Introducing I/O Channels in the BESSY Device Database

Thomas Birke

based on work by
T. Birke, B. Franksen, R. Lange, P. Laux, R. Müller
BESSY • Germany

8th ICALEPCS • November 2001 • San Jose
Introduction

- BESSY uses EPICS as control system
- *channel* is the atomic element and e.g. represents an I/O signal
- *channels* are usually records in a runtime-DB (RTDB)
- Each *device* (abstract unit) consists of a set of signals
  \[ \rightarrow \text{channels: } \langle \text{DEVICE} \rangle : \langle \text{channel} \rangle \]
- *channels* have *attributes* that configure behavior or carry data
  \[ \rightarrow \langle \text{DEVICE} \rangle : \langle \text{channel} \rangle . \langle \text{ATTR} \rangle \]
- Naming convention only for *devices*
- All tools as well as RTDB need the full channel name
  - many of them even need the attribute name
Current State

Database System

Device oriented model in RDBMS (DB)

- Configuration/parameters are stored and retrieved by device or device class (power supply, magnet, rf, vacuum pump...)
- Every class has its own set of tables/views...
- Devicename is bootstrap

<DEVICE>:<channel>.<ATTR>

but <channel> and <ATTR> are not (always) in DB!

→ Additional knowledge is needed in every DB client to deduce the full channel name
Current State - Database System

Schematic View

Power Supplies
- family
- I/O params
- PS name
- device specs
- EPICS specs

Magnets
- device specs
- magnet name
- family
- measure

Controls
- node name
- CAN segment
- node specs
- IOC
Current State

**Creation of RTDB**

- RTDB-templates contain functionality/logic especially the `<channel>` and `<ATTR>` part
- RTDB-templates created from other source e.g. graphical editor
- Device (class) specific parameters are in DB
- Scripts/programs are used to retrieve the data from the DB and create configuration files for RTDB (substitution files)
- Powerful representation of device classes in the DB but just a few signals/channels and attributes
Current State

Creation of RTDB

Standard mechanism
power supplies, vacuum system, RF PLC-interface, triggers/timings, …

Many instances (substitutions) of few different types (templates)
Current State – Creation of RTDB

“Flat” I/O

◆ Preconditions
  ■ Interfacing systems that have many signals
    – but “no” structure in this signals
  ■ e.g. RF: 5 identical PLCs with ~300 signals each

◆ Data in RDB
  ■ Device-name
  ■ Signal/channel-names and –types
  ■ Attributes and I/O specifications of signals

◆ Simple templates
  usually one or two records

◆ DB harder to “read”/maintain manually

◆ Tcl/perl-scripts to create substitution rather simple
  (without any exceptions, everything except \texttt{<ATTR>} is in DB)
Current State – Creation of RTDB

Power Supply

- **Precondition**
  - About 400 PS of 12 different types
  - All share same interface
    - I/O card, embedded controller and CANbus interface

- **Data in DB**
  - PS specific data (~400 entries)
    - name, type, max. current, drive limits, precision...
  - I/O specific data (~400 entries)
    - embedded controller, CANbus segment/node...
  - Definition of different power supply types (12 entries)
    - analog I/O, commands, status bits

- DB relatively easy to “read”/maintain manually
- Tcl-script to create substitution has ~800 lines!
  (including several “special” cases, that are not handled in DB)
Current State - Creation of RTDB

"Complex" Singular Devices

- **Preconditions**
  - Few instances (usually one) of a particular device type
  - "complex" template with many tightly connected records
    (e.g. GPIB scope ↔ 32 records)

- **Data in DB**
  - *None at all!*

- **RTDB created manually or from template**

- **If at all, substitutions created manually or with a very simple script**

- **Tcl/perl-scripts are rather primitive (~20 lines)!**
Current State - Creation of RTDB

Bootscript Generation

**Requirements**
- Which drivers to load and initialize
- Which RTDBs to load
- Which hardware to initialize and how
- Which Programs to start

**Data in DB**
- Few information spread but not used
- Application with manually maintained ASCII-files creates bootscripts
- Highly generic, but still maintained manually
Applications

Different types of applications

- A script creates configuration files for an application (save/restore/compare, archiver, alarm handler…)

- The application accesses the DB directly (modeling tools, orbit, optics…)

- All have knowledge about the `<channel>..<ATTR>` combinations that are available for all devices they configure an application for
Current State – Applications

Roles of Devices

- Value of information depends on operating condition and/or information about other devices
- Applications have to behave different depending on the state/mode of the whole machine

Examples:
- Alarms: transient/conditional alarms
- Archiving: active period, frequency/monitor
- Save/Restore: load condition of snapshots
- Model: active inactive elements

- Coding these in every application is error-prone
- Experience with machine necessary to define these roles
Current State - Applications

Archiver / Alarms / Save/Restore

◆ Requirements
  ■ Which channels to archive / monitor / save/restore at all
  ■ Hierarchy/grouping of channels
  ■ Roles of a device

◆ State
  ■ Scripts contain hierarchies of sophisticated SQL-queries and raw name-lists or configurations are hand-coded
  ■ Roles either hand-coded or non-existent at all

◆ Problems
  ■ `<channel>..<ATTR>` missing in DB
    → “assumptions” in script
  ■ Raw name-lists are error-prone
  ■ No roles-definitions at all
Current State – Applications

Save/Restore/Compare

Requirements
- Which channels to save/restore at all
- Hierarchy/grouping of channels
- Save/Restore conditions (roles) of a device

State
- Scripts contain hierarchy of sophisticated SQL-queries and raw name-lists
- Conditions hand-coded

Problems
- `<channel>.<ATTR>` missing in DB
  → “assumptions” in script
- Raw name-lists are error-prone
- No roles-definitions at all
Current State – Applications

Archiver

Requirements

- Which channels to archive in which way
- Retrieval: hierarchy/grouping of channels
- Dynamic archiving conditions

State

- Some configs created by simple scripts, some hand-written
- No dynamic archiving at all

Problems

- `<channel>.<ATTR>` missing in DB
- → “assumptions” in script
- Raw name-lists are error-prone
- No roles-definitions at all
Current State – Applications

Alarm Handler

Requirements

- Which channels to monitor alarms of
- hierarchy/grouping of channels
- Qualification of alarms / conditional alarms (roles)

State

- Some configs created by simple scripts, some hand-written
- Qualification/conditions hand-coded

Problems

- `<channel>.<ATTR>` missing in DB
  → “assumptions” in script
- Raw name-lists are error-prone
- No roles-definitions at all
Current State - Applications

Modeling/Orbit/Optics

Requirements
- Magnets with function, length and position
- Mapping magnets ↔ power supplies
- Conversion factors $H = f(I)$
- Device-Interdependencies and HW-status

State
- All information in DB and retrieved on startup
- Meaning/function solved by naming convention
- But mapping to `<channel>.<ATTR>` still missing

Problems
- Interdependencies/HW-status solved, but not in DB
- So is mapping to `<channel>.<ATTR>`
Deficiencies

For many device classes `<channel>.<ATTR>` is not available in the DB but only in the RTDB-template

- Additional knowledge required in scripts that access DB
- Redundant information is spread among several scripts
  → Scripts are hard to maintain

Tailored sets of tables per device class

- Require additions for new device classes or even changes in existing ones
- Some table-sets are similar but not identical
- Sums up to ~200 tables (some of which have >30 columns)!
- Impossible to have a generic interface to data in the DB
  → Not very flexible, hard to maintain
Missing

Set of tables/views to hold

- All existing `<DEVICE>:<channel>.<ATTR>`
- Meanings of channels
- Application-specific attributes
- Generic roles of devices/channels
- Dependencies between devices

Some of the missing information could be “patchworked” into the existing DB, but most *not* (DB would become unmaintainable)
The Idea – A New Scheme

- Produce a common generic framework for all devices and applications
- Not lose too many features we got used to
- Cover all aspects we missed (so far)
- Device class tables, RTDB-templates and attributes build new core of DB-structure
- Specify configuration data not only for individual devices but also for groups
- Control system/device structure not modeled in DB-structure but DB-data
The Idea – Further Goals

- Consistent and complete model of control system down to single channels, including
  - Global name service
  - Global repository for re-usable signal-definitions
  - Capability to not only store control system I/O specific information but also support higher level applications

- Extendable and re-configurable hierarchy of devices

- Unified data source for all EPICS applications at every level of abstraction
  - e.g. high level (client) and low level (RTDB) data

- All configurations are to be generated from the DB
  - No more hardcoded knowledge in scripts
**Basic Elements**

- **Name**
  
  The central repository of device-names (naming convention)

- **Gadget**
  
  - A specific device or a group (class, family) of devices/groups
  - Tree-like structure (leafs are actual devices)
    
    Higher level nodes provide grouping/abstraction
  - Every gadget has a unique *name*

- **Signal**
  
  - The building block of which devices are composed
  - Tree-like structure (leafs are actual signals/records)
  - Roughly corresponds to RTDB template
  - Usually contains one - or few tightly interacting - record(s)
  - e.g. “an analog readback from a CAN-bus I/O-card”
Basic Elements

◆ Attribute
  - Each signal can have a number of attributes
  - Each attribute is related to exactly one signal
  - An attribute of one signal can be “translated” into one or more attributes of a child signal

◆ Record
  - Instantiates an entity, that can be filled with values
  - Connects gadgets with signals
  - Completely automated table (insert/delete)

◆ Process Variable (PV)
  - The actual configuration values
  - Connects an attribute with a record and may assign a value
Example (simplified)
Example (simplified)

Signals / Attributes

xtend

Uout  short

set

on

stat

rdbk

off
Example (simplified)

<GADGET> : <signal> . <ATTR>

PS : rdbk . MUX ← 2
PS-1 : rdbk . MUX ← 3
PS : simple . EGU ← "A"
MK-2 : simple . MAX-I ← 350
PS-3 : Uout . PREC ← 5
Quad : alpha . VAL ← 4.7538
The Price

The advantages do not come for free!

- Graphical editor to create RTDB or RTDB-templates can no longer be used directly
- The model is *very* (too?) abstract
  - Tables no longer maintainable by entering SQL manually
- Thus we need high-level tools and scripts
  - To fill and update the tables
  - To convert existing applications from template files to DB
- The plan is, to develop generic web browser frontends and standalone command line scripts

→ A lot of work to be done!
Project Status

- Tables and general structure are implemented
  - No real data (except names) in those tables yet

- Views, frontends and generic scripts still missing

- Next steps
  - Test system with a new application
  - Develop necessary frontends and scripts in parallel

- Dream: a graphical editor with SQL backend to create/maintain the signals
Summary

- Most difficult problem: determination and breakdown of natural data sources

- High-effort reengineering task – current situation may be an efficient compromise

- Migration of applications, that actually work

- Persistent and boring work-load due to manual configuration update is constant source of motivation to pursue this project