ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-WEPV018

THE LINAC4 SOURCE AUTOPILOT

M. Hrabia, M. Peryt, R. Scrivens, CERN, Geneva, Switzerland
D. Noll, European Spallation Source, Lund, Sweden

Abstract

The Linac4 source is a 2MHz, RF driven, H- ion source, using caesium injection to enhance H- production and lower the electron to H- ratio. The source operates with 800µs long pulses at 1.2 second intervals. The stability of the beam intensity from the source requires adjustment of parameters like RF power used for plasma heating.

The Linac4 Source Autopilot improves the stability and uptime of the source, by using high-level automation to monitor and control Device parameters of the source, in a time range of minutes to days.

This paper describes the Autopilot Framework, which incorporates standard CERN accelerator Controls infrastructure, and enables users to add domain specific code for their needs. User code typically runs continuously, adapting Device settings based on acquisitions. Typical use cases are slow feedback systems and procedure automation (e.g. resetting equipment).

The novelty of the Autopilot is the successful integration of the Controls software based predominantly on Java technologies, with domain specific user code written in Python. This allows users to leverage a robust Controls infrastructure, with minimal effort, using the agility of the Python ecosystem.

INTRODUCTION

The CERN Linac4 ion source is a 2MHz RF driven H^- ion source, using caesium injection to enhance H^- production and lower the electron to H^- ratio. The source operates for Linac4 with 800µs long pulses at 1.2 second intervals.

The stability of the H^- beam intensity from the source over the period of minutes to days requires adjustment of parameters like the RF power used for plasma heating. Controlling the source on this timescale is the objective of the Autopilot. It is not conceived to work from pulse to pulse, or within a pulse, which would require dedicated systems at the front-end computer level.

The Autopilot Framework was developed in 2019 and used successfully to automatically tune the source during the test runs for Linac4. It is currently operating 24/7 helping the operations team to run the linear accelerator. Another instance has been successfully deployed in the CERN ion linac, Linac3. It replaces a highly specific version developed in 2017 [1], which lacked flexibility.

FRAMEWORK

The Autopilot Framework, henceforth referred to as the framework, is a set of services and tools (see Fig. 1.) that allow the users to deploy and execute the algorithmic tasks in a self-service manner, where the framework provides the necessary tooling and infrastructure. Typical use cases are slow feedback systems and automated procedures (like resetting and restarting equipment that has stopped due to a fault state).

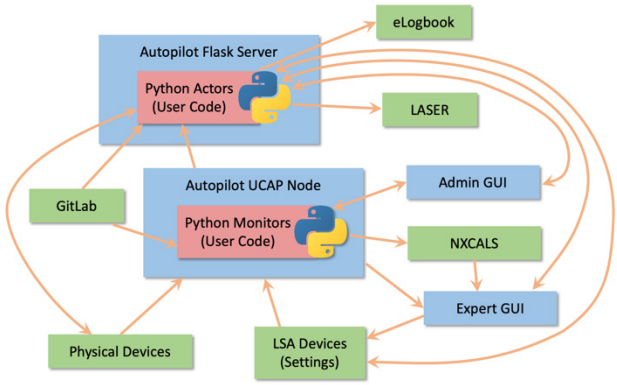

Figure1: Schematic diagram of the Autopilot Framework. Blue boxes signify the Autopilot framework components.

The framework allows the users to subscribe to the Control parameters of their choice, process the received values, and publish the output as properties of Virtual Devices, i.e. Control Devices that are implemented exclusively in software, with no hardware component. Those Virtual Devices can in turn be used like any other Control Device, in particular they can provide inputs to the user-supplied regulation algorithms (subsequently referred to as actors) that interact with the Linac4 H^- ion source.

A particularity of the framework is that although it is based on the accelerator Controls software stack, that is predominantly developed in Java, it allows the user code to be written in Python. Another characteristic of the framework is that it leverages the well-established services and components that form the accelerator Controls, in particular the Controls Configuration Service (CCS) [2], Controls Middleware (CMW) [3], Role-Based Access System (RBAC) [4], the Unified Controls Acquisition and Processing (UCAP) framework [5], LSA (Accelerator Settings Management) [6], LASER (Accelerator Alarms Service) [7], and NXCALS (Accelerator Logging Service) [8], thus reducing the need for developing custom components.

At the core of the framework lies a UCAP node that subscribes to physical and virtual Devices according to the recipes provided by the users. The recipes also contain the triggering conditions. When those conditions occur, the events containing the accumulated data are constructed and forwarded to the user-provided transformation routines, known as “Monitors”. The Monitors calculate the output values and give them back to UCAP, for publishing as virtual Device Properties through controls middleware.

Along with the UCAP recipes, and the Monitors, the Flask server also manages the user-provided control tasks called “Actors”. These are basically Python scripts whose job is to act upon the updates received from the Monitors

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-WEPV018

and perform the necessary actions to keep the parameters of the Linac4 source within the requested limits. The benefit of the Actors using Monitors, is that the UCAP system takes care of synchronizing multiple data inputs, allowing the Actor to receive all required data in a single subscription.

Server

The server is a REST service powered by the Flask micro-framework combined with Gunicorn as HTTP Server. The REST communication is hidden behind a simple web interface, which serves as a bridge between the user and the framework itself. The service was built with the use of Acc-Py, a CERN Accelerator Controls Python distribution.

The web interface consists of three subpages, each dedicated to a separate part of the Autopilot Tasks: Monitors, Actors and UCAP configuration. Each of them provides a status view of uploaded user scripts, as well as interactive tools to manage the content of the service, such as adding and removing script files or starting and stopping Task execution.

To make sure that the communication is secure, an HTTPS connection to the server has been set up, with the help of tools provided by the CERN Certification Authority, giving an encrypted communication channel between the service and the user.

To restrict access to authorised users, Role Based Access (RBAC) is used for user authentication. On top of that, to distinguish between the different types of users and restrict access even further, three different internal roles (Guest, Operator, SuperUser) have been defined allowing for the fine-tuned levels of control over the framework.

The web interface also provides a set of features to help the users service their Tasks in case they do not behave as expected, such as:

- Checking python script correctness at start-up time – when the user tries to start a new python script, the check for basic code errors, like wrong indentation level, is performed. Since obvious mistakes like this would prevent the script from running correctly, an error message specifying the reason of the failure is shown to the user, and the new process is never started.
- Logging mechanisms – the user is provided with a predefined logger, which can be used inside user's script to log any useful information during code execution. User logs are stored on the server and are easily accessible through the web-interface.
- Automatically catching runtime exceptions – execution errors which may happen at runtime and are not handled by user code are captured by the framework and put into the user logs for post-mortem analysis.
- Subscription status preview – for each Monitor Task, the list of all its Data Sources and their connection statuses is provided. This allows the user to see in real time if the data from each Data Source is available. For quick debugging of broken subscriptions, a remote 'test get' feature is provided to query the input devices.
- File peeking – the user can visualise the task code directly in the web interface.
- Test data generation – before the Monitor script will be uploaded to the server it has to be written offline. Python at its core is a language where it is easy to make coding errors, so we give the possibility to record a specified amount of input data from UCAP in the form of JSON files, that can be downloaded by the users and fed to their algorithms offline, making it easier to find some obvious problems upfront, before the final version will be uploaded to the server.

Source Code Management – GitLab

To manage the state and content of user script files, the server is connected to a dedicated git repository hosted on CERN's GitLab service. The repository contains a configuration file listing explicitly all Actors, Monitors and UCAP configuration files. This allows the server to easily recognize the expected role of a given file and handle it accordingly.

By using GitLab, the history of changes is available for all files, providing a clear trace of how content was changed, by whom and when. It also gives a simple way to roll back to a previous version of the file in case the new version is not behaving as expected.

The server uses the GitLab REST API to compare files from the repository with those currently hosted on the server as well as download them if necessary.

From the server perspective, updating any script file is as simple as two clicks on the web panel. First click to 'check' for any changes, where the server compares commit IDs of the files, and the second click to 'synchronize' the file, meaning downloading the latest version of the file from the GitLab repository if it is newer than the version currently on the server.

For safety reasons the script files can only be synchronized if the corresponding Autopilot Task is not in a 'running' state. However, the version check can be done at any time, as it does not impact the content of the server.

When any script file has been recognized as outdated – a notification label is shown next to the file name on the web panel, making it easily noticeable. As an added value, the label itself is a hyperlink to the GitLab webpage commit details, which highlights changes in the script file content between versions, thus allowing for a quick verification before deciding to synchronize script files to the server.

The server is also capable of being notified by a GitLab web hook automatically, after any file has been changed, sparing the user the necessity of invoking the 'check' operation manually.

No automation is foreseen for the non-interactive synchronization of script files. Since those files are actually running on the server, it is critical for stable operations, to only update runtime content when there is a safe operational window for doing that. This update can only be performed by super-users, as defined in the preceding section.

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-WEPV018

UCAP

The Unified Controls Acquisition and Processing (UCAP) project aims to provide "a generic, self-service, controls data processing platform".

In simpler terms, UCAP is a platform that allows the creation of a standard Virtual Device, which receives data from a predefined list of signals, gives that data to the user-written or configured code (known as a Transformation), which can in turn perform any kind of calculations and processing, and publishes the result back to the Controls system using a standard middleware protocol. State is retained between calls to the user transformation, so that it is possible to apply transformations over multiple beam cycles (for example averaging a value over some time window).

At the core of the Linac4 Source Autopilot, Monitors are in fact UCAP Virtual Devices, publishing transformed data that is in-turn used by the Actors to interact with the Source. Corresponding data is stored in the logging database (NXCALS) and can be used later for purposes such as diagnostics.

Monitors

As explained above, Monitors are Virtual Devices running on a UCAP node. The Autopilot server simply acts as a man-in-the-middle between the user and the UCAP framework, meaning the user uploads UCAP configuration and transformation code files using the aforementioned web panel, and the Autopilot servers takes care of turning that into a request understandable by the UCAP framework.

It is the responsibility of the UCAP framework to create a new process to receive, transform and publish data. The Autopilot server does nothing more than ask UCAP to add/remove or start/stop the Virtual Device in question.

The data transformation code is written in Python using the provided ucap-python package, which contains all interfaces representing the data model of UCAP. The usage of this package is mandatory for every Autopilot Monitor script.

By design, Monitors only read and transform the data published by other Devices and do not interact themselves with the Control system.

Monitors can also receive configuration parameters for their execution by including LSA virtual settings values in their data input. For example, a minimum and maximum needed to perform a value validation can be created as a LSA virtual setting, and input to an Autopilot Monitor along with the real Device acquisition for comparison. In this way the LSA parameters allow many different ways to incorporate settings into Autopilot applications, at the same time keeping a history of changes to these settings inside the LSA settings history repository.

Actors

Actors are Tasks that continuously receive data produced by Monitors, and possibly some other sources such as LSA settings (both from real Devices, and Virtual Devices). Based on these inputs they can decide to invoke a particular action if necessary. Each Actor Task runs as a separate process.

Actor Tasks can use a special API to notify the Autopilot server about certain events by invoking specific actions, which are typically reflected on the web panels (and through other services like the electronic Logbook (eLogbook) and LASER alarms system). They give useful real-time feedback of a running Actor's behaviour. Invokable actions include:

- heartbeat – an Actor script can call this method to announce that it is alive. It is usually invoked by the user whenever new data is received by the Actor.
- set_last_acted – an Actor script can notify that it is performing an action it considers as an 'act' method (like applying a new hardware setting) and provide a short message describing it.
- set_last_error / clear_last_error – an Actor script can notify the server that it recognized a situation considered as an error, for example that it received corrupted data.
- abort_task – an Actor script can ask the server to cleanly abort the Task execution, for example where a set of conditions have no path defined, and the situation is considered unsafe for the Actor.
- write_to_elogbook – an Actor script can write a short message to the eLogbook service.
- set_alarm / clear_alarm – an Actor script can create a new alarm that will be shown on the LASER console expanding further possibilities of notifying operators that something on the Autopilot requires attention.

Unlike Monitors whose lifecycle is managed within UCAP, the lifecycle of Actor Tasks, is managed directly by the Autopilot server. This means the Autopilot server is responsible for creating a new process and shutting it down later when the user requests the Task to be started or stopped, respectively.

Graphical User Interface

The Autopilot can be monitored and, to a certain extent, controlled through an interactive expert application (GUI) that can be started from the Common Console Manager (CCM). The main screen of the GUI is the Beam Current Transformer (BCT) Stabilize panel. This panel provides a graphical overview of the current and historic state of the source current stabilization Task, discussed in some detail further in this document. It also allows one to start and stop the regulation Tasks, and to control the current set points.

Along with the BCT Stabilize panel, the GUI provides a number of additional panels with a similar functionality, namely the electron to H^- (eH) Ratio panel, the Caesium Flow panel, and the High Voltage (HV) Reset panel.

Integration with Accelerator Controls Services

The configuration of the input and output devices of the Autopilot is handled by the Controls Configuration Service.

To allow the Autopilot server to communicate its important actions to the outside world when necessary, Actor

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-WEPV018

Tasks have the possibility to write messages to the eLogbook. Behind the scenes the server uses the REST API of the eLogbook to write messages from the Actor Task to the appropriate section of the logbook.

This functionality is limited solely to Actor Tasks, as they are the only processes whose actions can have implications for the operation of the accelerator, and thus should be announced to operators when necessary.

The Autopilot server is also integrated with the LASER (Alarms) service, giving Actor Tasks the possibility to create a new alarm that will be visible by operators on the LASER console. The functionality is limited to Actor Tasks for the same reason as explained above for the eLogbook integration.

LSA (Settings Management Service) is used by the Autopilot to manage Actor settings. This allows the standard Accelerator controls graphical applications to visualise data published by the Autopilot, and to influence the behaviour of the Autopilot.

NXCALS (Logging Service) is used by the Autopilot to log the values published by the Monitors. Data is also extracted from NXCALS when starting the Autopilot GUI, to prefill charts with a certain amount of historical data.

Deployment

The two core parts of the framework are the Autopilot Flask Server, and the Autopilot UCAP Node (see Fig. 2.). The Autopilot Flask Server, as an Acc-Py-based service, uses Acc-Py tools for the releases of new versions. However, for the deployment on its dedicated server, a custom deployment script had to be developed, as currently Acc-Py lacks tools for services deployment. The UCAP Node, being a Java application at its core, uses CBNG [9] for both releasing and deployment.

Both services run on the same server running CentOS 7.

USER TASKS FOR THE SOURCE AUTOPILOT

In this section we describe the user tasks available and their status as of this writing.

BCT Stabilize

This task is the main workhorse of the Autopilot and its effect is illustrated in Fig. 3. Its objective is to keep the H^- beam current constant by adjusting the forward power of the 2MHz RF used for plasma production and heating based on readings from a Beam Current Transformer (BCT) in the Low Energy Beam Transport line (LEBT).

Phase Stabilize

As well as the RF power to the ion source, the phase of the RF is also measured. This phase can be used to assess the efficiency of the coupling of the RFQ power to the source plasma. The Phase Stabilize Task monitors the measured phase over defined windows and compares them to the set point. If outside a dead band, the frequency is adjusted along the pulse to return to the phase set point.

HV Reset

Ion sources typically suffer from sparking events in the order of a few per day. The source power converters go into a fault state when an over-current is detected, and do not deliver the voltage until reset.

The HV Reset Monitor delivers the present status of the HV converters, as well as counting the number of times that they have gone from ON to FAULT status in the last 24 hours.

The Actor reads the repeated HV converter statuses from the Monitor, and when it moves from ON to FAULT status, it initiates a procedure to restart them.

eH Ratio

The eH ratio is a Monitor that uses the currents measured on the source high voltage power converters, and the BCT

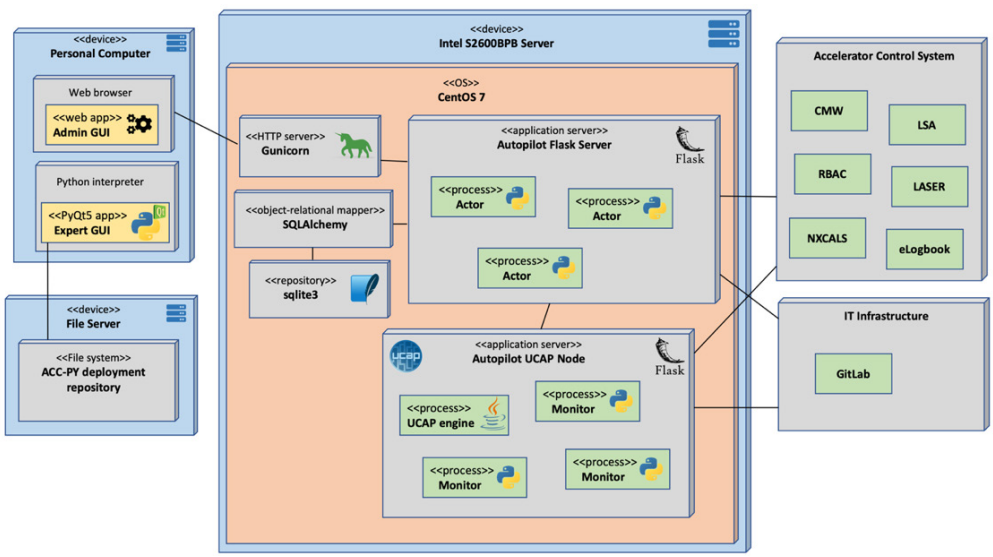

Figure 2: Autopilot deployment overview.

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-WEPV018

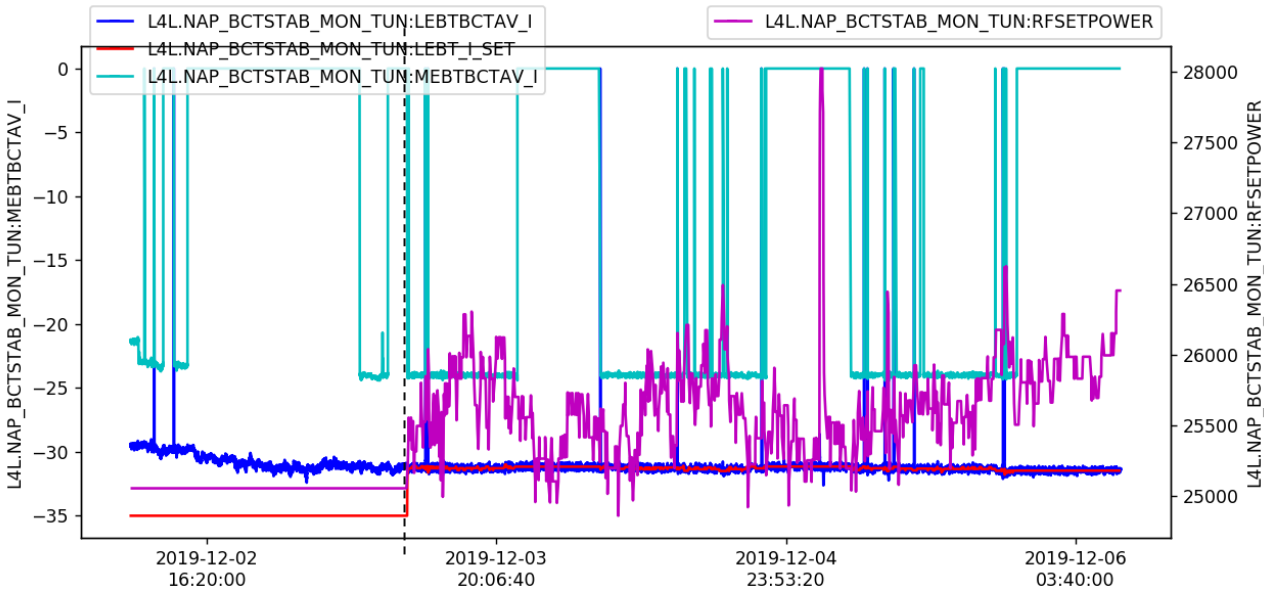

Figure 3: Beam intensity and RF set power from 2 December 2019 to 5 December 2019. Black vertical line shows the time after which the autopilot was started to maintain the MEBT beam intensity constant. Cyan= MEBT measured intensity. Blue=LEBT measured intensity, Orange=LEBT requested intensity, Purple=RF requested power.

current in the LEBT, in order to calculate the electron to H^- ratio from the source.

Cs Flow

The Linac4 ion source uses caesium to reduce the work function of the plasma electrode, which enhances negative ion production. Caesium is evaporated into the source, under vacuum, from a heated oven.

The amount of caesium that exits the oven can be estimated by scaling of the vapour pressure in the oven to a calibration point. Integrating the flow over time gives a total mass. This monitor produces this calculation in real-time, and allows the instantaneous flow and total mass data to be logged over time using the NXCALS system.

SUMMARY

Within the Linac4 Source Autopilot project, a framework has been developed to allow users to develop code in Python that runs on a stable server, and interfaces to several accelerator controls components to allow monitoring, settings, diagnostics, and testing. The framework is in use at CERN helping to control the Linac3 and Linac4 sources. The framework described in this paper can be deployed in various operational scenarios involving the integration of user code with general controls services. Although the accelerator Controls services leveraged by the framework are CERN specific, we are confident that fundamental building blocks of the framework can be reused outside the Organization.

Within this framework, tasks have been developed to continuously adjust the source RF power and perform automatic resets of the High Voltage converters. Thanks to this the stability of the source has improved remarkably over this period.

ACKNOWLEDGEMENTS

The authors would like to thank the Linac4 ion source team in BE-ABP for providing the motivation for this project and testing time, as well as to the software teams within BE-CCS for their work to make this project possible, in particular the UCAP, Acc-Py, and LASER teams.

REFERENCES

[1] G. Voulgarakis, J. Lettry, S. Mattei, B. Lefort, and V. J. Correia Costa , "Autopilot regulation for the Linac4 H− ion source", *AIP Conference Proceedings* 1869, 030012 (2017), doi: 10.1063/1.4995732

[2] B. Urbaniec and L. Burdzanowski, “CERN Controls Configuration Service - Event-Based Processing of Controls Changes”, presented at the 18th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'21), Shanghai, China, Oct. 2021, paper MOPV043, this conference.

[3] J. Lauener and W. Sliwinski, “How to Design & Implement a Modern Communication Middleware Based on ZeroMQ”, in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017, pp. 45-51. doi:10.18429/JACoW-ICALEPCS2017-MOBPL05

[4] W. Sliwinski, P. Charrue, and I. Yastrebov, “Status of the RBAC Infrastructure and Lessons Learnt from its Deployment in LHC”, in *Proc. 13th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'11)*, Grenoble, France, Oct. 2011, paper WEMMU009, pp. 702-705.

[5] L. Cseppento and M. Buttner, “UCAP: A Framework for Accelerator Controls Data Processing @ CERN”, presented at the 18th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'21), Shanghai, China, Oct. 2021, paper MOPV039, this conference.

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-WEPV018

[6] D. Jacquet, R. Gorbonosov, and G. Kruk, "LSA - the High Level Application Software of the LHC - and Its Performance During the First Three Years of Operation", in *Proc. 14th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'13)*, San Francisco, CA, USA, Oct. 2013, paper THPPC058, pp. 1201-1204.

[7] Z. Zaharieva and M. Buttner, "CERN Alarms Data Management: State & Improvements", in *Proc. 13th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'11)*, Grenoble, France, Oct. 2011, paper MOPKN011, pp. 110-113.

[8] J. P. Wozniak and C. Roderick, "NXCALS - Architecture and Challenges of the Next CERN Accelerator Logging Service", presented at the 17th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'19), New York, NY, USA, Oct. 2019, paper WEPHA163.
doi:10.18429/JACoW-ICALEPCS2019-WEPHA163

[9] L. Cseppento, V. Baggiolini, E. Fejes, Zs. Kovari, and N. Stapley, "CBNG - The New Build Tool Used to Build Millions of Lines of Java Code at CERN", in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017, pp. 789-793.
doi:10.18429/JACoW-ICALEPCS2017-TUPHA163