python does not natively (one can use Py4j module)[2].

Since we started with MATLAB, in this presentation we will use MATLAB (I know, MATLAB is not free...).

[2] Pythoneers are challenged.

A simple Java class

- A simple class with 1 property, 2 methods and 0 event.

```
public class quadrupole{
private double current;
public quadrupole(){
setCurrent(0);
}
public void setCurrent(double aCurrent){
current = aCurrent;
}
public double getCurrent(){
return current;
}
}
```

- Compiling it

```
javac quadrupole.java
```

- Making an archive

```
jar -cvf quadrupole.jar quadrupole.class
```

The magic (in MATLAB)...

- Configure the JAVA path in MATLAB

```
>> javaaddpath /afs/cern.ch/work/s/sterbini/...
public/matlab/forYourMD/quadrupole.jar
```

- Instantiate the class in MATLAB

```
>> myQuad=quadrupole;
```

- Use the class in MATLAB

```
>> myQuad.setCurrent(1.2)
>> myQuad.getCurrent
ans =
     1.2
```

This class is eventless. It is very simple/boring to deal with it.

The magic (in Mathematica[3])...

- Instantiate the class in Mathematica

```
Needs["JLink`"];
InstallJava[];
AddToClassPath["/afs/cern.ch/work/s/sterbini/
    public/matlab/forYourMD"];
quadrupole = LoadJavaClass["quadrupole"]
myQuad = JavaNew[quadrupole]
```

- Use the class in Mathematica

```
myQuad@setCurrent[1.2]
myQuad@getCurrent[]
1.2
```

This class is eventless. It is very simple/boring to deal with it.

[3] in the respect of cultural minorities.

Plan

Back to reality

For our purpose we do not have to write/compile/archive the Java classes: they are developed and maintained by CO in the JAPC framework.

What do you have to do then?

- To find where your needed jar is.
- To setup the javaclasspath in MATLAB.
- To instantiate it.
- To learn how to use the java class in MATLAB.
- To enjoy the MD.

Reading the following few slides our goal is to leave to you only the "enjoy" part.

Let us start

What do you need?

- as usual, good luck and wishful thinking,
- a computer access on the CERN Technical Network (TN),
- a MATLAB working license on that computer.

Launch MATLAB from the TN

- Login to a console in the TN. From the GPN you can access special servers (i.e., cs-ccr-ps1)

```
ssh -XY sterbini@cs-ccr-ps1
```

- Go to

```
cd  /afs/cern.ch/work/s/sterbini/public/matlab/
    forYourMD
```

- Open MATLAB using the prepared shell script

```
./matlab.sh
```

Starting the game...

For convenience, we (read Davide[4]) created two MATLAB classes that are wrappers for the JAPC interfaces (you can forget about JAPC...).

- matlabJAPC: to GET/SET the parameter (w/o events)
- matlabMonitor: to monitory the parameter (w/ events)

In the following we will discuss some simple examples:

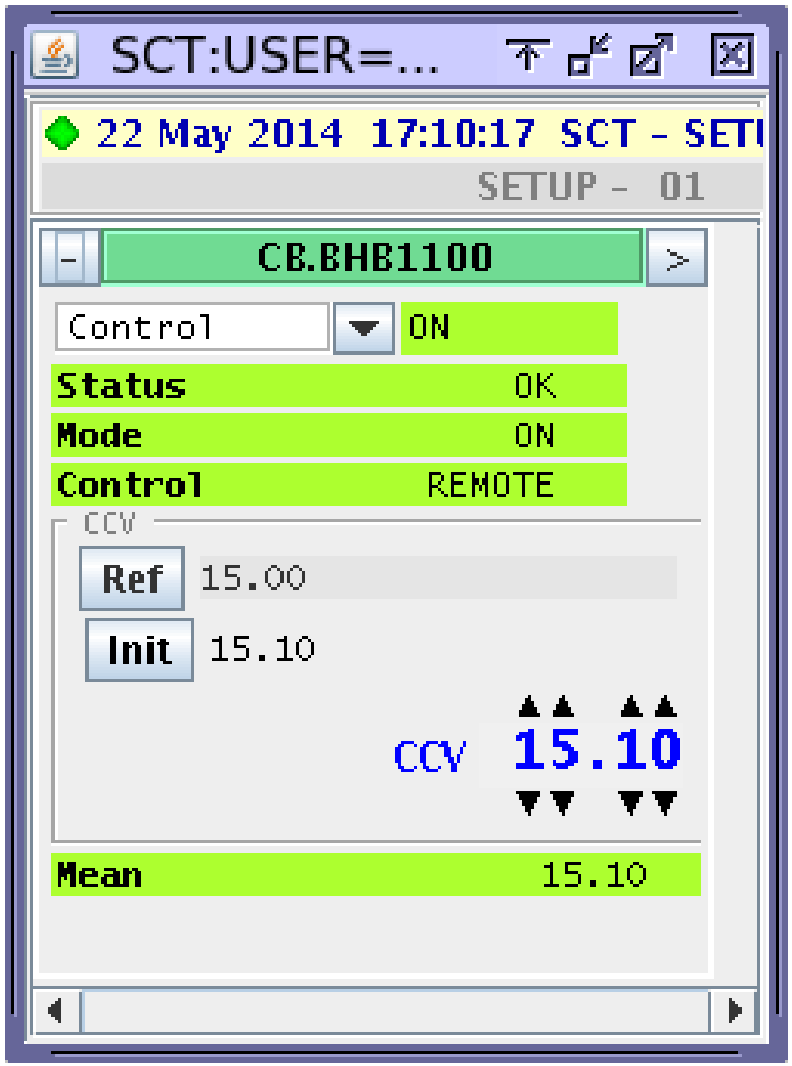
1. get/set a magnet current,
2. get a sampler signal,
3. get a OASIS signal,
4. get/set a GFA signal

and finally we will discuss the monitoring.

[4] for more information and suggestions davide.gamba@cern.ch

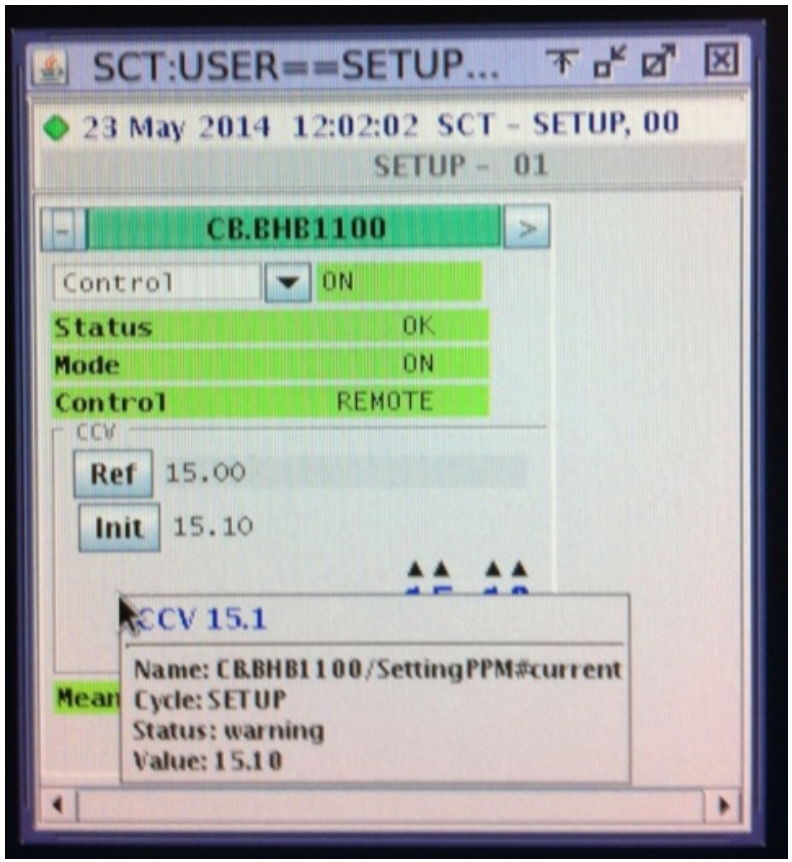
Control a simple knob

A very common situation. . .



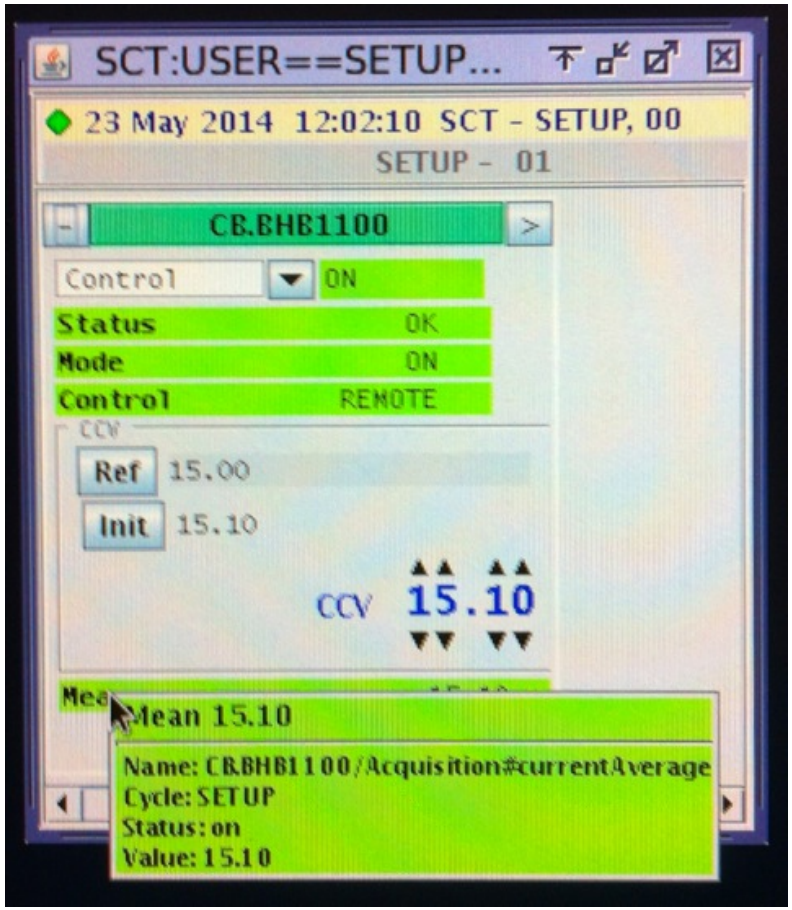
Control a simple knob

A very common situation. . .



Control a simple knob

A very common situation. . .



Control a simple knob

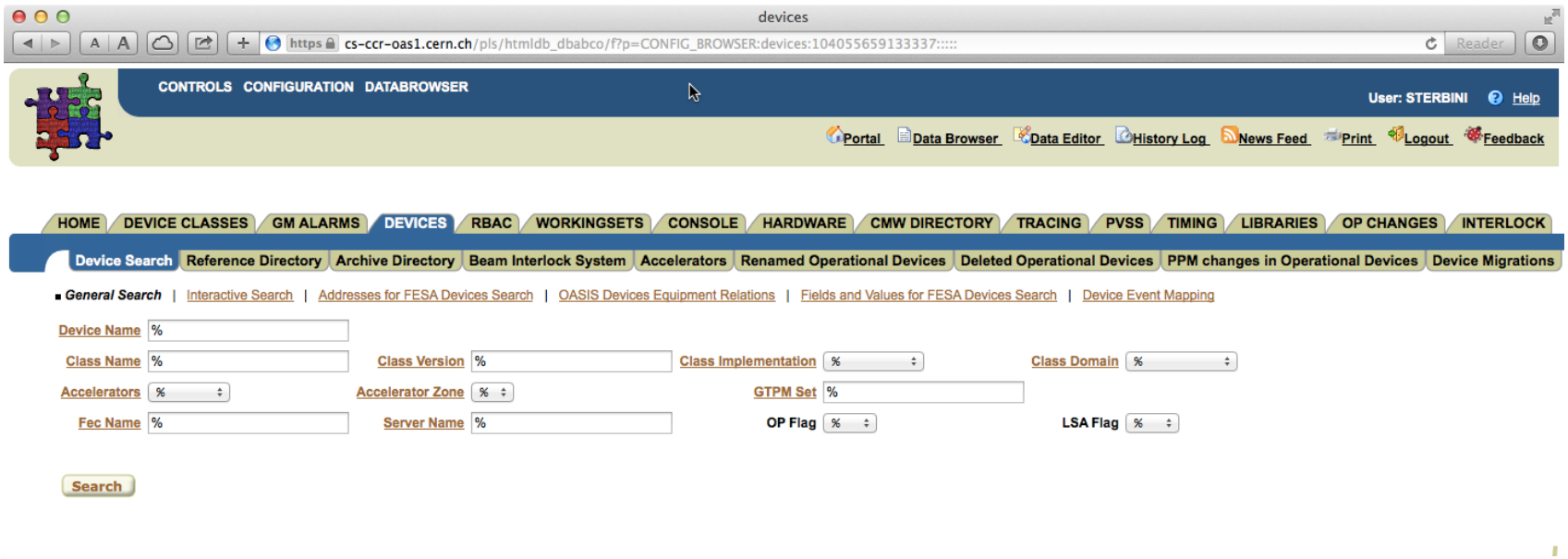
You have just to use the two static methods of matlabJAPC.

```
%% Get and set a magnet current
%
parToGet = 'CB.BHB1100/Acquisition#currentAverage';
parToSet = 'CB.BHB1100/SettingPPM#current';
mySelector = 'SCT.USER.SETUP';

% Get them
value1 = matlabJapc.staticGetSignal(mySelector,...
parToGet)
value2 = matlabJapc.staticGetSignal(mySelector,...
parToSet)

% Set it
matlabJapc.staticSetSignal(mySelector,parToSet,value2)
```

Retrieving or setting the parameters of the WorkingSet is trivial since you can immediately identify the name of the parameters. For the rest there's wwwpsco.

The wwwpsco site

The DEVICES tab has a description of all the parameters that we can get, set and monitor.



Reading a sampler I

For the moment we show how to get/set PPM scalar parameters. There are a lot of PPM parameters that are functions varying along the time cycle (beam current, cycle of the different magnetic circuit, klystron power, tune, chromaticity, ...). They are conveniently sampled by software samplers implemented in the XenericSampler FESA class.

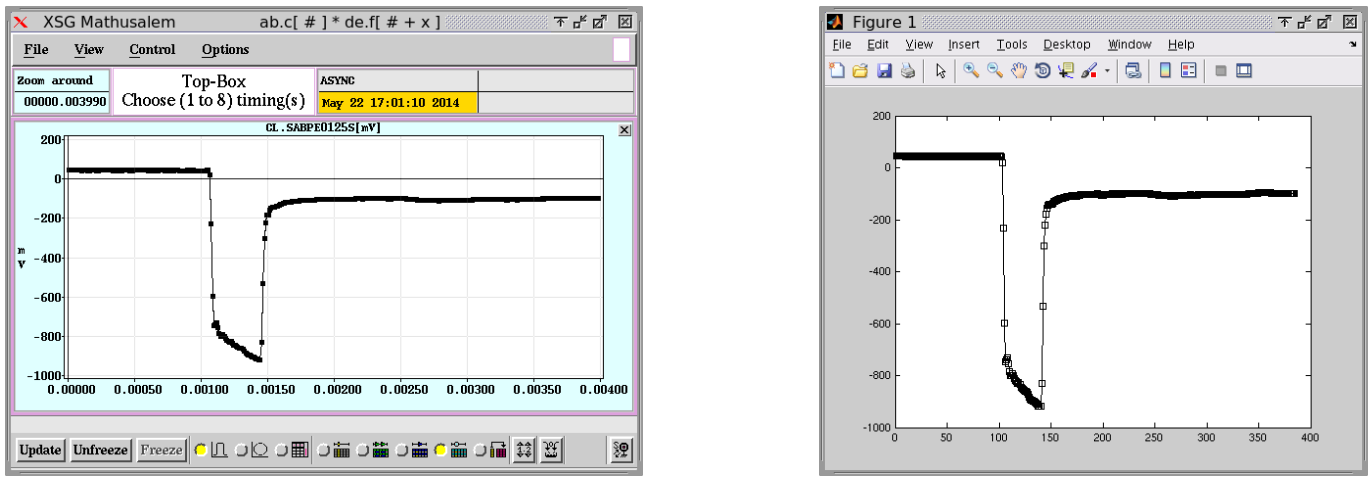


Figure: From the CCM Sampler (left), from MATLAB (right).

Reading a sampler II

Here you are the code

```
samplerSamples = 'CL.SABPE0125S/Samples#samples';
mySelector = 'SCT.USER.SETUP';

% get it
auxArray = matlabJapc.staticGetSignal(mySelector,
    samplerSamples);

% plot it
plot(auxArray,'sk-');
```

If you need more flexibility on the trigger and on the sampling you can use OASIS. OASIS allows us to connect via digitisers on the ethernet to a large set of analogue signals and triggers.

Reading an analogue signal: OASIS

```
oasisSamples = 'CL.SCOPE02.CH01/Acquisition#value';
mySelector = 'SCT.USER.SETUP';
% Get it
auxArray = matlabJapc.staticGetSignal(mySelector,
    oasisSamples);
% Plot it
plot(auxArray,'g-');
set(gca,'Color','k');
set(gcf,'Color','k');
set(gca,'XColor','w');
set(gca,'YColor','w');
grid on
```

If you are eager to acquire a > 2 MSamples/cycle of data you need other solutions (e.g., a dedicated local digitiser: a summer student will work on it).

Reading an analogue signal: OASIS

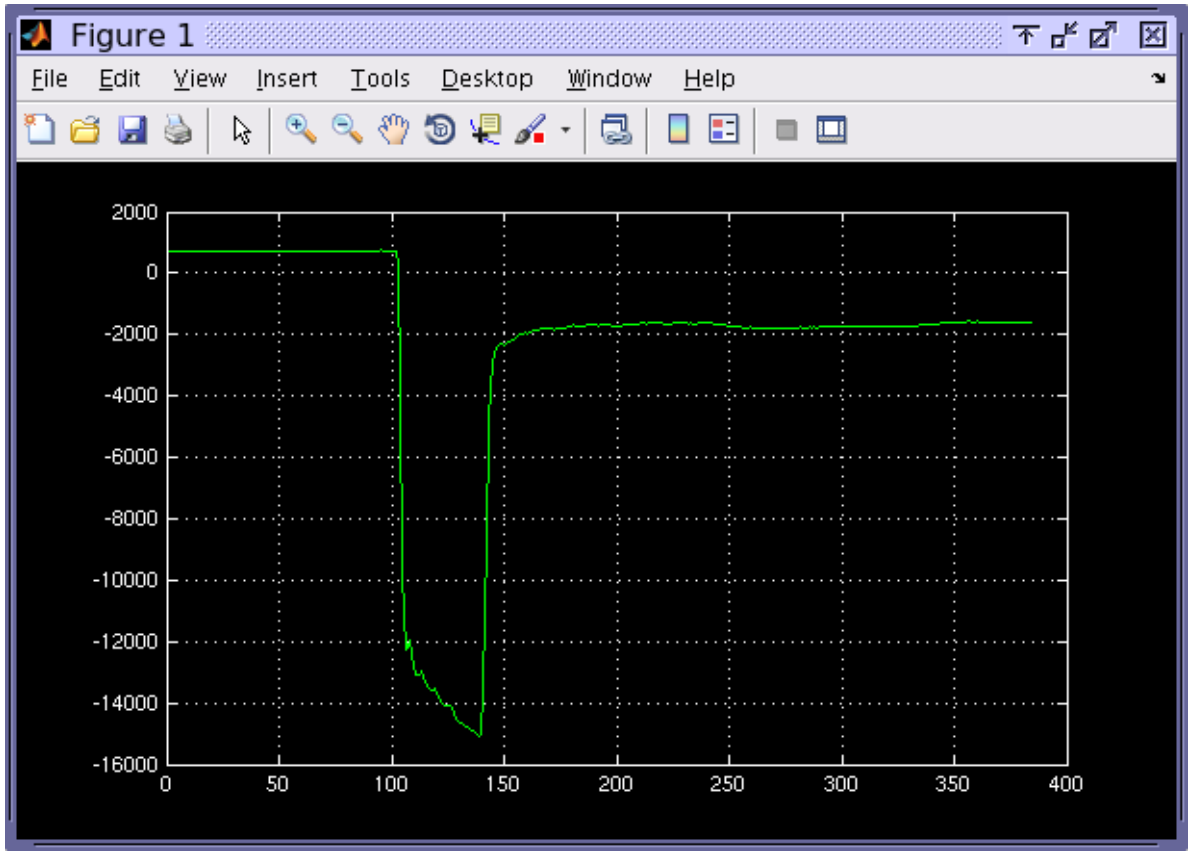
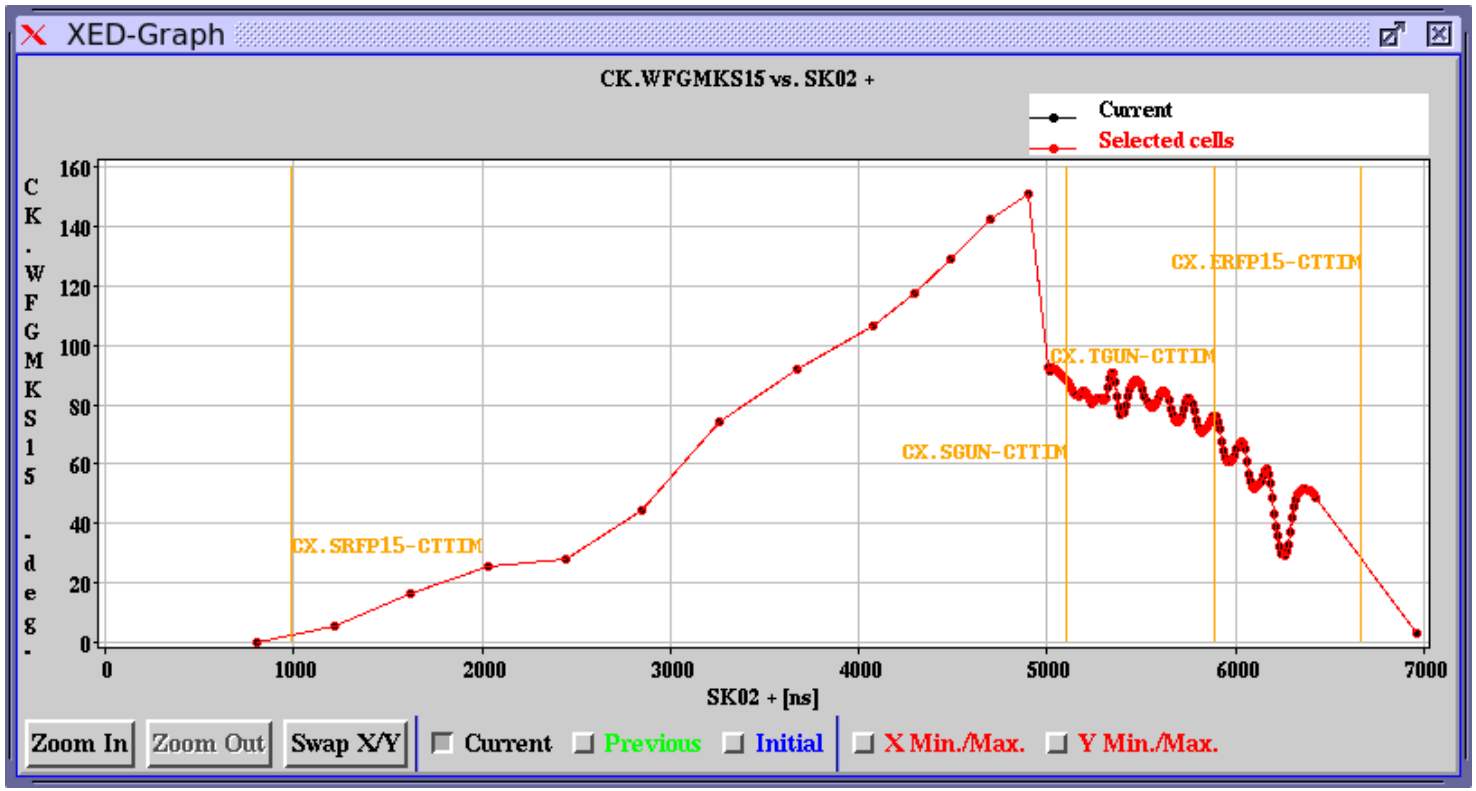


Figure: The information that we obtained from the OASIS signal in MATLAB. The scale are in arbitrary units in this plot...

Setting parameters changing along the cycle: the GFAs

We learnt how to get time function. To set a GFA (klystron phase, cavity voltage, current in a magnetic circuit...) you have to set a GFA (Générateur de fonction analogique). There are a lot of FESA classes for GFAs. Here you are an example.



Setting parameters changing along the cycle: the GFAs

```
GFA = 'CK.WFGMKS15/Profile#profile';
GFA_nsample = 'CK.WFGMKS15/Profile#profileSize';
mySelector = 'SCT.USER.SETUP';
% get it
auxArray = matlabJapc.staticGetSignal(mySelector,GFA);
auxSize = matlabJapc.staticGetSignal(mySelector,
    GFA_nsample);
auxTimeLine = auxArray(1:2:2*auxSize);
auxValues = auxArray(2:2:2*auxSize);
% plot it
plot(auxTimeLine,auxValues,'r-');
% set it
matlabJapc.staticSetSignal(mySelector,GFA_nsample,
    auxArray)
```

Setting the bar high: the monitoring

Until now we have got parameters asynchronously wrt the cycle (we can get a parameters value even when our cycle is not played). If we want to get a parameters each time the cycle is played we have to monitor the parameters. If we need to set a parameter each time the cycle is played we need to listen to the event "there is a new pulse for you".
Davide wrote the matlabMonitor class for that. With it you can record data, make repetitive scan when the cycle is played, read the machine observables and act to the machine actuators in an automatic ways (soft-feedbacks), ideas scounting... In few words, it opens new horizons.

The matlabMonitor: to record data

```
%% Record data for each cycle
signalsToMonitor = {'CL.QDA0415/AQN#value', ...
'CB.BHB1100/Acquisition#currentAverage'};
mySelector = 'SCT.USER.SETUP';

myMonitor = matlabMonitor(mySelector, ...
signalsToMonitor, ...
@(data)disp('New data!'), ... % callback
'two magnets'... % comment
);
myMonitor.saveDataPath = fullfile('./');
myMonitor.saveData = true;
%% start
myMonitor.start
%% stop
myMonitor.stop
```

ALERT: limit the number of monitored parameters not to overload the server CPUs.

The matlabMonitor: to plot

```
%% subscription with plot *
samplerSamples = {'CL.SABPE0125S/Samples#samples'};
mySelector = 'SCT.USER.SETUP';

myMonitor = matlabMonitor(mySelector, ...
samplerSamples, ...
@(data)plot(data.CL.SABPE0125S.Samples.samples.value),
'a Xeneric Sampler'... % comment
);
myMonitor.saveDataPath = fullfile('./');
myMonitor.saveData = false;
%% start
myMonitor.start
%% stop
myMonitor.stop
```

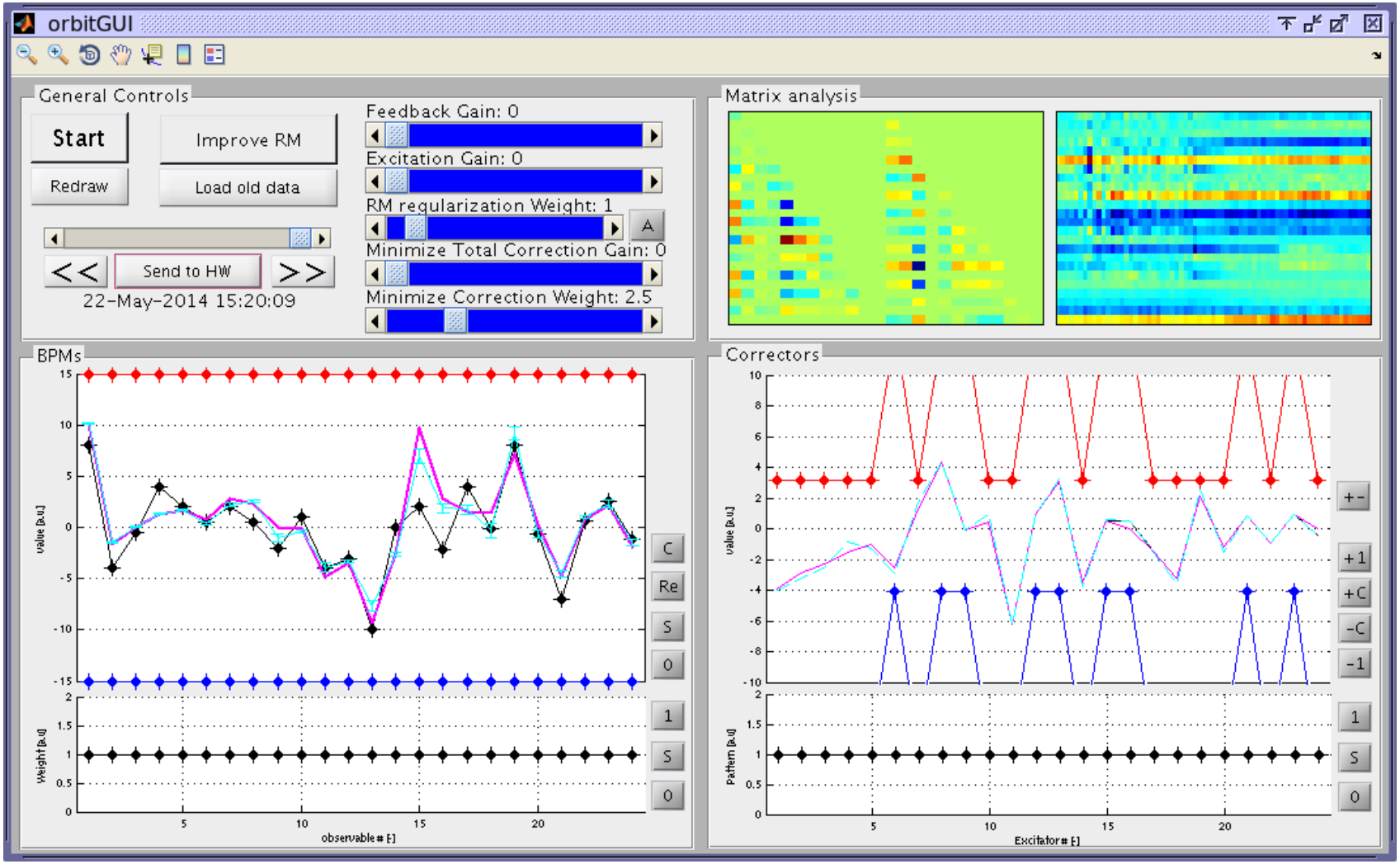
The matlabMonitor: to scan a parameter I

```
signalsToMonitor = {'CP.QFC0210/AQN#value',...
'CP.QDC0215/AQN#value'};
mySelector = 'SCT.USER.SETUP';

myMonitor = matlabMonitor(mySelector, ...
  signalsToMonitor, ...
  @(data)disp('New data!'), ... % callback
  'A scan'... % comment
);
myMonitor.saveDataPath = fullfile('./');
myMonitor.saveData = true;
%%
myScan=scanIt(mySelector,'CP.QDC0205/CCV#value',...
[9.6 9.6 9.6 8.6 8.6 8.6 7.6 9.6],1)
addlistener(myMonitor, 'newBeam', @(h,e)myScan.varyIt)
%% start
myMonitor.start
%% stop
myMonitor.stop
```

The matlabMonitor: to scan a parameter II

```
classdef scanIt<handle

  properties
    myIndex
    myVector
    mySelector
    myParam
  end

  methods

    function obj=scanIt(mySelector,myParam,myArray,myIndex)
      obj.myIndex=myIndex;
      obj.myVector=myArray;
      obj.mySelector=mySelector;
      obj.myParam=myParam;
    end

    function varyIt(obj)
      i= obj.myIndex;
      if i>length(obj.myVector)
        disp('Scan ended!')
      else
        obj.myVector(i)
        obj.myIndex=obj.myIndex+1;
        matlabJapc.staticSetSignal(obj.mySelector,obj.myParam,obj.myVector(i))
      end
    end

  end
end
```

The matlabMonitor: to control




```
%% too long, but we can discuss at the cafeteria!
```

Plan

Conclusions

We showed how to use the JAPC framework to script in MATLAB for

- recording,
- setting,
- automatically controlling the machine parameters
- and ideas scouting out

during the MD. In our experience this tool proved to be extremely useful.

We encourage you to try a similar approach in your favourite scripting language (Mathematica, Jython, Python...) and to share your experience!

Thank you!

Plan

On line demonstration[5]

We will move to the CERN TN and execute/comment the file

```
/afs/cern.ch/work/s/sterbini/public/matlab/forYourMD/
    simpleJob.m
```

In this file you can find some selected and inspirational example to use during your MD.

[5] if you did not fall asleep.