THE CERN ACCELERATOR MEASUREMENT DATABASE: ON THE ROAD TO FEDERATION

C. Roderick, R. Billen, M. Gourber-Pace, N. Hoibian, M. Peryt, CERN, Geneva, Switzerland

Abstract

The Measurement database, acting as short-term central persistence and front-end of the CERN accelerator Logging Service, receives billions of time-series data per day for 200,000+ signals. A variety of data acquisition systems on hundreds of front-end computers publish source data that eventually end up being logged in the Measurement database.

As part of a federated approach to data management, information about source devices are defined in a Configuration database, whilst the signals to be logged are defined in the Measurement database.

A mapping, which is often complex and subject to change/extension, is required in order to subscribe to the source devices, and write the published data to the corresponding named signals.

Since 2005, this mapping was done by means of dozens of XML files, which were manually maintained by multiple persons, resulting in a configuration that was error prone.

In 2010 this configuration was fully centralized in the Measurement database itself, reducing significantly the complexity and the actors in the process. Furthermore, logging processes immediately pick up modified configurations via JMS based notifications sent directly from the database.

This paper will describe the architecture and the benefits of current implementation, as well as the next steps on the road to a fully federated solution.

MEASUREMENT SERVICE OVERVIEW

The Measurement Service (MS) [1] is a vital component of the mission critical CERN accelerator Logging Service [2].

As shown in Fig. 1, the MS processes are responsible for subscribing to data published from accelerator equipment and persisting the results in either the Oracle Measurement database (MDB) or in SDDS (Self Describing Data Sets) files.

An optimized Java API has been developed, by which data for some 200 thousand signals are persisted to the MDB, from where a sub-set of the data is later transferred to the Oracle Logging database (LDB) for long-term storage. A complimentary logging to SDDS files is used typically to store data for complex data structures such as image captures or multi-megabyte beam profiles.

Java APIs are provided for extracting data from both the Logging Service databases and the SDDS file repository.

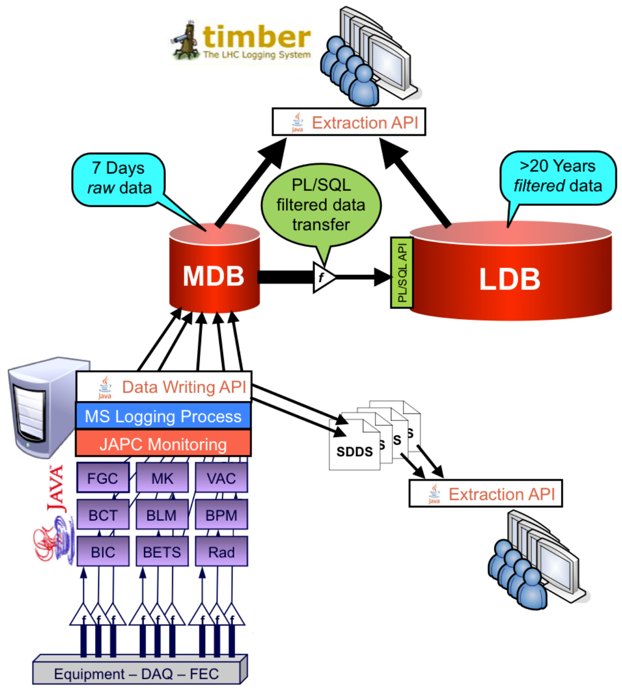


Figure 1: Measurement Service architecture and position within the Logging Service.

CONFIGURATION ESSENTIALS

In order to log data, a number of configuration steps are required:

Data Logging

It is a prerequisite to register all signals for which data needs to be logged in the MDB. The names of the signals must adhere to the well-established naming conventions, and complimentary information such as units and a meaningful description are required.

For signals whose data should be kept beyond the 7-days supported by the MDB, the filtering criteria (dead band and dead time smoothing, precision etc.) for transfer to the LDB must also be defined.

Data Acquisition

In order to acquire values to be logged, controls middleware (CMW) subscriptions via JAPC [3] need to be configured to obtain publish values. This implies declaring which device-properties to subscribe to, under which conditions (i.e. only for beams to SPS, LHC, etc.), and how the data should be time stamped (either the acquisition time, or the start time of the corresponding magnetic cycle). In addition, it is possible to specify special conversion code to be applied to published values prior to logging.

Bridging the Gap

Once signals are defined and JAPC subscriptions configured, it is necessary to bridge the gap between the two domains. This requires that individual fields of device properties are mapped to logging signals. For data published in array format, it is possible to configure the mapping of either individual elements or a subset of elements to scalar or array type logging signals respectively.

INITIAL IMPLEMENTATION

The initial implementation of the configuration of the MS was comprised of 3 components:

- Logging signals pre-registered in the MDB, and defined by means of data providers completing an Excel template with the details of the signals to be logged, which was validated by members of the logging team responsible for the MDB.
- JAPC parameter subscriptions defined in an XML file, filled by data providers, and validated by members of the logging team responsible for the MS processes.
- JAPC to logging signal mappings defined in an XML file, also filled by data providers, and validated by members of the logging team responsible for the MS processes.

This approach was quite heavy for the data providers, and susceptible to errors due to the need to manually ensure the consistency across the three separate configuration components. Flexibility was also lacking, since adding a new logging signal or modifying an existing one implied modifying 2 XML files, committing them in the code repository, and re-releasing and re-starting the relevant MS process. Although this approach was used successfully during 5 years, there was definitely scope to improve.

A FEDERATED APPROACH

The CERN accelerator control system is fully data-driven, based on several distributed Oracle databases, which collectively cover all data domains such as layout, asset management, controls configuration, and operational data. As part of a federated approach to data management, each database is considered the source for data in a particular domain, and data synchronization procedures are in place – either executed automatically or manually – to propagate necessary data to dependent databases [4].

The Controls Configuration database (CCDB) is the source for all data that describes the configuration of the control system. This includes amongst many other things the definition of multiple device-property models that describe deployed devices, and their available properties and fields, which can be subscribed to and/or modified [5]. The device-property data can be quite dynamic, mainly due to evolving requirements or to follow hardware or software developments.

The Goal

To achieve the goal of ensuring a smooth uninterrupted running of the MS, it is essential to have a consistent measurement configuration that is fully synchronized with the CCDB source data. That is to say that any changes to CCDB device-property data (such as device renames, or changes of properties and fields) should automatically be crosschecked against the MS configuration. Compatible changes should be immediately propagated to the active MS configuration, while non-compatible changes (such as removal of a property or field) should generate an alert.

On the Road

In order to reach the aforementioned goal, the first step was to provide an infrastructure inside the MDB to define the *complete* MS configuration metadata using a relational model. This metadata needed to include not only the existing definitions of the signals to be logged, but also the JAPC subscription parameters (device-property), and the mapping between the domains, as mentioned above.

Such an approach was deployed during 2010, making it possible to simplify the code and configuration of the many MS data logging processes, as well as bringing other advantages:

- A MS data logging process can now simply retrieve its full configuration by calling a single method with its process name as input.
- Requestors for data logging only need to make a single request to configure logging with a common Excel template for defining all parameters. This eliminates the possibility of having inconsistencies in the metadata describing JAPC subscriptions, logging signals, and mappings, both at the time of initial definition, and for subsequent updates to the device-property model data.
- The time required to configure logging of additional signals, or modify existing configurations is reduced significantly.
- With a single point of definition for MS process configuration, using an Oracle relational database, it is possible to envisage the simple propagation of CCDB device-property model data to the MDB using custom PL/SQL procedures.

ONLINE UPDATES

Previously, whenever requests to modify MS process configurations had been fully treated, it was necessary to restart the process concerned in order to apply the changes. This implies stopping all existing JAPC parameter subscriptions, and then restarting them based on the latest configuration. Considering that on average each MS process treats data for tens of thousands of signals, this approach was both time consuming and implied undesirable data loss. For example, to add the logging of a single new signal, it would be necessary to restart the related MS process and incur a non-negligible downtime of logging of many of the other existing signals already being logged.

To minimize such data loss and make configuration updates more flexible, an online update mechanism was put in place (see Fig. 2):

- A virtual device-property was defined in the CCDB to indicate if a MS process configuration has been modified.
- The MS process code was adapted to subscribe to this property at start-up via JAPC.
- Once a MS process configuration is modified in the MDB, an updated value for the aforementioned virtual device-property is published directly from the MDB using Oracle Advance Queuing [6].
- When a MS process receives a notification that its configuration has been modified, it retrieves the new configuration from the MDB, compares with its current configuration, and then makes the minimum number of JAPC re-subscriptions necessary in order to get an up-to-date process configuration running.

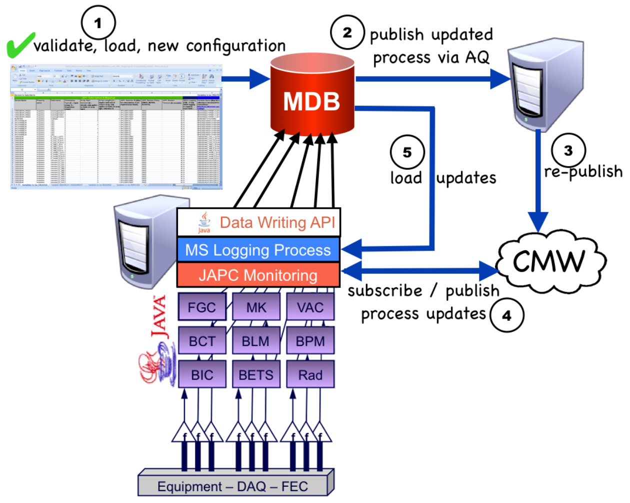


Figure 2: Online updates of MS configuration.

In summary, modifications to MS configurations can now be made online, and no longer imply data loss for other signals.

RESOURCE OPTIMIZATION

The MS writes up to 5.4 billion records per day (i.e. ~270GB) to the MDB. In order to optimize resources, some additional data driven features have been integrated into the MDB data writing API:

Log-on-Change

Depending on device-property structures and / or other requirements (e.g. continuous content for Fixed Displays), it is quite common for unchanging values to be published at relatively high frequencies. Examples include status flags, temperatures, counters etc. In most cases it is sufficient to log these values only when they change.

To implement this functionality, additional attributes were defined in the MDB metadata for each signal:

- A flag to turn on/off the on-change comparison.
- An integer value to specify a precision for on-change comparison.
- A flag to qualify if the precision refers to a number of digits left or right of the decimal point, or to a number of significant figures.
- An interval, after which to force logging of a published value even if it is not considered as changed.

These new attributes were exposed in the MDB data loading API, which then caches the last logged value for each signal which has on-change logging enabled. The new attributes are used to establish if new values can be considered to have changed sufficiently with respect to the last logged value. Newly published values are then only published on-change, or if the specified interval since the last logged value has elapsed.

Value Rounding

Numeric data is stored in the MDB using the Oracle NUMBER data type, which requires a variable number of bytes according to the scale and precision of the numeric values to be stored.

Many device-property field values are published with a much higher precision than the real physical accuracy (e.g. due to calculations prior to publishing), or what is actually required. An example would be a thermometer voltage-to-temperature conversion with 10 digits of precision.

In the same manner as for the on-change logging described above, additional metadata attributes were defined and exposed per signal – allowing the MDB data loading API to round values before writing to the MDB, thus saving storage, and improving I/O response times.

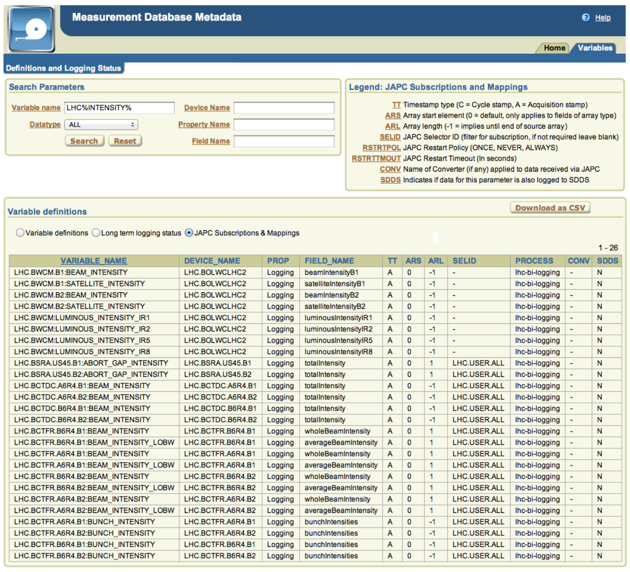
Not Just Resources

It is worth noting that while the described optimizations reduce the amounts of network activity, processing to load data, and storage required, the main benefit is actually for the users of the data, since having only significant data logged improves retrieval times and facilitates data analysis.

A CLEAR VIEW

With the configuration of MS processes and MDB signals now fully defined within the MDB, it was necessary to provide data providers, consumers, and MS experts with a means to visualize this configuration data.

To satisfy this requirement, a Web interface was developed using Oracle Application Express (APEX) to search and report on the current MS configuration (see Fig. 3).



Search Parameters

Variable name: LHC%INTENSITY%
Datatype: ALL
Device Name:
Property Name:
Field Name:

Legend: JAPC Subscriptions and Mappings

TT	Timestamp type (C = Cycle stamp, A = Acquisition stamp)
ARS	Array start element (0 = default, only applies to fields of array type)
ARL	Array length (-1 = implies until end of source array)
SELID	JAPC Selector ID (filter for subscription, if not required leave blank)
RSTRTPOL	JAPC Restart Policy (ONCE, NEVER, ALWAYS)
RSTRTTMOUT	JAPC Restart Timeout (In seconds)
CONV	Name of Converter (if any) applied to data received via JAPC
SDDS	Indicates if data for this parameter is also logged to SDDS

Variable definitions

( ) Variable definitions ( ) Long term logging status [✓ JAPC Subscriptions & Mappings]

1 - 26

VARIABLE_NAME	DEVICE_NAME	PROP	FIELD_NAME	TT	ARS	ARL	SELID	PROCESS	CONV	SDDS
LHC.BWCM.B1:BEAM_INTENSITY	LHC.BOLWCLHC2	Logging	beamIntensityB1	A	0	-1	-	lhc-bi-logging	-	N
LHC.BWCM.B1:SATELLITE_INTENSITY	LHC.BOLWCLHC2	Logging	satelliteIntensityB1	A	0	-1	-	lhc-bi-logging	-	N
LHC.BWCM.B2:BEAM_INTENSITY	LHC.BOLWCLHC2	Logging	beamIntensityB2	A	0	-1	-	lhc-bi-logging	-	N
LHC.BWCM.B2:SATELLITE_INTENSITY	LHC.BOLWCLHC2	Logging	satelliteIntensityB2	A	0	-1	-	lhc-bi-logging	-	N
LHC.BWCM:LUMINOUS_INTENSITY_IR1	LHC.BOLWCLHC2	Logging	luminousIntensityIR1	A	0	-1	-	lhc-bi-logging	-	N
LHC.BWCM:LUMINOUS_INTENSITY_IR2	LHC.BOLWCLHC2	Logging	luminousIntensityIR2	A	0	-1	-	lhc-bi-logging	-	N
LHC.BWCM:LUMINOUS_INTENSITY_IR5	LHC.BOLWCLHC2	Logging	luminousIntensityIR5	A	0	-1	-	lhc-bi-logging	-	N
LHC.BWCM:LUMINOUS_INTENSITY_IR8	LHC.BOLWCLHC2	Logging	luminousIntensityIR8	A	0	-1	-	lhc-bi-logging	-	N
LHC.BSRA.US45.B1:ABORT_GAP_INTENSITY	LHC.BSRA.US45.B1	Logging	totalIntensity	A	0	1	LHC.USER.ALL	lhc-bi-logging	-	N
LHC.BSRA.US45.B2:ABORT_GAP_INTENSITY	LHC.BSRA.US45.B2	Logging	totalIntensity	A	0	1	LHC.USER.ALL	lhc-bi-logging	-	N
LHC.BCTDC.A6R4.B1:BEAM_INTENSITY	LHC.BCTDC.A6R4.B1	Logging	totalIntensity	A	0	-1	LHC.USER.ALL	lhc-bi-logging	-	N
LHC.BCTDC.A6R4.B2:BEAM_INTENSITY	LHC.BCTDC.A6R4.B2	Logging	totalIntensity	A	0	-1	LHC.USER.ALL	lhc-bi-logging	-	N
LHC.BCTDC.B6R4.B1:BEAM_INTENSITY	LHC.BCTDC.B6R4.B1	Logging	totalIntensity	A	0	-1	LHC.USER.ALL	lhc-bi-logging	-	N
LHC.BCTDC.B6R4.B2:BEAM_INTENSITY	LHC.BCTDC.B6R4.B2	Logging	totalIntensity	A	0	-1	LHC.USER.ALL	lhc-bi-logging	-	N
LHC.BCTFR.B6R4.B1:BEAM_INTENSITY	LHC.BCTFR.B6R4.B1	Logging	wholeBeamIntensity	A	0	1	LHC.USER.ALL	lhc-bi-logging	-	N
LHC.BCTFR.B6R4.B1:BEAM_INTENSITY_LOBW	LHC.BCTFR.B6R4.B1	Logging	averageBeamIntensity	A	0	1	LHC.USER.ALL	lhc-bi-logging	-	N
LHC.BCTFR.A6R4.B1:BEAM_INTENSITY	LHC.BCTFR.A6R4.B1	Logging	wholeBeamIntensity	A	0	1	LHC.USER.ALL	lhc-bi-logging	-	N
LHC.BCTFR.A6R4.B1:BEAM_INTENSITY_LOBW	LHC.BCTFR.A6R4.B1	Logging	averageBeamIntensity	A	0	1	LHC.USER.ALL	lhc-bi-logging	-	N
LHC.BCTFR.B6R4.B2:BEAM_INTENSITY	LHC.BCTFR.B6R4.B2	Logging	wholeBeamIntensity	A	0	1	LHC.USER.ALL	lhc-bi-logging	-	N
LHC.BCTFR.B6R4.B2:BEAM_INTENSITY_LOBW	LHC.BCTFR.B6R4.B2	Logging	averageBeamIntensity	A	0	1	LHC.USER.ALL	lhc-bi-logging	-	N
LHC.BCTFR.A6R4.B2:BEAM_INTENSITY	LHC.BCTFR.A6R4.B2	Logging	wholeBeamIntensity	A	0	1	LHC.USER.ALL	lhc-bi-logging	-	N
LHC.BCTFR.A6R4.B2:BEAM_INTENSITY_LOBW	LHC.BCTFR.A6R4.B2	Logging	averageBeamIntensity	A	0	1	LHC.USER.ALL	lhc-bi-logging	-	N
LHC.BCTFR.A6R4.B1:BUNCH_INTENSITY	LHC.BCTFR.A6R4.B1	Logging	bunchIntensities	A	0	-1	LHC.USER.ALL	lhc-bi-logging	-	N
LHC.BCTFR.A6R4.B2:BUNCH_INTENSITY	LHC.BCTFR.A6R4.B2	Logging	bunchIntensities	A	0	-1	LHC.USER.ALL	lhc-bi-logging	-	N
LHC.BCTFR.B6R4.B1:BUNCH_INTENSITY	LHC.BCTFR.B6R4.B1	Logging	bunchIntensities	A	0	-1	LHC.USER.ALL	lhc-bi-logging	-	N
LHC.BCTFR.B6R4.B2:BUNCH_INTENSITY	LHC.BCTFR.B6R4.B2	Logging	bunchIntensities	A	0	-1	LHC.USER.ALL	lhc-bi-logging	-	N

Figure 3: Measurement Service configuration report.

The report currently includes data acquisition configuration per MS process, applied data filtering criteria, and long-term storage status. In addition, the interface allows data providers to download existing configurations to a standard Excel template, from where configurations can be modified and/or extended and then submitted to service administrators for validation and integration into the system.

NEXT STEPS

In order to continue on the road to a fully federated data management solution, the links to the CCDB device-property model data need to be enforced, with automatic checks, updates, and notifications in place as outlined above.

The interfaces for browsing MS configuration data, and CCDB device-property data should be linked in both directions. This will simplify the process of browsing and configuring the MS for data consumers and providers alike. For example, it will be possible when browsing the device-property data to see which property fields are logged, and when browsing the MS configuration, to see the underlying data types of the device-property fields, or additional fields which may need to be logged.

Flexible Data Extraction

The Logging Service data extraction API currently only supports extraction of data based on signal names. However, it is foreseen to reuse the MS configuration data that maps JAPC parameters to logging signals within the data extraction API, thus allowing the extraction of logged data based on JAPC parameters. This will give users of logged data a more flexible means to retrieve data and facilitate the development of certain types of applications such as fixed displays that can optionally display a historical set of data.

SUMMARY

With respect to the initial implementation, the current infrastructure has already significantly simplified and improved the robustness of the process of configuring the Measurement Service.

The Measurement Service is well on the road to federation in the distributed database environment of accelerator controls. The foundations have been established, with all configuration data now centralized in a relational Oracle database, and the next steps to achieve the desired goals are foreseen.

REFERENCES

[1] M. Gourber-Pace et al., "Status Report of The Measurement Service for the CERN Accelerator Logging", ICALEPCS'09, Kobe, Japan, October 2009, TUP106.
[2] C. Roderick and R. Billen, "Capturing, Storing and Using Time-Series Data for the World's Largest Scientific Instrument", November 2006, CERN-AB-Note-2006-046 (CO).
[3] V. Baggiolini et al., "JAPC - the Java API for Parameter Control", ICALEPCS'05, Geneva, Switzerland, October 2005, TH1_5-8O.
[4] R. Billen et al., "Accelerator Data Foundation: How It All Fits Together", ICALEPCS'09, Kobe, Japan, October 2009, TUB001.
[5] Z. Zaharieva et al., "Database foundation for the Configuration Management of the CERN Accelerator Controls Systems", ICALEPCS'11, Grenoble, France, October 2011, MOMAU004.
[6] K. Kostro et al., "On-change Publishing of Database Resident Control System Data", ICALEPCS'09, Kobe, Japan, October 2009, TUP013.