LHC Signal Monitoring Project

*Development of an Embedded Domain Specific Language for Signal Query and Analysis*

Kasper Andersen, Zinur Charifoulline, Per Hagen, Michał Maciejewski, Christoph Obermair, Arjan Verweij, Sivert Sagmo

TE-MPE-MS: Thibaud Buffet, Jean-Christophe Garnier, Tiago Martins Ribeiro, Markus Zerlauth
IT-DB-SAS: Piotr Mrówczynski, Prasanth Kothuri
IT-ST-FDO: Diogo Castro
Developer’s Story

- Requirements
- First Design
- Reality
- Limitations
- Code Quality Decrease
- Second design

LHC Signal Monitoring
Data concerning converters, busbars, current leads, magnets, QPS, cryogenics, etc, as stored in:

1. CALS
2. PM files

CALS and PM will soon be merged into NXCALS.

Often we are interested in parameters that are derived from one or more existing signals, e.g. $R = \frac{V\_MEAS}{I\_MEAS}$, current decay time constant during a FPA, etc.
# Logging Databases - Overview

<table>
<thead>
<tr>
<th></th>
<th>PM</th>
<th>CALS</th>
<th>NXCALS</th>
</tr>
</thead>
<tbody>
<tr>
<td>time definition</td>
<td>event</td>
<td>period of time</td>
<td>period of time</td>
</tr>
<tr>
<td>signal definition</td>
<td>system, source,</td>
<td>signal</td>
<td>system, (device,</td>
</tr>
<tr>
<td></td>
<td>className, signal</td>
<td></td>
<td>property), signal</td>
</tr>
<tr>
<td>return type</td>
<td>json</td>
<td>dictionary of</td>
<td>spark</td>
</tr>
<tr>
<td></td>
<td></td>
<td>arrays</td>
<td>DataFrame</td>
</tr>
<tr>
<td>time unit</td>
<td>ns</td>
<td>us</td>
<td>ns</td>
</tr>
<tr>
<td>API</td>
<td>REST</td>
<td>pytimber</td>
<td>Apache spark</td>
</tr>
</tbody>
</table>
Logging Databases - API

PM – REST API

http://pm-api-pro/v2/pmdatadata/signal?system=FGC&className=51_self_pmd&source=RPTE.UA47.RB.A45&timestampInNanos=142622046952000000&signal=STATUS.I_MEAS

CALS - pytimer

```python
import pytimer
ldb = pytimer.LoggingDB()
```

NXCALS - spark

```python
from cern.nxcals.pyquery.builders import *
import pandas as pd
signal_df = DevicePropertyQuery.builder(spark).system('CMW')
    .startTime(pd.Timestamp('2015-03-13T04:20:59.491000000').to_datetime64())
    .endTime(pd.Timestamp('2015-03-13T04:22:49.491000000'))
    .entity().device('RPTE.UA47.RB.A45').property('SUB')
    .buildDataset().select('acqStamp', 'I_MEAS').dropna().sort('acqStamp').toPandas()
```
Logging Databases – Analytics

<table>
<thead>
<tr>
<th>PM</th>
<th>PM</th>
<th>CALS</th>
<th>CALS</th>
<th>NXCALS</th>
<th>NXCALS</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>signal</td>
<td>signal</td>
<td>feature</td>
<td>signal</td>
<td>feature</td>
</tr>
<tr>
<td>query</td>
<td>query</td>
<td>query</td>
<td>query</td>
<td>query</td>
<td>query</td>
</tr>
<tr>
<td>timing</td>
<td>fast</td>
<td>fast</td>
<td>can be</td>
<td>rather</td>
<td>slow</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>slow</td>
<td>fast</td>
<td></td>
</tr>
<tr>
<td>execution</td>
<td>serial</td>
<td>serial</td>
<td>serial</td>
<td>?</td>
<td>serial</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>parallel</td>
</tr>
<tr>
<td>use</td>
<td>simple</td>
<td>simple</td>
<td>simple</td>
<td>simple</td>
<td>simple</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>hard</td>
</tr>
</tbody>
</table>

→ Need to extend analysis capabilities natively provided by the databases
Unified Database Access

DbSignal
+read()
+write()

PmDbSignal
CalsDbSignal
NxcalsDbSignal
InfluxDbSignal

PM REST API v2&3
pytimber
spark
Influxdb.Data FrameClient

Dataframe
(data structure composed of rows and columns like an Excel file)

ELQA
## DbSignal Classes

### PM

```python
i_meas_df = Signal().read('pm', signal='STATUS.I_MEAS', system='FGC',
                        source='RPTE.UA47.RB.A45', className='51_self_pmd',
                        eventTime='14262204695200000000')
```

### CALS

```python
import pytimber
ldb = pytimber.LoggingDB()

i_meas_df = Signal().read('cals', signal='RPTE.UA47.RB.A45:I_MEAS',
                          t_start='2015-03-13 05:20:59.4910002', duration=[(10, 's'), (100, 's')], ldb=ldb)
```

### NXCALS

```python
i_meas_df = Signal().read('nxcals', signal='I_MEAS', nxcals_system='CMW',
                          nxcals_device='RPTE.UA47.RB.A45', nxcals_property='SUB',
                          t_start='14262204694910000000', duration=[(10, 's'), (100, 's')], spark=spark)
```

How to get the signal name and metadata?
The **Metadata** module contains methods to access various signal and circuit names.

### Circuit Tree

**LHC CIRCUITS**
- **MAIN DIPOLE**
- **MAIN QUADRUPOLE**
- **IT**
- **IPQ**
- **IPD**
- **600 A EE**
- **600 A no EE**
- **600 A no EE crowbar**
- **80-120 A**
- **60 A**

**System**
- CIP
- CRYO
- PIC
- PC
- QDS
- QH
- BUSBAR
- DIODE
- VF
- LEADS_EVEN
- LEADS_ODD
- EE

<table>
<thead>
<tr>
<th>System</th>
<th>Signal Name</th>
<th>Circuit Name</th>
<th>Wildcard</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>I_A</td>
<td>RB.A12</td>
<td>Cell</td>
</tr>
<tr>
<td></td>
<td>I_B</td>
<td>RB.A23</td>
<td>Magnet</td>
</tr>
<tr>
<td></td>
<td>I_EARTH</td>
<td>RB.A34</td>
<td>Crate</td>
</tr>
<tr>
<td></td>
<td>I_EARTH_PCNT</td>
<td>RB.A45</td>
<td>VF</td>
</tr>
<tr>
<td></td>
<td>I_MEAS</td>
<td>RB.A56</td>
<td>Busbar</td>
</tr>
<tr>
<td></td>
<td>I_REF</td>
<td>RB.A67</td>
<td></td>
</tr>
<tr>
<td></td>
<td>V_MEAS</td>
<td>RB.A78</td>
<td></td>
</tr>
<tr>
<td></td>
<td>V_REF</td>
<td>RB.A81</td>
<td></td>
</tr>
</tbody>
</table>

Mapping circuit components
Structure for each circuit is the same
Signal names change over time

→ Circuit topology in one place
→ single analysis for many circuits
→ we need to keep track of the changes
Several design flaws leading to inconsistency and code duplications:

- use of multiple methods, multiple arguments (duplicated across methods)
- multiple local variables (naming consistency across analysis modules)
- order of methods and arguments (with duck typing) not fixed

What if we want to get current for each circuit?
What if we want to get several current signals?
Domain Specific Language

Natural languages have certain structure [1]

English: {Subject}.{Verb}.{Object}: John ate cake

Japanese: {Subject}.{Order}.{Verb}: John-ga keiki-o tabeta
        John cake ate

One can enforce syntactical order in code:
- Domain Specific Language – new language, requires parser
- Embedded Domain Specific Language – extends existing language

We propose a python embedded Domain Specific Language (pyeDSL):

{DB}.{CIRCUIT_TYPE}.{DURATION}.{METADATA}.{QUERY}

+ each parameter defined once (validation of input at each stage)
+ single local variable
+ order of operation is fixed
+ support for vector inputs
+ time-dependent metadata

e.g.

```python
def = QueryBuilder().with_db().with_circuit_type().with_duration().with_metadata()\ .signal_query().dfs[0]
```
Demo 1

Signal query
How does it work?

At each stage only a few methods are available, which update hidden container.

```java
QueryBuilder
+with_db()

query
```

```java
dfs = QueryBuilder();
```

→ Nested Builder Design Pattern
→ Abstract Factory Design Pattern
How does it work?

At each stage only a few methods are available, which update hidden container.

```python
dfs = QueryBuilder().with_db('PM')
```

→ Nested Builder Design Pattern
→ Abstract Factory Design Pattern
How does it work?

At each stage only a few methods are available, which update hidden container.

```python
dfs = QueryBuilder().with_db('PM').with_circuit_type('RB')
```

- **Nested Builder Design Pattern**
- **Abstract Factory Design Pattern**
pyeDSL – Examples (1/2)

PM – event query

```python
QueryBuilder().with_db('PM').with_circuit_type('RB')
    .with_duration(t_start='2015-03-13 05:20:59.4910002', duration=[(100, 's'), (100, 's')])
    .with_metadata(circuit_name='RB.A45', system='PC').event_query().df
```

PM – signal query

```python
QueryBuilder().with_db('PM').with_circuit_type('RB')
    .with_timestamp(1426220469520000000) 
    .with_metadata(circuit_name='RB.A45', system='PC', signal='I_MEAS').signal_query().df[0]
```

CALS – signal query

```python
QueryBuilder().with_db('CALS').with_circuit_type('RB')
    .with_duration(t_start='2015-03-13 05:20:59.4910002', duration=[(100, 's'), (100, 's')])
    .with_metadata(circuit_name='RB.A45', system='PC', signal='I_MEAS').signal_query(dbconnceter=ldb).df[0]
```

→ One can quickly change a database and circuit type, name
→ A sentence created with the language corresponds to database type
pyeDSL – Examples (2/2)

NXCALS – signal query

```python
QueryBuilder().with_db('NXCALS').with_circuit_type('RB')
    .with_duration(t_start='2015-03-13 05:20:59.4910002', duration=[(100, 's'), (100, 's')])
    .with_metadata(circuit_name='RB.A45', system='PC', signal='I_MEAS').signal_query(dbconnector=spark).dfs[0]
```

NXCALS – feature query*

```python
QueryBuilder().with_db('NXCALS').with_circuit_type('RB')
    .with_duration(t_start='2015-03-13 05:20:59.4910002', duration=[(100, 's'), (100, 's')])
    .with_metadata(circuit_name='RB.A45', system='PC', signal='I_MEAS')
    .feature_query(features=['min', 'max', 'std', 'mean'], dbconnector=spark).dfs[0]
```

*work in progress
pyeDSL – Polymorphism

Multiple circuit names

```python
QueryBuilder().with_db('NXCALS').with_circuit_type('RB')
    .with_duration(t_start='2015-03-13 05:20:59.4910002', duration=[(100, 's'), (100, 's')])
    .with_metadata(circuit_name=['RB.A12', 'RB.A45'], system='PC', signal='I_MEAS')
    .signal_query(dbconnector=spark).dfs[0]
```

Multiple system names

```python
QueryBuilder().with_db('PM').with_circuit_type('RB').with_timestamp(15446221495980000)
    .with_metadata(circuit_name='RB.A12', system=['LEADS_EVEN', 'LEADS_ODD'], signal='U_HTS')
    .signal_query().dfs
```

Multiple signal names

```python
QueryBuilder().with_db('PM').with_circuit_type('RQ').with_timestamp(15446221495980000)
    .with_metadata(circuit_name='RQD.A12', system='QDS', signal=['U_1_EXT', 'U_2_EXT'],
                  source='16L2', wildcard={'CELL': '16L2'})
    .signal_query().dfs
```

Wildcard

```python
QueryBuilder().with_db('CALS').with_circuit_type('RQ')
    .with_duration(t_start=int(1544622149613000000), duration=[(50, 's'), (150, 's')])
    .with_metadata(circuit_name='RQD.A12', system='DIODE_RQD', signal='U_DIODE_RQD', wildcard={'MAGNET': '.*'})
    .signal_query(dbconnector=ldb)
```

→ Internal handling of for loops – reduced amount of code in analysis
Adding Adjectives

Once a signal is queried, we can perform some operations on each of them. In this case, the order of operations does not matter (but can be checked)

\[
\{DB\}.\{CIRCUIT\_TYPE\}.\{DURATION\}.\{METADATA\}.\{QUERY\}.\{PRE-PROCESSING\}
\]

- synchronize_time()
- convert_index_to_sec()
- filter()
- remove_initial_offset()

e.g.

df = QueryBuilder().with_db().with_circuit_type().with_duration().with_metadata() \[\text{signal_query()}.\text{synchronize_time()}.\text{convert_index_to_sec()}.\text{dfs}[0]\]
Demo 2

Signal query and processing
Hardware Commissioning procedures check ranges of certain signals

<table>
<thead>
<tr>
<th>Responsible</th>
<th>Type of analysis</th>
<th>Criterion</th>
</tr>
</thead>
</table>
| MP3         | Automatic analysis on earth current  | I_EARTH_PLI3_A5 < I_EARTH_MAX  
 |             | and error current                    | I_ERR_PLI3_A5 < I_ERR_MAX                                                 |
|             | Splice signals                       | From board A and board B separately                                      |
|             |                                       | R_bus_max <3 nOhm                                                         |
|             |                                       | Individual R_splice_max<0.5nOhm                                          |
|             |                                       | R_mag<50 nOhm                                                             |
|             | Current lead                         | 46 < TT891A < 54K                                                        |
|             | Abs(U_RES)< 40mV and no drift        |                                                                           |
|             | Abs(U_HTS) < 0.5mV                   |                                                                           |
|             | Calorimetric (if done)               | dT/dt (TT821)< 5 mK/hr                                                   |

AssertionBuilder class performs signal assertions.
Demo 3

Signal assertion
Signal analysis (e.g., quench heater discharges) requires extraction of certain characteristic features.

FeatureBuilder performs feature engineering in a generic way.
Demo 4

Signal feature extraction
With a solid signal query and processing API we can advance faster with developing HWC and monitoring notebooks and extend to other circuits.
### Signal Monitoring Workflow

Acquisition → Exploration → Modelling → Monitoring

<table>
<thead>
<tr>
<th>Component</th>
<th>RB</th>
<th>RQ</th>
<th>RQX</th>
<th>IPQ/D</th>
</tr>
</thead>
<tbody>
<tr>
<td>QH</td>
<td>RB</td>
<td>RQ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COLDBB</td>
<td>RB</td>
<td>RQ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC</td>
<td>RB</td>
<td>RQ</td>
<td></td>
<td>600A</td>
</tr>
<tr>
<td>DIODE</td>
<td>RB</td>
<td>RQ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GND</td>
<td>RB</td>
<td>RQ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EE</td>
<td>RB</td>
<td>RQ</td>
<td></td>
<td>600A</td>
</tr>
<tr>
<td>MAGNET</td>
<td>RB</td>
<td>RQ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DFB</td>
<td>RB</td>
<td>RQ</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### HWC and Quench Analysis

Run 3 - Monitoring
Analysis - Modelling

1. With historical data we derive expected behavior and trends.
2. With on-line data we compare behavior with others.

digital-twin  trends  intra-component  cross-population
Modelling Methods

Physical Models
+ Provide access to non-measurable states
+ High-numerical precision
- As accurate as available parameters, measurements

Data-Driven Models

Threshold-Based
+ Embed expert knowledge on parameters
+ Give a clear answer
- Require adjustments

Probabilistic
+ Distribution of parameters (features)
- Do not give a clear answer
- As good as data

Machine Learning
+ Encode non-obvious relations
+ Find patterns, correlations
+ Encode expert knowledge
- Do not give a clear answer
- As good as input data

Hybrid Methods
+ Mixture of deterministic (limited) and non-deterministic (flexible)
- More complicated analysis
Automatic execution of monitoring application depends on the operation state:
- triggered by PM events (PC, QH, MAGNET)
- triggered by change in the beam mode (GND, COLDBB)
- in regular intervals, e.g. every hour (DFB)
Automatic Execution

Manual execution of HWC notebooks

- HWC Sequencer
  - queue
  - Manual notebook execution
  - NXCALS Cluster
  - *.csv
  - EOS
  - *.html

Automatic execution of long-running historical analyses*

- NXCALS Beam mode
  - Queue of timestamps to process
  - NXCALS Cluster
  - NXCALS HDFS
  - *.csv

→ Need for analysis trigger and analysis supervision (Apache AirFlow)

Courtesy: P. Mrówczyński
https://gitlab.cern.ch/db/swan-spark-notebooks
https://gitlab.cern.ch/LHCDATA/lhc-sm-apps/merge_requests/1
Summary

1. Introduction of pyeDSL unifies database query and simplifies code
2. The signal processing and feature engineering is provided but limited
3. The pyeDSL introduces clear structure by enforcing order of operations
4. The directions for extension are clearly identified
5. The development time and maintenance effort have reduced considerably
We rely on industry-standard tools for the development automation. Majority (except for PyCharm IDE and Python Package Index) services are supported by CERN IT.
API
In order to use the project the API has to be installed in SWAN

```
 pip install --user lhcsmapi
```

Check the latest version at [https://pypi.org/project/lhcsmapi/](https://pypi.org/project/lhcsmapi/)
The documentation for the API is stored at [http://cern.ch/lhc-sm-api.](http://cern.ch/lhc-sm-api)
The repository of the API is available at a GitLab [http://gitlab.cern.ch/lhcdata/lhc-sm-api](http://gitlab.cern.ch/lhcdata/lhc-sm-api)

Applications
The released use cases are available at the SWAN gallery
The beta versions of the use cases are stored at [http://gitlab.cern.ch/lhcdata/lhc-sm-apps](http://gitlab.cern.ch/lhcdata/lhc-sm-apps)

Project website: [https://twiki.cern.ch/twiki/bin/view/TEMPEPE/Signal_Monitoring](https://twiki.cern.ch/twiki/bin/view/TEMPEPE/Signal_Monitoring)