A USERS VIEW ON EXPLOITING THE CONTROL SYSTEM IN MDs

G. Iadarola*, R. De Maria, V. Baggiolini, J. Gonzalez Cobas, T. Levens, J. Wozniak,
CERN, Geneva, Switzerland

Abstract

After almost 10 years of beam operation, machine studies at the LHC are becoming more and more advanced and complex. This translates into a need for advanced software tools to acquire, log, manipulate, analyze machine data as well as to control the installed equipment. The different teams, often in collaboration, have developed many innovative solutions for this purpose. Notably several of these solutions have been developed using the Python language and by now it is possible to interact with the Logging System, with the LSA database, and directly with LHC equipment using Python. Although these solutions were not officially supported up to now, the Controls Group is presently working to improve the available infrastructure and support in this direction. This is an important step to allow tools developed for machine studies and hardware commissioning to enrich the diagnostics toolbox available for the daily operation of the LHC.

INTRODUCTION

Machine Development (MD) users and equipment experts typically interact with the control system of the Large Hadron Collider (LHC) to perform two kinds of activities, which will be discussed in detail in the coming sections:

1. Extract data from the CERN Accelerator Logging System (CALS) [1];

2. Interact directly with LHC equipment.

The standard way of performing these tasks is to use the dedicated applications, mostly developed in Java, provided by the Controls Group (BE-CO), the Operations Group (BE-OP) or by other teams. While covering the most common and standard activities, this approach lacks the flexibility that is often required when performing machine studies or commissioning new equipment. The supported way of interacting with the control system with full flexibility is to use the Java Application Program Interface (API) provided by BE-CO. Unfortunately this is impractical for many users as Java is not part of the background of the "average physicist" and presents a quite long learning curve.

Over the years the different teams came up with several creative and unconventional solutions to bridge this gap. Here in particular we will summarize approaches and tools that are shared by relatively large groups of users and on which development efforts are being invested.

* Giovanni.Iadarola@cern.ch

The "Python trend"

A general trend that can be clearly identified is that many of these solutions are developed using the Python language [2,3]. This has been chosen for several recent projects, while many users who had previously adopted different solutions (e.g. Matlab, LabVIEW) are now migrating their tools to Python. Python is an open-source general-purpose language and is a very popular choice for scientific computing, already widely used by the LHC MD community for data analysis, calculations and numerical simulations. The main reasons that make Python very suitable for these applications are:

- The learning curve is very fast;
- The syntax is simple, flexible, and concise. This renders the development generally faster than with other languages, and makes Python very suitable for quick prototyping and testing of new ideas;
- The language is very suited for interactive development;
- It is used and supported by a large community (most questions can be answered with a quick internet search);
- It is equipped with a solid and complete set of tools for numerical analysis and plotting, which are free and open-source (e.g. numpy, scipy, matplolib, pandas).

EXTRACTION AND ANALYSIS OF LOGGED DATA

When analyzing machine data, from an MD session or from operation, the "classical paradigm" consists in two separate stages:

1. Extract the data from Logging System [1] and store it on the local drive (this is typically done using the TIMBER application provided by BE-CO);

2. Perform the data analysis (e.g. data manipulation, correlations, plotting, etc.) using one's favorite environment, e.g. MATLAB, Mathematica, Python, Root, Excel.

While being perfectly suited for a large set of studies, this approach can be very inefficient for situations in which the data extraction is strongly entangled with the data analysis, for example when time-intervals for which further data needs to be downloaded are only identified iteratively during the data processing.

PyTimber

In order to be able to effectively interact with the logging system in a programmatic way and within an environment

Example: download and plot LHC luminosities

```
import pytimber, pylab
ldb = pytimber.LoggingDB()

data = pytimber.DataQuery(ldb,
     ["ATLAS:LUMI_TOT_INST","CMS:LUMI_TOT_INST"],
     "2017-11-02 14:00:00", “2017-11-0308:00:00”)

data.plot_2d()

pylab.show()
```

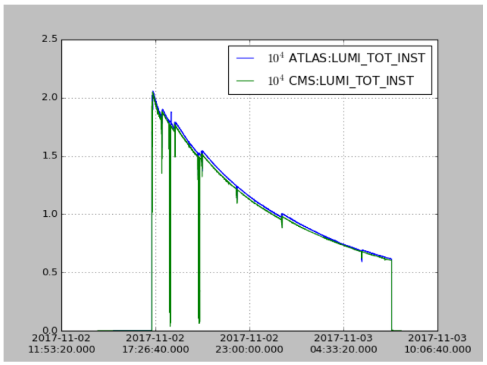

Figure 1: Example showing how to download and plot data from the CALS system using PyTimber.

that is also suitable for data processing and analysis, a group of MD users has developed a Python module called PyTimber, which allows accessing the logging system through Python [2,4]. PyTimber is a Python wrapper of the CALS Java API, which is built exploiting the Jpype module [5] to expose Java objects in Python. In this way most of the “Java technicalities” are hidden providing a user-friendly but scriptable interface (see Fig. 1). Started at the beginning of Run 2 as a personal project, it is by now adopted by hundreds of users within CERN, and has been further evolved by the user community through collaborative development based on GitHub.

PyTimber has also been made available within the SWAN environment [6], developed by the LHC experiments to perform interactive data analysis based on cloud computing resources. One of the advantages of this approach is that the user can drive the analysis using his web browser, without having the need for installing any dedicated software on his local machine.

The future of the Logging System: NXCALS

The data-flow to the logging system has significantly increased over the years, as shown in Fig. 2, and is rapidly approaching the maximum performance by the present implementation [7]. The BE-CO group is presently working on the full renovation of the Logging System, developing a new system, called NXCALS, in order to better fulfill these growing needs [8,9].

NXCALS is based on open-source technology and in particular on Spark [10] and Hadoop [11], which are leading players for “Big-Data” applications. The new system is expected to show a much better horizontal scalability, allowing to keep a good performance in spite of the very large amount of stored data.

The present Java API will be maintained, which implies that the present applications including Timber and PyTimber will still work. Other ways of interacting with the system will also become available (Python, jupyter, spark, and PySpark on Swan). In particular, more advanced analysis will be possible using the Big-Data approach, in which the user does not anymore download the data to his local machine

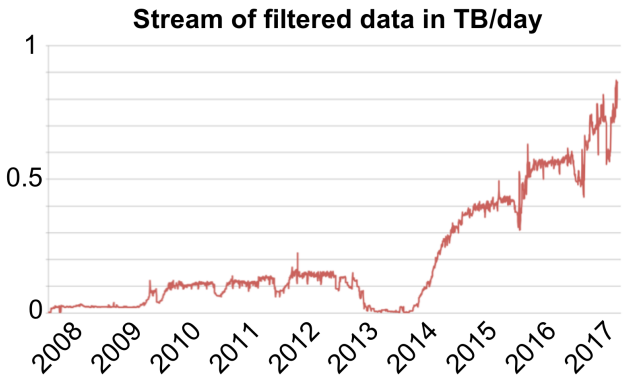

Figure 2: Evolution of the data stream to the CALS system.

but sends to the system data processing code to be executed directly on distributed storage and computing resources.

The data migration from the present system to NXCALS is presently ongoing, together with the analysis of all the present use cases, in order to guarantee that no loss of functionality will take place when switching to the new system.

INTERACTION WITH LHC EQUIPMENT

Machine studies often require interacting with LHC equipment in particular when using diagnostics from beam instrumentation, RF, and other equipment. Often, this does not consist in a passive observation of published data but requires sending information to the equipment, like commands, settings, or triggers. The conventional way of performing these tasks is to use the available applications (mostly written in Java), but this does not cover many cases of interest, notably the commissioning of new hardware and experiments involving “unconventional usage” of existing devices.

Several approaches have been adopted by the different teams to overcome this limitation. Some can afford to develop ad-hoc Java applications, but this requires resources and expertise that are often not compatible with small projects. An alternative is to use a scripting language to interact with the LHC controls stack as shown in Fig. 3.

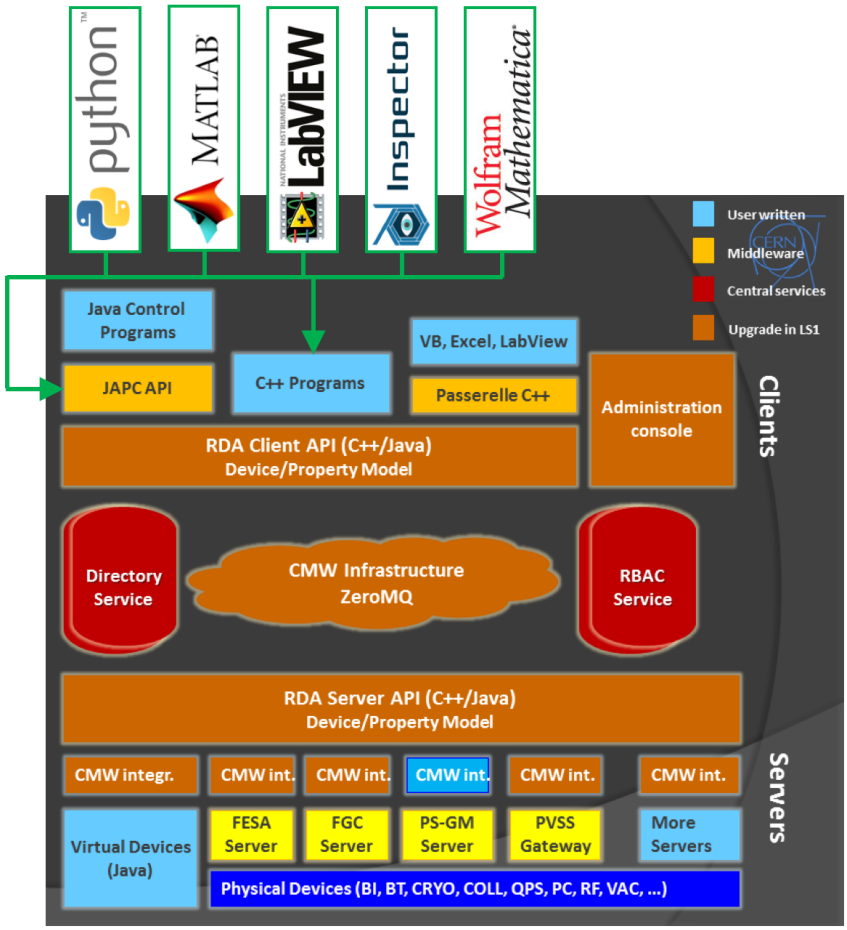

Figure 3: The accelerator controls stack, with (indicated in green) the most common tools used to interact with the system during commissioning and machine studies.

For this purpose, tools have been developed that hide some of the complexity and allow for a simpler development (see [12] for a complete overview). Here we will briefly mention three popular choices: PyJAPC, Inspector and PjLSA.

PyJAPC

PyJAPC is a simplified Python interface to accelerator hardware, allowing for example access to the Front-End Software Architecture (FESA) [2,13,14]. Its implementation makes use of JPype to call functions of the "Java API for Parameter Control" (JAPC) directly from Python. The basic functionalities can be used without knowing anything about the underlying JAPC API, as illustrated by the example in Fig. 4. Furthermore, more advanced features can still be accessed by manually calling the relevant JAPC functions from Python. Today PyJAPC is under the hood of many interfaces that are routinely used in the control room, like the ADT Diagnostic Panel, the RF Stable Phase Measurements, the HeadTail Viewer, the BSRT vistar.

Inspector

An alternative solution is provided by the Inspector tool [15], an environment developed and maintained at CERN, which allows using graphical programming (the same approach used in LabVIEW) to build control applications. As for PyJAPC, its implementation is based on the Java API (JAPC). Inspector allows for very fast development of Graphical User Interfaces (GUIs) and displays and it is used for several expert interfaces and MD tools, for example expert interfaces to the LHC transverse damper and to the LHC main RF system, the configuration panel of the LHC Instability Trigger Network, the Beam Transverse Function (BTF) interface (shown in Fig. 5).

PjLSA

The LSA (LHC Software Architecture) database contains important information (e.g. functions, trim history, knob definitions). Conventionally this can be accessed, only within the Technical Network (TN), using the operational applications (e.g. LHC Trim) or through the available Java API. Following the method used to develop PyTIMBER and Py-

Example: plot BBQ spectrum

```
# instantiate pyjapc object
import pyjapc
japc = pyjapc.PyJapc(selector="LHC.USER.ALL",
          incaAcceleratorName="LHC", noSet=True)

# RBAC login
japc.rbacLogin(loginDialog=True)

# Get vector data from LHC BBQ
v = japc.getParam(
 "LHC.BQ.ONDEMAND.B1/SummaryMeasurement#averageMagnitudeH")

# plot
import numpy as np; import pylab as pl
xVect = np.linspace(0, 11e3/2, len(v), endpoint=False )
pl.plot(xVect/1e3, v, label=par)

# RBAC Logout
japc.rbacLogout()
```

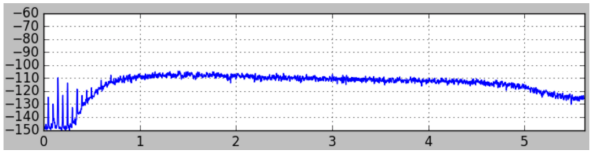

Figure 4: Example showing how to interact with LHC equipment using the PyJAPC tool.

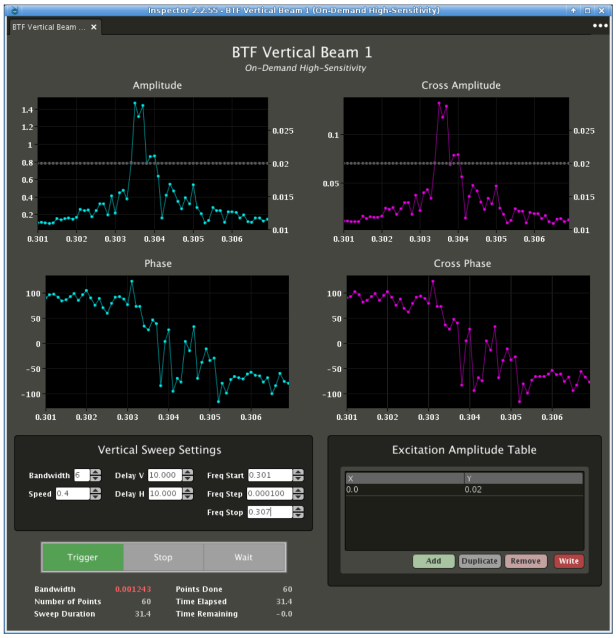

Figure 5: The BTF interface developed using the Inspector environment.

JAPC, a Python wrapper of the LSA Java API has been recently setup. This new tool has been named PjLSA [16].

By design the access is limited to the read functions only, preventing therefore any change of the LHC configuration driven by PjLSA. A very useful feature for MD and operation follow-up is the fact that PjLSA can be accessed not only within the TN but also through the General Purpose Network (GPN) within CERN, thanks to an LSA gateway server provided by BE-CO. Furthermore, PjLSA can be easily combined with PyTIMBER and PyJAPC obtaining a complete scriptable toolbox for machine studies.

In the LHC injectors this approach has been pushed even further. Similar tools are used not only to read from LSA but also to automatically send settings and perform automatic scans. The potential of opening this possibility also in the LHC will be investigated in the future, together with the necessary measures to prevent wrong manipulations which could affect machine protection and availability.

FROM MD AND EXPERT TOOLS TO MORE GENERAL USAGE

Many of the tools developed for Machine Studies and commissioning of new equipment have potential to become useful in the daily LHC operation [17]. An example in this direction was the integration of the coupling measurement tool, developed in collaboration by optics and transverse feedback experts, within the operational Accelerator Cockpit application [18]. The general perception is that, for different reasons (including differences in the employed development language, environment, work-flow), the potential of this “tool-transfer” process is not fully exploited. Some concrete steps that could be made in order to facilitate it can be summarized as follows:

From the side of the BE-CO and BE-OP groups:

- Provide guidelines and feedback on how to develop and maintain (semi-)operational tools within the LHC software ecosystem;
- Enlarge the set of allowed/supported programming languages and development environments, for example Python (see next subsection);
- Provide an agile way to develop Graphical User Interfaces (GUIs). Inspector provides an interesting example in this sense.

From the side of the MD and equipment groups:

- Write code respecting basic programming good practices, in order to ease further development, maintenance and integration with other tools;
- Understand and respect the guidelines defined together with BE-CO and BE-OP;
- Participate in the software maintenance and support, or guarantee a proper hand-over of the responsibility to BE-OP and/or BE-CO.

Evolution of Python support

Until recently Python was not officially supported by BE-CO for software development within the LHC controls ecosystem. In particular there was no official Python installation available and supported in the Technical Network.

Some "unofficial" Python infrastructure was built by the users themselves. In particular several Python installations are available in AFS and NSF public folders (e.g. the "BI

installation" [19]), which were prepared by the users for their own needs, of course without any commitment for general maintenance and support.

In order to efficiently handle Java dependencies within Python tools based on Java APIs (as PyTimber, PyJAPC, PjLSA) a dedicated module called "CommonBuild Dependency Manager" has been developed, which automatically identifies Java dependencies, downloads the required jars, and sets up the required Java Virtual Machine (JVM) within Python [20].

Interest and support in Python applications has been rapidly growing within BE-CO in the latest period [2]. For example, BE-CO was involved in the development of PjLSA and the new CommonBuild (CBNG) [21] supports Python usage through a dedicated web-server [22]. Moreover, a dedicated Python Focus Group has been put in place to provide a common forum for the development of Python tools with the LHC controls environment [23] and to setup the necessary infrastructure. The first concrete steps will be:

1. The setup of Python installations supported by BE-CO (based on those available within the LHC Computing Grid) within the TN;

2. The setup of a basic environment for the development, deployment and maintenance of Python applications.

SUMMARY AND CONCLUSIONS

The MD community has developed several solutions to exploit the potential of the LHC hardware and control system in a flexible and experimental way. In particular, we start having a complete and powerful Python toolbox (PyTimber, PyJPAC, PjLSA, PyLogbook) developed collaboratively across different teams, which allows interacting with logging, LSA, and LHC hardware as well as performing advanced data analysis, all in the same environment.

Up to now the usage of Python tools within the LHC control system was not officially supported by BE-CO. Recently a dedicated Python focus group has been put in place to provide a common forum and deploy a supported Python environment in the TN.

The logging system is undergoing a major upgrade which will allow better coping with the increasing volume of data and using the Big-Data philosophy for the analysis of the stored data.

The new infrastructure that is being put in place should facilitate the migration of tools developed for equipment commissioning and Machine Studies into the LHC operational life, provided that guidelines and good practices for this process are defined and followed.

ACKNOWLEDGMENTS

The authors would like to thank G. Arduini, T. Argyropoulos, H. Bartosik, K. Fuchsberger, M. Hostettler, E. Metral, T. Persson, R. Tomas, M. Solfaroli Camillocci, G. Sterbini, J. Wenninger for the important input provided to this contribution.

REFERENCES

[1] The CERN Acclerator Logging System (CALS), https://be-dep-co.web.cern.ch/content/cals.

[2] D. Cobas, R. De Maria, A. Hushauer, T. Levens, P. Mendez Lorenzo, D. Piparo, N. Tsvetkov, "Python tools for machine data analysis and equipment control", BE Seminar, CERN, 21 Apr 2017.

[3] Python official website, https://www.python.org.

[4] PyTimber repository on GitHub, https://github.com/rdemaria/pytimber.

[5] Homepage of the jpype project, http://jpype.sourceforge.net.

[6] The CERN-SWAN data analysis platform, https://swan.web.cern.ch.

[7] J. Wozniak, "Evolution of the logging system", presentation at the LMC meeting, CERN, 6 July 2016.

[8] NXCALS home page, https://wikis.cern.ch/display/NXCALS/NXCALS+Home.

[9] NXCALS git repository, https://gitlab.cern.ch/acc-logging-team/nxcals.

[10] The Apache Spark computing engine, https://spark.apache.org.

[11] The Apache Hadoop software library, http://hadoop.apache.org.

[12] Scripting tools for equipment control, https://wikis.cern.ch/display/ST/Libraries+Available.

[13] PyJAPC documentation pages, http://bewww/~bdisoft/pyjapc/index.html#.

[14] PyJAPC repository on GitLab, https://gitlab.cern.ch/scripting-tools/pyjapc.

[15] Inspector wiki pages, https://wikis.cern.ch/display/INSP/Inspector+Home.

[16] PjLSA repository on GitHub, https://github.com/rdemaria/pjlsa.

[17] D. Jacquet, "Evolving expert tools into operational software?", Proc. of the 7th Evian Workshop, Evian 13-15 Dec 2016, CERN-ACC-2017-094.

[18] K. Fuchsberger, T. Persson, D. Valuch, "Linear coupling tools", presentation at the LMC meeting, CERN, 8 Nov 2017.

[19] Th BI Python installation on NFS, https://wikis.cern.ch/display/ST/BI+Python+installation+on+NFS.

[20] Common-Build Dependency Manager repository on GitLab, https://gitlab.cern.ch/scripting-tools/cmmnbuild-dep-manager.

[21] CBNG documentation, https://wikis.cern.ch/display/DVTLS/CBNG#CBNG-Wheretostart.

[22] The CBNG Web Service, https://wikis.cern.ch/display/DVTLS/CBNG+Web+service

[23] Homepage of the Python Focus Group, https://wikis.cern.ch/display/PyFG/Python+Focus+Group+Home.