ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA143

A DATABASE TO STORE EPICS CONFIGURATION DATA

M. Ritzert*, Heidelberg University, Mannheim, Germany†

Abstract

The operation of extensive control systems cannot be performed by adjusting all parameters one by one manually. Instead, a set of parameters is loaded and applied in bulk. We present a system to store such parameter sets in a type-safe fashion into and retrieve them from a configuration database.

The configuration database is backed by an SQL database. Interfaces to store and retrieve data exist for the C++, Java and Python programming languages. GUIs are available both as a standalone program using C++ and Qt, and integrated into Control System Studio (CSS) [1]. The version integrated into CSS supports data validators implemented as Eclipse plug-ins that are run before each commit.

The format of the configuration data that can be stored is XML-like, and export and import to/from XML is implemented. The database can hold several completely independent "files" of configuration data. In each file, several branches can be stored, each branch consisting of a chain of commits. Each commit can easily be retrieved at any time. For each entry, the modification history can easily be queried.

INTRODUCTION

The context of the software presented here is the slow-control system of the Belle II pixel detector (PXD) [2].

The format of the data that is stored is similar to the structure of XML files: A tree structure of nodes gives structure to the data, and each leaf node contains one data point. To simplify integration with EPICS [3], a way to assign a PV name to each leaf is defined. For that purpose, attributes can be added to each node that add part of a PV name, or define units (for display purposes), or minimum and maximum values.

The database provides versioning of the data. It can contain several independent data sets, called *files*, and for each file, several *commits* can be stored in several *branches*. This structure is also shown in Fig. 1. The data are stored in such a way that any commit can be identified through just a single integer variable, the commit id. No data are ever overwritten, only updates creating new commits are possible.

One important decision taken during the design phase is that the access to the configuration storage has been separated from the hardware access. The IOC accessing the configuration database reacts to just one PV defining the commit id to load, and exports all PVs defined in the data set. On the hardware access side, the configuration data are taken from these PVs and applied to the hardware at the appropriate time, usually by means of a state machine.

* michael.ritzert@ziti.uni-heidelberg.de
† For the DEPFET Collaboration

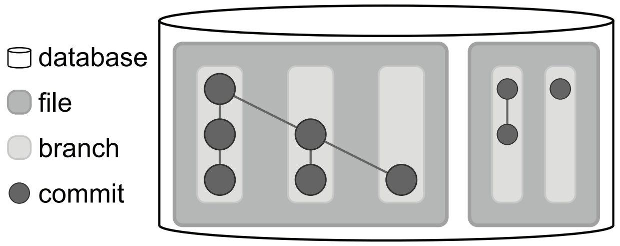


Figure 1: Contents of the Database. Several files with several branches and several commits per branch can be stored.

The overview of the proposed system is shown in Fig. 2: The editor is used to put the correct configuration into the database, from where it is retrieved by the configuration IOC. The IOC extracts the stored PV names and makes then available via the Channel Access protocol. State machines that control the configuration of the various subsystems read the data and apply them to the hardware.

Storage in the database and editing via the APIs are type safe. The supported data types are integer, floating point, string, and arbitrary-length binary. The data structures are compatible with the XML format, and this is the default format when importing from or exporting to a file.

DESCRIPTION OF THE SYSTEM

Graphical User Interface

The main interface to edit the configuration data is through an interface integrated into Control System Studio (CSS). In Fig. 3, the editor view available in the CSS environment is shown.

To help with the implementation of OPIs working with the configuration database, a Jython module is provided that can be used to access configuration database functionality from within OPI scripts. The OPIs can use a special data source, config://, to access the loaded configuration and its metadata. Together, this allows for device-specific configuration screens to be easily developed by means of the OPI editor.

It is also possible to compare the loaded configuration with the currently active values, and to selectively accept new values into the database.

Validation In order to prevent the operator from entering invalid data into the database, all data are validated before they are committed to the database. The idea is to detect obvious errors in the configuration way before it is activated in the system. The validation is run immediately before uploading to the database, and the results are reported to the user in the CSS GUI.

A generic framework to validate settings against fixed limits is available. As an example, typical limits could define

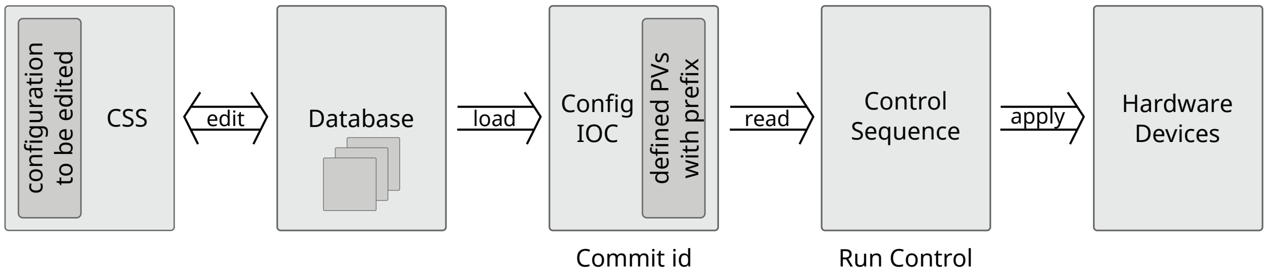
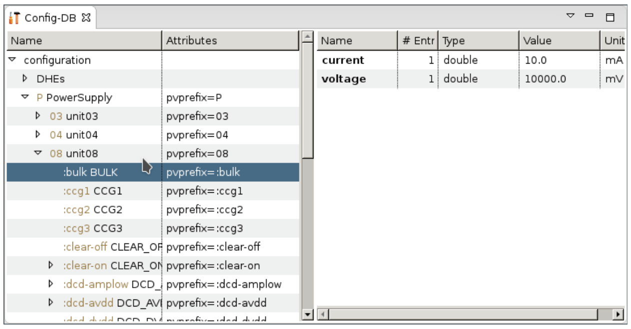


Figure 2: Overview of a control system using the configuration database.



Config-DB

Name	Attributes
configuration	
DHEs	
P PowerSupply	pvprefix=P
03 unit03	pvprefix=03
04 unit04	pvprefix=04
08 unit08	pvprefix=08
:bulk BULK	pvprefix=:bulk
:ccg1 CCG1	pvprefix=:ccg1
:ccg2 CCG2	pvprefix=:ccg2
:ccg3 CCG3	pvprefix=:ccg3
:clear-off CLEAR_OF	pvprefix=:clear-off
:clear-on CLEAR_ON	pvprefix=:clear-on
:dcd-amplow DCD_	pvprefix=:dcd-amplow
:dcd-avdd DCD_AVI	pvprefix=:dcd-avdd

Name	# Entr	Type	Value	Unit
current	1	double	10.0	mA
voltage	1	double	10000.0	mV

Figure 3: Screenshot of the main configuration editor integrated into CSS.

hard voltage or current limits of power supply units. Beyond simple limits for single settings, in virtually any complex system, there are interdependencies between various settings. A framework to implement such checks in Java code is available.

History View For each commit, descriptive metadata is stored. This includes a free-text commit messages, the username performing the commit, and the current timestamp. Changes to the data over time can be visualized in several ways. It is possible to query the history of a selected value, displaying the metadata of all commits that altered it. Another display shows all configuration differences between two commits.

Programming Language Interfaces

No complicated algorithms need to be implemented for the database, so supporting several interfaces in different languages is easily possible. The most advanced interface is in the form of the Java module that's also behind the CSS GUI. An implementation is C++ has been used to create the configuration database IOC, and a basic, standalone GUI in the form of a Qt application. An interface to the configuration database from Python is also available. In the PXD use case, this is used to automatically upload new calibration constants generated in a Python-driven measurement campaign.

Configuration Database IOC

The final part of the configuration database environment is the IOC that retrieves the data from the database and exports

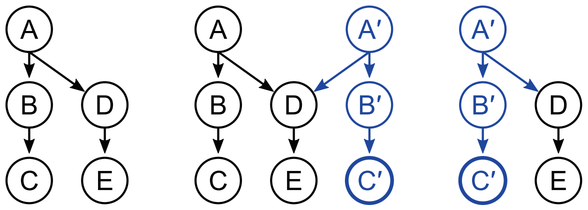


Figure 4: Node structure stored in the database. Left: Initial. Middle: After changing C to C'. Right: Only nodes reachable from A'.

the defined PVs. To load a new configuration, the desired commit id is written to the only writable PV managed by the IOC. The IOC then loads the configuration and exports the defined PVs. A fixed prefix is added to each PV, so that it doesn't collide with the active PVs, and to allow several configuration IOCs to run in parallel.

IMPLEMENTATION

Database

The data are stored using a data model that represents a "walking tree" data structure. The nodes of the tree are stored as individual rows, with an $M : N$ relationship between parent and child nodes: Obviously, each parent can have several children, and multiple parents of a node belong to different revisions of the data set. Updating a commit with new data requires only the leafs containing new data, and all nodes from these leafs up to the root of the tree to be written. All other data can be reused, so that the storage requirements are greatly reduced. As new nodes are written, they are assigned unique ids. Knowing the id of the newly written root node is sufficient to retrieve all data for the tree with a single, recursive select statement. This is also shown in Fig. 4: On the left, an initial tree is shown. In the image in the middle, a new revision has been committed, where C has been changed to C'. All data in blue had to be added to store this update. Node D now has two parents. On the right, the correctly updated tree that is returned when descending from the new root node A' is shown.

Access restrictions on PostgreSQL level ensure that no data can be modified once it has been committed. The implemented rules only allow new rows to be added to the database, but no updates of existing data. This ensures that once a commit has been stored in the database, its data cannot be altered in any way.

The current implementation uses the PostgreSQL database, but the Java implementation uses the generic JDBC database access layer, and could easily be ported to any RDB implementation that supports some kind of recursive query[1].

CSS Graphical User Interface

The GUI that can be integrated into CSS (or any other Eclipse-based application) uses the standard widget toolkit (SWT) [4] for a seamless integration. It provides a new view “Config-DB” that serves as the entry point to all configuration database functionality. Other views such as the history view, or the differences view are accessed via the view’s menu.

A RAP version for access via the Internet is also available.

The config:// datasource integrates into the diirt framework [5]. It enables access to all PVs defined in the currently loaded data set, as well as all corresponding metadata. A class providing static functions that is exposed to the Jython interpreter used by OPI scripts can be used to trigger actions in the configuration database from any OPI.

Validation The implementation of the validation framework is easily extensible, since it uses Eclipse plug-in extension points to decouple the validation framework from the various validators. All configuration data are available to a validator implementing the extension point, so that complex checks can be performed.

The static limit validator included with the CSS GUI uses XML files that map regular expression for the defined PV names to the corresponding limits. These XML files can be packaged in an Eclipse plug-in so that new settings can easily be distributed via the Eclipse p2 update mechanism. More complex checks can be implemented in a new plug-in that can also be managed via p2.

Configuration Database IOC

The main challenge in the implementation is that the set of exported PVs is dynamic, depending on the loaded configuration. Therefore, the CA server library [6] is used to implement the IOC in C++.

There is no hardware access in the configuration IOC. Instead, each device uses a state machine to apply the data to the hardware after retrieving them from the configuration IOC.

In the case of the PXD, the various state machines are synchronized by means of a global run control/power supply control scheme [7], and loading a new data set is only possible when the systems are in the idle state. Additionally, direct write access to devices’ PVs by the operator is prohibited in a PV gateway [8] while any system is running.

[1] MSSQL (CTE), and Oracle (CONNECT BY) should work.

CONCLUSION

We present a solution to manage configuration data for an EPICS-based control system in a database. The system comprises the API implemented in various programming languages, a powerful, extensible implementation of a configuration editor integrated in the CSS environment, and an IOC to provide the configuration to the control system.

ACKNOWLEDGEMENTS

This work has been supported by the German Federal Ministry of Education and Research (BMBF).

REFERENCES

[1] “Control system studio,” http://controlsystemstudio.org

[2] M. Ritzert, “Status of the epics-based control and interlock system of the belle ii pxd,” in *Proc. 15th International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS 2015)*, 10 2015, paper MOPGF164.

[3] “Experimental physics and industrial control system,” http://www.aps.anl.gov/epics

[4] “Swt: The standard widget toolkit,” https://www.eclipse.org/swt

[5] “diirt: Data integration in real-time,” http://diirt.org

[6] “Cas: Channel access server library,” http://aps.anl.gov/epics/extensions/cas/index.php

[7] T. Konno, R. Itoh, M. Nakao, S. Y. Suzuki, and S. Yamada, “The slow control and data quality monitoring system for the belle ii experiment,” *IEEE Transactions on Nuclear Science*, vol. 62, no. 3, pp. 897–902, 06 2015.

[8] “Gateway: The process variable gateway,” http://www.aps.anl.gov/epics/extensions/gateway