ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA156

CONTROLS CONFIGURATION DATABASE AT ESS

R. Fernandes†, S. Gysin, S. Regnell, European Spallation Source (ESS), Lund, Sweden
M. Vitorovič, S. Sah, Cosylab, Ljubljana, Slovenia
V. Vuppala, Facility for Rare Isotope Beams (FRIB), East Lansing, USA

Abstract

At the European Spallation Source (ESS), thousands of (physical and logical) devices will be in production and execute a wide range of functions to enable both the machine and end-station instruments to perform as expected from a controls point of view. Typical examples of such devices are racks, power supplies, motors, pumps, PLCs and IOCs. To properly manage the information of devices in an integrated fashion and at the same time allow external applications (consuming this information) to perform well, an application called Controls Configuration Database (CCDB) was developed at ESS. The present paper introduces this application, describes its features, architecture and technology stack, data concepts, interfaces, and ecosystem; finally, it enumerates development directions that could be pursued to further improve it.

INTRODUCTION

Many research facilities are routinely facing the challenge of managing huge amounts of heterogeneous controls-related information in a proper manner. Most have a panoply of databases to tackle this or, worse, a monolithic database composed of innumerous tables. Few facilities, however, have a truly centralized, flexible and coherent approach to manage such information which can ensure that 1) the development effort is kept at a reasonable level (by avoiding the proliferation of databases or a dense database which is difficult to maintain), 2) data duplication/inconsistency is mitigated and, most importantly, 3) users have access to a holistic view of the control system.

To overcome this challenge, the CCDB [1] was developed with the goal of enabling the collection, storage and distribution of controls configuration data needed to install, commission and operate the ESS control system from day one. Along with single sign-on capabilities shared with disparate ESS applications, a fine-grained user authorization mechanism, several ways to read/write data from/into, multiple views over the same information, the CCDB also has modelling capabilities: Users can define device types and associate properties to these, create devices (out of device types), and specify relationships between these. Additionally, users can represent the ESS control system in a hierarchical fashion through containers and slots (placeholders in the control system structure for devices). This modelling capability is actually the most prominent feature of this application as it enables it to cope with virtually any kind of controls scenario (that needs to be modelled/stored) but also lowers both its development effort and persistence layer complexity (i.e. database density). This feature considerably reduces the time users have to wait to represent/store devices as no software development is required.

Moreover, thanks to a well-defined programmatic interface, the CCDB supports external applications (e.g. IOC Factory) to perform their domain specific businesses adequately and share data in an efficient way.

DESCRIPTION

The origins of the CCDB can be traced back to a project called Proteus [2] developed at the Facility for Rare Isotope Beams (FRIB) with the main goal of managing controls configuration information. Due to this goal being similar to what ESS was pursuing in 2013, the initial code base of the CCDB actually sprung out of Proteus. A significant portion of this code base (and to a lesser extension the database schema) had to be re-written to cope with new (ESS) requirements though. Development of the CCDB started in early 2014 and went into production at the end of 2015. Since then, several versions have been deployed, with version 1.3 released in October 2017 being the latest. Table 1 summarizes key CCDB metrics.

Table 1: Metrics about the CCDB

Description	Value
Tables (persistence tier)	40
Constraints (persistence tier)	100
Indexes (persistence tier)	7
Lines of code (persistence tier)	0
Classes in Java (business tier)	388
Lines of code (business tier)	38,224
Web pages (presentation tier)	7
Dialogs (presentation tier)	21
Lines of code (presentation tier)	3,863

Nowadays, the CCDB is part of an international collaboration called DISCS [3]. This collaboration is composed of several research facilities (ESS being one of them) with the aim of developing databases, services and applications that any experimental physics facility can easily configure, use and extend for its commissioning, operation and maintenance. The CCDB is being developed as the configuration module within this collaboration, and is considered to be the most important module.

Features

Besides its major feature (modelling capabilities), the CCDB has other features or characteristics worth mentioning. These are:

† ricardo.fernandes@esss.se

- Single sign-on: thanks to RBAC [4] (an in-house application that provides user access based on roles), the CCDB shares single sign-on capabilities with other applications at ESS – e.g. Naming Service, IOC Factory. This eliminates the need of logging in more than once when working with these.
- Fine-grained user authorization: currently the CCDB authorizes users to perform operations through one of three existing roles defined in RBAC: 1) Default (role with minimum privileges that only allows users to view stored data), 2) User (role with intermediate privileges allowing users to view stored data and create/update/delete devices only), and 3) Administrator (role with maximum privileges allowing users to view stored data and create/update/delete all data concepts). This provides full control of who can do what in the CCDB.
- Multiple interfaces: several interfaces are provided by the CCDB to serve different needs including a graphical interface (web-based to support the ESS business model which has many users scattered across its European member states), batch data entry through EXCEL files, and RESTful services for applications to consume (see subsection Interfaces for additional details).
- Data reuse: data entries defined/done by one user (e.g. a device type) may be reused by another. This not only lowers the effort spent in performing this kind of task but also minimizes data inconsistencies that can otherwise emerge.
- Multiple views: the CCDB offers several views over the same data based on the type of relationship. In other words, users may browse data in function of Contains, Controls, Powers and Connects relationships, providing a better picture of the controls workflow.
- Inventory: basic inventory functionalities can be found in the CCDB allowing all devices at ESS to be registered (representing a warehouse) and a subset to be installed in a data concept called slots (representing devices in production).
- Traceability: the CCDB logs all operations that were performed which have changed its data (e.g. create a new property). Users may browse this to have a full understanding of who has done what and when.

Architecture and Technology Stack

The CCDB is a distributed system based on a classical client-server model [5] where users access its functionalities remotely through a web-based graphical interface and external applications access its data through a programmatic interface. This model – or architecture – is composed of three tiers, namely: Presentation (the layer which users interact with), Business (the layer which implements business logic) and Persistence (the layer in which data is stored and retrieved from). Fig.1 illustrates the architecture of the CCDB.

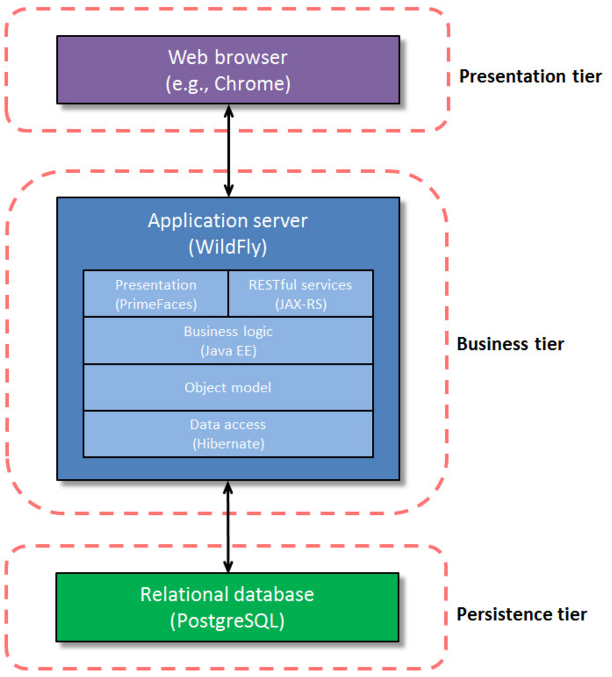


Figure 1: Architecture of the CCDB based on three tiers.

Several technologies are employed to implement this type of architecture guaranteeing that the CCDB is developed according to user requirements and expectations. Primordial criteria to select the technologies were that they had to be open-source, mature, well documented and actively maintained by the community. With these in mind, PostgreSQL, a relational database management system, is used to implement the persistence tier (i.e. database) of the CCDB. The business tier is implemented in Java EE running in an application server called WildFly. It uses Hibernate (a JPA implementation) to access data from the persistence tier and JAX-RS (a Java API) to expose data stored in the CCDB through RESTful [6] services. Finally, the presentation tier (i.e. graphical interface) of the CCDB is based on PrimeFaces, a JSF implementation.

Data Concepts

The CCDB provides several data concepts which users can work with to model any controls scenario they may have in mind. These can be seen as building blocks (*à la* LEGO) that can be assembled, fitted, organized and related together making it possible to represent the scenarios in a complete, flexible and hierarchical fashion. The following describes the data concepts that exist in this application.

- Unit: describes a measurement of a physical quantity, defined and adopted by convention or by law (e.g. volt, watt, hertz). This is the most elementary concept and is used to further define properties.
- Enumeration: groups values according to a certain context or logic (e.g. the ESS lifecycle has several states which can be grouped as construction, com-

mission and operation). This concept is used to further define properties.

- Property: describes a characteristic or attribute (e.g. manufacturer, model, frequency). This concept is used to further define device types.
- Device Type: describes a certain device (e.g. rack, motor, pump) in an abstract way through properties.
- Device: instance of a certain device type (e.g. rack_023, motor_098, pump_171).
- Container: folder that may contain other containers and/or slots allowing the creation of hierarchical models (e.g. BUILDING_2, ROOM_9, LEBT).
- Slot: placeholder for a specific device to be installed in which follows ESS naming convention (e.g. ISRC-1:Ctrl-Rack-1, LEBT-00:Ctrl-IOC-08, LEBT-00:Ctrl-PS-05). This is the most elaborated concept.

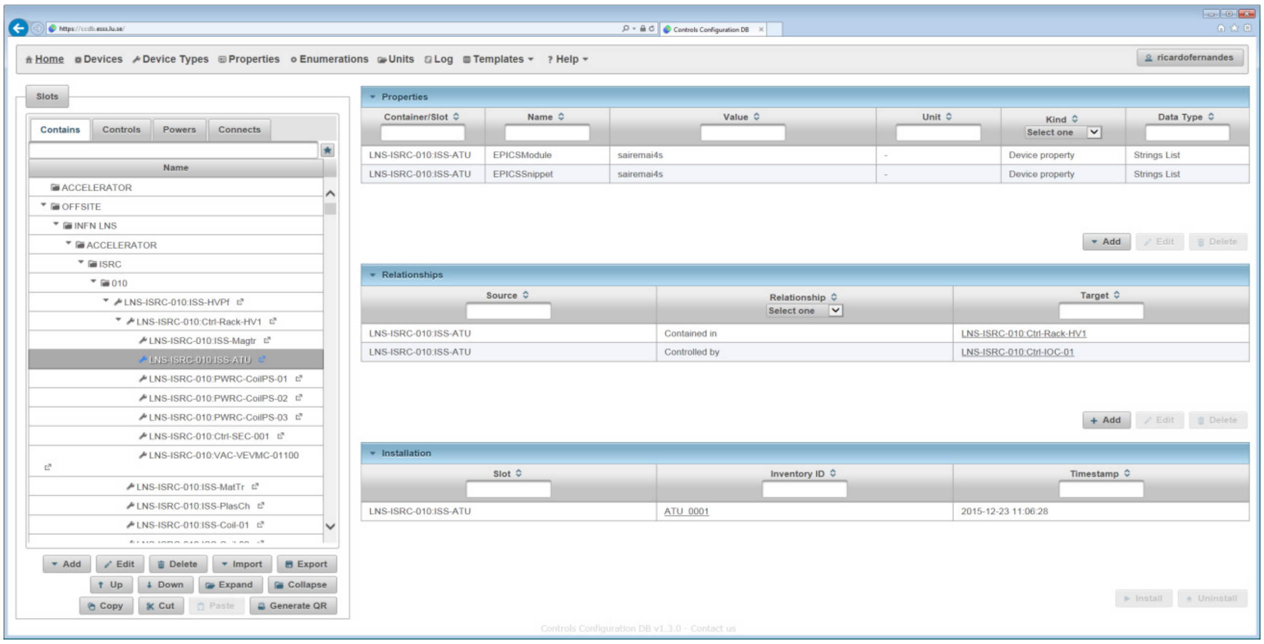
Another important concept in the CCDB, closely related with the previous ones, is relationship. This concept binds containers/slots with other containers/slots according to the domain specific logic, modelling the controls workflow appropriately. Currently, four types of relationships are offered: Contains (e.g. container ROOM_9 contains slot ISRC-1:Ctrl-Rack-1), Controls (e.g. slot LEBT-00:Ctrl-IOC-08 controls slot LEBT-00:Ctrl-PS-05), Powers (e.g. slot LEBT-00:Ctrl-PS-05 powers slot LEBT-00:ISS-Coil-02), and Connects (e.g. slot DTL-30:PBI-BCM-01 connects to slot DTL-30:PBI-BCM-01FE).

While the first three relationships are explicitly defined by users, the last one is automatically (and dynamically) calculated by the CCDB – i.e. users do not have to define it. This automation is possible thanks to RESTful services provided by an application called Cable Database [7] (responsible for managing the information of controls cables at ESS) and consumed by the CCDB which allows it to “know” what device connects to what.

Interfaces

As the workhorse for storing and distributing controls-related information at ESS, the CCDB provides several interfaces to read/write data from/into it, which are tailored to cope with different needs and levels of expertise. These interfaces, available to both users and external applications, are:

- Graphical interface: users can perform CRUD (i.e. create, read, update and delete) operations in all data concepts through a web-based interface provided by the CCDB. This type of interface is ideal when users need to view information in a user-friendly way, sometimes remotely, with heterogeneous devices (e.g. desktop computer, mobile phone), or to make punctual changes to information. Fig. 2 depicts the graphical interface of the CCDB (displaying partially the ESS Ion Source model).
- EXCEL file: users needing to perform bulk data import into the CCDB may use well-defined EXCEL files (downloadable from this application) for that purpose. Moreover, the same EXCEL files are used by the CCDB to export data, which users can modify and import back afterwards, thus closing the loop of bulk operations.
- RESTful interface: users can programmatically consume RESTful services of the CCDB to develop scripts and applications. This type of interface is highly effective as it enables 1) users to tackle specific needs that cannot easily be solved with neither of the aforementioned interfaces and 2) external applications to perform well and guarantee they continue being lean (by avoiding applications to store data – already in the CCDB – in their own persistence layers).



Name
ACCELERATOR
OFFSITE
INFN LNS
ACCELERATOR
ISRC
010
LNS-ISRC-010:ISS-HVPf
LNS-ISRC-010:Ctrl-Rack-HV1
LNS-ISRC-010:ISS-Magtr
LNS-ISRC-010:ISS-ATU
LNS-ISRC-010:PWRC-CoilPS-01
LNS-ISRC-010:PWRC-CoilPS-02
LNS-ISRC-010:PWRC-CoilPS-03
LNS-ISRC-010:Ctrl-SEC-001
LNS-ISRC-010:VAC-VEVMC-01100
LNS-ISRC-010:ISS-MatTr
LNS-ISRC-010:ISS-PlasCh
LNS-ISRC-010:ISS-Coil-01

Properties

Container/Slot	Name	Value	Unit	Kind	Data Type
LNS-ISRC-010:ISS-ATU	EPICSModule	sairemai4s	-	Device property	Strings List
LNS-ISRC-010:ISS-ATU	EPICSSnippet	sairemai4s	-	Device property	Strings List

Relationships

Source	Relationship	Target
LNS-ISRC-010:ISS-ATU	Contained in	LNS-ISRC-010:Ctrl-Rack-HV1
LNS-ISRC-010:ISS-ATU	Controlled by	LNS-ISRC-010:Ctrl-IOC-01

Installation

Slot	Inventory ID	Timestamp
LNS-ISRC-010:ISS-ATU	ATU_0001	2015-12-23 11:06:28

Figure 2: Graphical interface of the CCDB.

Ecosystem

A set of applications have been developed (or are under development) to support both integration and controls efforts at ESS. Most of these applications are consumers of the CCDB which, together with producers of the latter, compose a rich ecosystem that exchange data amongst them, promoting consumers to (stay) focus in executing well-defined tasks to solve issues intrinsic to their domain specific businesses – and delegating the responsibility of controls data storage, modelling and distribution to the CCDB. Fig. 3 shows the ecosystem with the CCDB at its core.

The CCDB has three producers where information is retrieved from in order to 1) authenticate & authorize users to perform certain operations or not (RBAC), 2) name slots in an official and unique way (Naming Service), and 3) view connections between devices automatically (Cable Database).

Concerning its consumers, the CCDB currently has two in production, namely the IOC Factory [8], which generates and manages IOCs in a lean, quasi-automated way (by retrieving the list of devices – and the names of EPICS modules needed to interface these – that an IOC controls from the CCDB), and the PLC Factory [9], which generates PLC code (by traversing the Controls relationships of a device and checking for attached artifacts to do search and replace operations). Additional consumers are under development, e.g., Calibration Service (which retrieves calibration coefficients as properties of devices from the CCDB to calibrate these), or are been considered such as the Preventive Maintenance Service [10] (which retrieves the MTBF and threshold values as properties of devices from the CCDB to notify users when maintenance of devices is due based on their effective usage and before becoming non-operational).

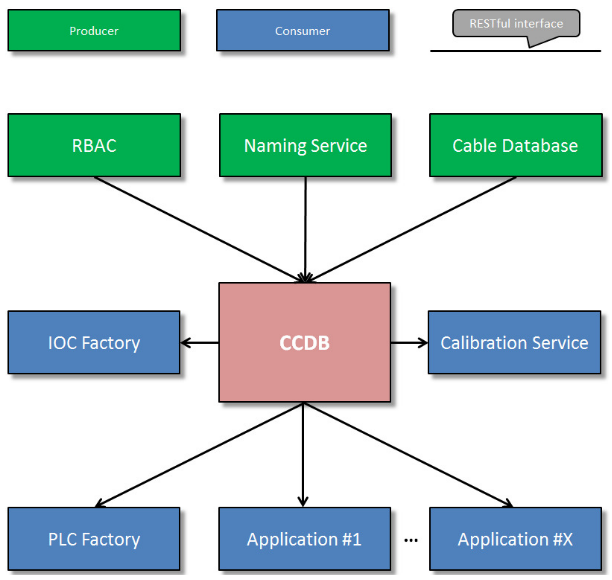


Figure 3: Overview of the CCDB ecosystem.

FUTURE DEVELOPMENTS

As the development of controls ramps up at ESS, a substantial amount of information about devices will need to be managed by the CCDB. Consequently, the user base of this application is expected to grow sharply along with requests for new functionalities. To anticipate some requests, the following (missing) functionalities are candidates for development in the near future:

- Data authorization: currently the CCDB implements user authorization at the level of functionality (e.g. who can add a new property to a certain device type). While this is most needed, user authorization at the level of data should be supported as well (e.g. who can update the value of a property of a certain device type).
- Data versioning: although the CCDB has a log functionality that fully tracks all changes (in other words, who has done what and when), it however does not let users easily view how the information of, for instance, a certain device evolves over time. Therefore, it could be beneficial to have a data versioning mechanism in the CCDB where users could view the evolution of the information of a device, compare two temporal versions of the information and, eventually, restore a version of interest for production.
- Data access: at present the CCDB provides 13 RESTful services allowing users to programmatically get (i.e. read) data stored in it. While these services cover all data concepts, they are intentionally generic, meaning that additional services should be developed to get data in more specific ways – this will not only increase performance between the CCDB and its consumers but also lower the development effort required by the latter. New RESTful services to put (i.e. write) data into the CCDB should also be developed so that consumers can benefit from these (this will for instance enable the IOC Factory, a consumer, to attach the list of PVs in each device stored in the CCDB after generating an IOC).
- Data import: while EXCEL files have been created to enable importing all data concepts in the CCDB, it may be time consuming to fill-up these files since they were designed to be as generic as possible. Therefore, additional EXCEL files to specifically represent complex devices (e.g. racks) should be created to lower both the time and effort users spend in doing (batch) data import into the CCDB.
- Data inheritance: in case two (or more) device types with almost identical properties need to be defined in the CCDB, users can create one device type, duplicate it (as many times as needed) and modify the copies to model what they are meant for. While this approach reduces the time spent in defining similar device types, it may lead to some difficulties in managing these in the long run. A more natural way to solve this is through inheritance. Consequently, the CCDB should provide an inheritance mechanism where users can define a generic device type (e.g.

PLC) and define specific device types (e.g. Siemens PLC and Schneider PLC) out of this generic type with some additional properties for each one of these.

CONCLUSION

The CCDB is in production at ESS since the end of 2015 with satisfactory results. Besides integration labs, this application already models/stores (partially) the information of ESS Ion Source, LEBT and MEBT systems, which users may view through its graphical interface or use external applications to manage specific issues based on information retrieved from the CCDB, enabling an effective integration and control of these systems.

While most users seem to have no difficulties to operate the CCDB, others do have some in particular when populating it. The reason for this is not inherent to the CCDB though but usually due to people lacking of modelling skills or access to relevant information – the latter being a typical situation found in research facilities in construction phase such as ESS.

Several interfaces (namely: graphical, EXCEL files importer/exporter, and RESTful) have been developed to cope with a multitude of requests so that users/external applications may achieve their goals with minimum effort imposed by the CCDB.

External applications have been developed to support both integration and controls efforts at ESS. These are consumers of the CCDB forming a (logical) ecosystem with/around it. Instances of existing consumers are the IOC Factory and the PLC Factory that generates IOCs and PLC code thanks to information retrieved from the CCDB. Additional consumers are under development (e.g. Calibration Service) or under consideration (e.g. Preventive Maintenance Service) to further enrich this ecosystem. A neat benefit of this architectural strategy is that the ecosystem may grow naturally as each of its members is developed independently, not disrupting or slowing down the others.

Finally, new functionalities are being envisaged such as authorization at the level of data, data versioning, new RESTful services to write data and read data in more specific ways, additional EXCEL files to import complex devices, as well as data inheritance. These will substantially improve the CCDB and allow users to profit even more from its usage.

ACKNOWLEDGEMENT

The authors would like to thank all the people that contributed with ideas and participated in discussions to further improve the Controls Configuration Database, in particular the members of the Hardware and Integration Group at ESS.

REFERENCES

[1] Controls Configuration Database, `http://openepics.sourceforge.net/configuration`

[2] V. Vuppala *et al.*, "Proteus: FRIB Configuration Database", in Proc. *ICALEPCS'13*, San Francisco, USA, October 2013, paper TUPPC03.

[3] V. Vuppala *et al.*, "Distributed Information Services for Control Systems", in *Proc. ICALEPCS'13*, San Francisco, USA, October 2013, paper WECOBA02.

[4] Role Based Access Control, `http://openepics.sourceforge.net/security`

[5] Client-server model, `https://en.wikipedia.org/wiki/Client%E2%80%93server_model`

[6] R. Fielding, "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000.

[7] Cable Database, `http://openepics.sourceforge.net/cables`

[8] IOC Factory, `http://openepics.sourceforge.net/ioc`

[9] G. Ulm *et al.*, "PLC Factory: Automating Routine Tasks in Large-Scale PLC Software Development", in *Proc. ICALEPCS'17*, Barcelona, Spain, October 2017, paper TUPHA046.

[10] Preventive Maintenance Service, `http://openepics.sourceforge.net/maintenance`