USE CASE: CONFIGURATION MANAGEMENT WITH A GENERIC RDB DATA-MODEL

T. Birke, B. Franksen, B. Kuner, R. Lange, P. Laux, R. Müller, G. Pfeiffer, J. Rahn
BESSY, Berlin, Germany

ABSTRACT

Advantages of flexibility and structural cleanness of the new RDB data-model for configuration management at BESSY correspond to increased demands on developer tools that handle data flow and structure. A sample application is given proving the feasibility of the approach: all configuration data are stored conforming to the new RDB data-model. The application itself controls a set of devices connected to the EPICS[1]-based control-system at BESSY. Configuration parameters are a set of device-specific hardware-access parameters and parameters for generic tools like alarm-handler, archiver and others. Use of the new RDB data-structure as well as sample tools to maintain the configuration parameters of this application are described.

Guidelines and conventions on how to use the new RDB data-model enable the developers to create generic tools for data-access and maintenance and thus allow to share pools of configuration-data and tools to access and maintain these data.

INTRODUCTION

This paper describes the intended use of the RDB configuration repository built up according to the "*Improved Data Model for Configuration Management*" introduced in [1]. The proposed second generation approach would be a full replacement for the current implementation, that is based on the *device* concept and in operation since ~1996. Today creation of configuration data is done with varying degree of automation: only the aspect of a device (or set of devices) can be retrieved from the RDB. Missing information necessary for the EPICS environment used at BESSY has to come from other sources:

1. Generation of **EPICS-DB** files – the real-time database running on an IOC[2]: Structure and functionality is designed using a schematics tool (*CapFast*) for the generation of templates. Instantiation data are automatically provided by RDB queries.
2. Generation of configurations for generic applications
   a. **Alarm-Handler** – the RDB provides complete device class lists, auxiliary informations allow for their grouping and partly the hierarchies. PVs[3] to be monitored are generated by manual completion of device names with the signal part, the logging instructions and the requested alarm action.
   b. **Archiver** – groups and prototype PVs are configured by hand, completion of device lists is done by RDB queries.
   c. **Save/Restore** – structure of PVs to be stored on-demand in snapshots of (parts of) the machine to be restored later is set up manually, filling of device lists and maintenance is provided by scripts using RDB calls.
   d. **autoSaveRestore** – application and IOC specific lists of PVs saved automatically to be restored on IOC-reboot (warm reboot), manually configured supplements of the IOC start-up scripts
   e. **DisplayManager** – Graphical User Interface – synoptics, frequently a mixture of documentation and device accessibility presented to the operator: only geometrical information or repetitive properties (multiplicities) can be extracted from the RDB.

---

[1]EPICS: Experimental Physics and Industrial Control System – see http://www.aps.anl.gov/epics/
[2]IOC: I/O Controller – a computer running EPICSbase, at BESSY a VME-based CPU
[3]PV: Process Variable – a signal, that can be accessed through the control-system

In summary: steps 1 and 2.a.-2.d. are well suited to be created fully automatic once the "which PVs" questions can be answered by the RDB. 2.e. will in most cases need manual contributions because of the difficulties to create intuitive GUIs automatically.

All these configurations are based on (or at least can be mapped into) hierarchies of information-structures. The inherent hierarchical structure is evident since modern applications and almost all newly developed systems hold configuration and application-data in structures based on XML, a strictly hierarchical markup-language.

MOTIVATION

While the generation of EPICS-DB files is automated to a high degree, some of the information needed to create application configuration files is located next to the applications (source-code, external file generator tools) – not in the RDB. The reasons for that misalignment are manifold: inappropriate extension of initially tentative or auxiliary approaches, continuously changing demands on applications, etc. The resulting complexity could not be mapped into the existing RDB-model. After several years of development and evolution of existing applications, there is too much information located outside the RDB, to prevent a fully automatic generation of configurations 2.a. to 2.d.. Even worse, it is difficult to actually modify the RDB data-model in its own application range to meet requirements of new device types and classes.

WHAT TO GENERATE?

- Create EPICS-DB without stand-alone template files
  - All information located in RDB (record-types, -descriptions and -names, field-names) to determine full PV-name.
  - Transform template/substitution-mechanism that is currently used to the new RDB data-model
    - build reusable components that can be instantiated and parametrized elsewhere
  - Allow overwriting of default-attributes set in a template in the instantiation
- Create Configuration for higher-level tools
  - Alarm-Handler – hierarchical clean prototype application
    - PV selection criteria, hierarchy, grouping, additional attributes (aliases, control-panel to launch, alarm-mask, alarm-filter settings)
  - Archiver – structural elements similar to alarm-handler, ordered by physical correlations
    - Differing PV selections, groupings, attributes (monitor/get, expected ROC of PV, ...)
  - Save/Restore – similar to alarm-handler, complex group-interdependencies
    - Another PV selection scheme, hierarchies different, sequential dependencies
  - autoSaveRestore – simpler than alarm-handler (flat data blocks)

GET STARTED

Step 1 – Elemental Compositions: Model the EPICS Database Definitions

The EPICS database definitions contain definitions of all known EPICS record-types that belong to a specific EPICS-release in a well-defined environment (application-developers, may define own or re-define existing record-types).

At BESSY it is declared convention to avoid to branch off application-specific record-type definitions to keep the shared code-base as large as possible and avoid setups specific to certain applications and/or IOCs.

EPICS Database

To open the possibility to create the complete EPICS Database (EPICS-db) from the RDB, it is a reasonable start to also put all the EPICS Database Definitions (EPICS-dbd) into the RDB. A simple (and incomplete but for this example sufficient) way to model the EPICS-dbd in the new RDB data-model is shown in Figure 1.

The main hierarchy consists of four levels of *gadgets*, since we just concentrate on the EPICS record definitions.

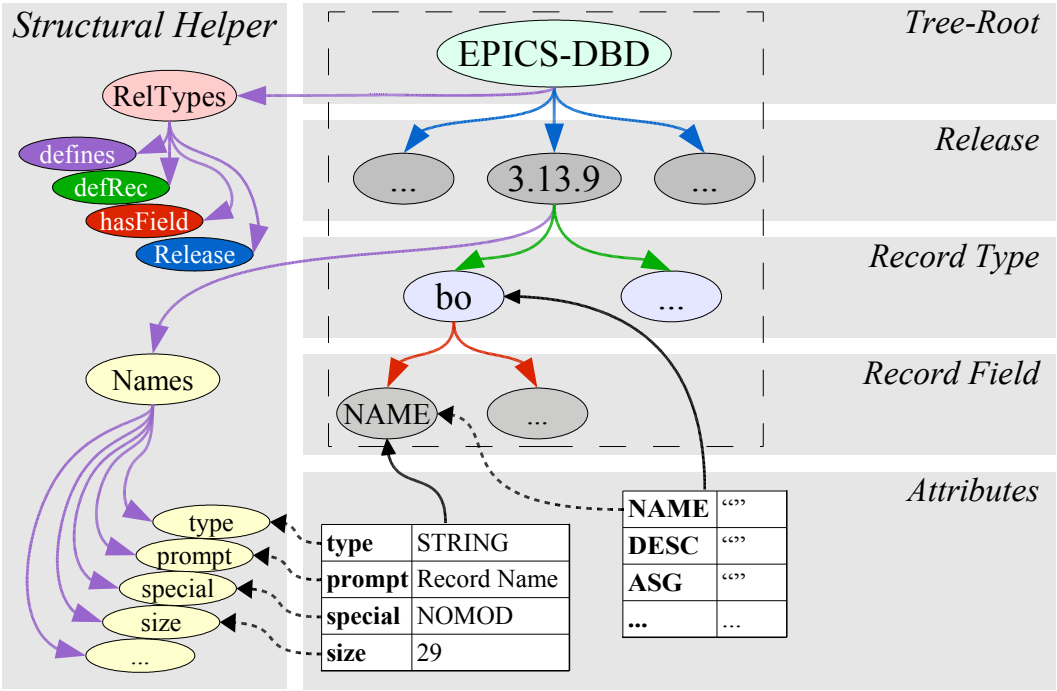

Figure 1: EPICS-dbd – the Record Definitions

1. The *tree-root*: One dedicated node, that can be seen as a group-identifier for the hierarchies to be modelled.
2. *Releases*: Different EPICS-releases may also differ in their database definitions. To accommodate this, we use the release-number as a node in the hierarchy.
3. *Record Types*: An EPICS dbd defines a certain set of EPICS record-types.
4. *Record Fields*: Every record-type defines a set of fields.

Every node in this hierarchy may have an arbitrary set of *attributes* attached to it. Attributes are name/value pairs, that may contain any type of information as its *value* while the *name* is not just a simple string, but a reference to a name-giving *gadget*. Parent/child-relations between gadgets have an additional gadget linked, to distinguish meanings of relations (relation-types). Those additional gadgets, that are necessary to build hierarchies and add attributes, are also regular gadgets, stored in separate hierarchies (structural helpers, see Figure 1).

Step 2 – Re-Compose Complete Devices: Definition and Instantiation of a Power Supply

For this example, it is sufficient to model a very simple power-supply type *MK1* having just a few I/O signals like an analog setpoint and two binary commands. This power-supply type is modelled as a template, that is then instantiated twice (*PS1* and *PS2*) where *PS1* overloads the attribute *DRVH* to set a different drive-limit.

A script/program can then create the complete EPICS db-file out of the RDB. The only information the script has to have coded into is *how* the hierarchies are built and what the relation-types are, that are used. So in this example, the path that holds all information is

/PowerSupApp/Instance/%/[isOfType]/%/%%/[hasField]/%

This path e.g. matches

*/PowerSupApp/Instance/**PS1**/[isOfType]/MK1/**setCurr**/ao/[hasField]/**DRVH***

which then defines one special PV *(PS1:setCurr.DRVH)* existing in the control system.

Figure 2 illustrates these hierarchies and shows the relations to the aforementioned EPICS-dbd hierarchy. The PS-templates, that are to be defined, correspond to e.g. the *CapFast*[4]-schematics that are currently used at BESSY to define EPICS-db templates. The hierarchies resemble the internal structures of the *CapFast*-schematics. In this example, the PS-template hierarchy has a depth of just

[4]CapFast: a schematic editor, widely used as a graphical frontend to create EPICS-db files

one level, but it is also possible to build complex template-hierarchies that re-use parts of other templates. This way, it is possible to transform all elements *CapFast* provides to structure a schematic into hierarchies in the RDB.

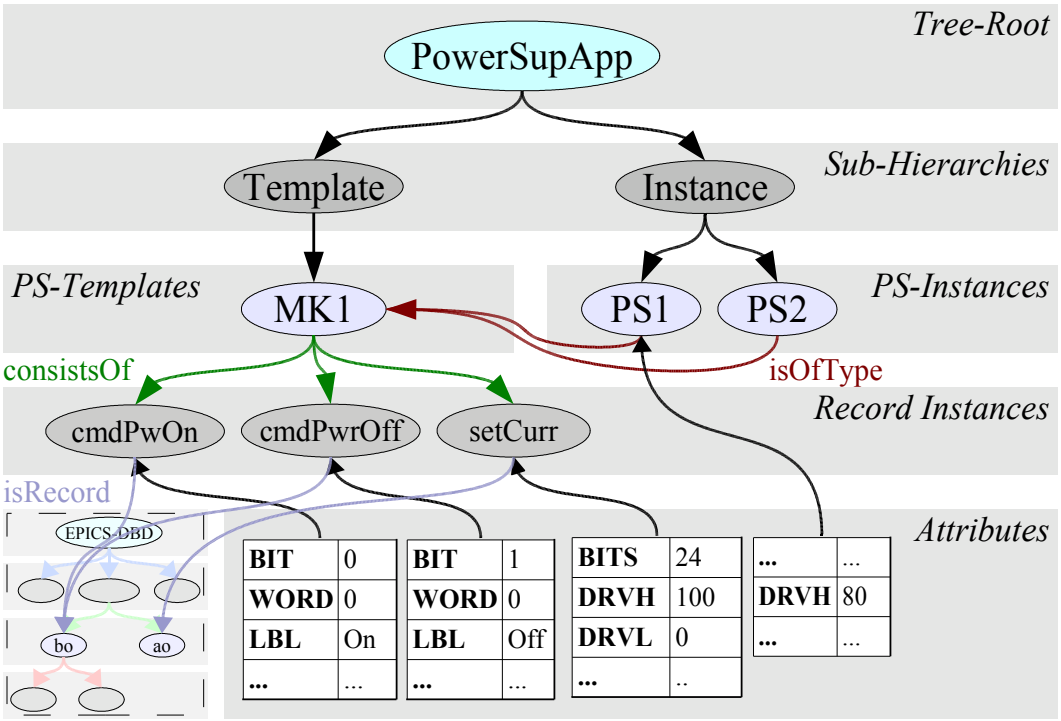

Figure 2: Definition and Instantiation of a Power-Supply

All these special cases and exceptions, that are currently coded in applications and scripts that query the RDB to create configurations, can now be covered by altering/extending the RDB *contents* instead of having to modify the RDB *structures*.

Step 3 – Generate Prototype Application Configuration: Alarm-Handler

An Alarm-Handler configuration for the EPICS standard alarm-handler *alh* consists of a multi-level hierarchy with nodes (groups) and leaves (channels). Each group or channel has additional attributes

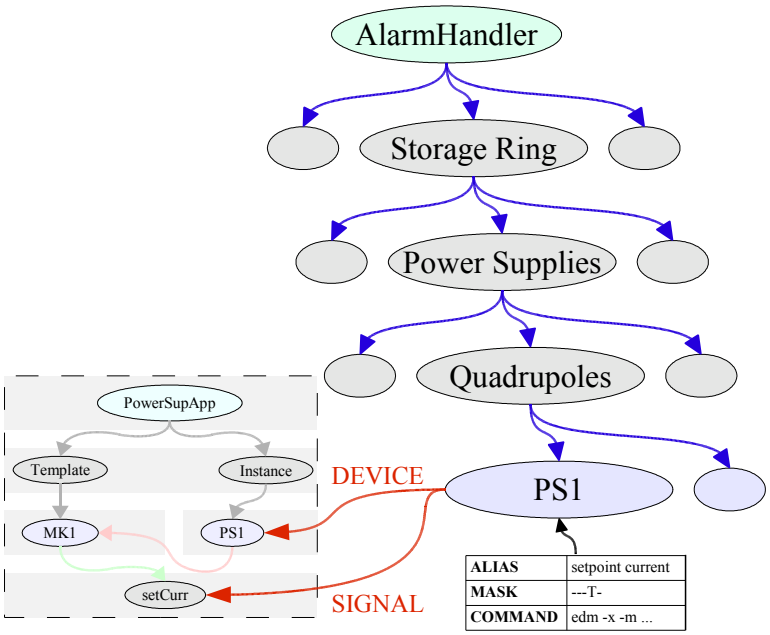

Figure 3: Alarm-Handler

that can be set individually. Leaves are the entries in the alarm-handler configuration, that reference entities in the RDB model, that are external to the alarm-handler config. For this reference an EPICS-record name is necessary (*DEVICE:signal* according to the BESSY naming-convention). So a valid channel entry in the alarm-handler configuration would refer to *PS1:setCurr* as shown in Figure 3.

For a complete RDB representation of alh configurations the DB has to deliver:

- ***DEVICE*** – corresponds to the PS-instance in Figure 2 (*PS1* or *PS2*)
- ***signal*** – corresponds to the record instance in Figure 2 (e.g. *setCurr*)
- ***structure*** **–** corresponds to PV hierarchies, groupings as they evolved according to operational needs, which have been manually implemented, refined and maintained so far.
- ***meaning*** – corresponds to the ever changing limits, severity levels, etc. that could possibly be classified according to specific operation conditions.

NEXT STEPS

Many functionalities and dependencies of the current control system configuration are scattered over many places. As it has been shown for the preceding examples it has to be analysed for all data pools which migration method is best suited:

Although EPICS-db files do not have any internal hierarchical structure but are a flat list of EPICS-records, it is common to all installations, that these files are created utilizing re-usable structural elements like template-files. The CapFast drawing editor used for the creation of templates e.g. allows for building hierarchical schematic drawings of EPICS-db files. These hierarchies are schematics that can be instantiated with different parameters multiple times in new schematics. This can be re-modelled as-is in the RDB using separate template hierarchies – alternatively a flat conversion tool could import all template instances into a less complex but functionally equivalent RDB structure.

During development of the first applications using the new RDB data-model, the image of how to model hierarchies (common names for relation-types, common names of gadgets grouping information structures...) will evolve and get clearer. In this phase, guidelines and conventions will manifest and allow for development of generic tools and applications to ease retrieval and maintenance of data stored in the new data-model.

Conditional statements in scripts implicitly implement filters and relations that have to be re-mapped into RDB structures. Frequently their present consequences or possible generalizations are unclear. Their scope and potential of duplication/contradictions have to be found out.

The result of this analysis will add to the conventions on how to use the new RDB structure to preserve simplicity, transparency and cleanness.

SUMMARY

It is still unclear how much restrictions on and knowledge about the resulting macroscopic structure of the RDB is required to make the content applicable and useful – or, on the other side, how much of the (global) structure elements can be retrieved by making use of the introspection capabilities of the new structure. In any case a lot of detailing work is required to convert the huge inventory of a first generation RDB configuration management tool with all its auxiliary data repositories into its next generation stream-lined, complete and homogenized form.

REFERENCES

[1] T. Birke et al., *Beyond Devices - An improved RDB Data-Model for Configuration Management*, ICALEPCS 2005, PO1.078-7