ATLAS Level-1 Calorimeter Trigger
Software Note 005

8 December 2001
Version Draft 0.2a

Configuration Database

Murrough Landon

1 Introduction

This document discusses the classes we need in the database schema to implement our configuration, calibration and trigger menu data.

The list of all the parameters required to configure the level 1 calorimeter trigger [1] has been already described in [2]. In this document we propose an object model to implement this. We also define the requirements of the data access libraries we will use to interface the database to higher levels of software.

1.1 Categories of Data

The data we need to describe and configure the trigger falls into a few different categories which are distinguished by the nature of the data and how it changes with time. The main categories are summarised here and discussed in more detailed in subsequent sections.

- **Hardware and Software Configuration**: this category of data is concerned with the description of the hardware setup and its interconnections; also with the collection of programs used to configure and monitor the system. The database schema is largely that provided as part of the Online Software Configuration Databases [3] component – with some extensions to the hardware schema specific to the calorimeter trigger. The software schema requires little or no change.

- **Trigger Menu**: this describes the physics choices, ie trigger thresholds and algorithms, to be used for a particular run. There may be many trigger menus prepared offline and distributed to the trigger systems. Of these, one will be selected at run time.

- **Calibration Data**: this includes the energy calibration, timing calibration of the input signals and their pulse shapes, also internal timing of links between the processor subsystems, readout pipeline latencies etc. These data will be derived from a number of different calibration procedures which may be run regularly or occasionally. All calibrations will be stored in the offline database, but at run time only the latest one is ever used.

- **Run Parameters**: there may be some operational parameters for which the default values (taken from the above databases) can be overridden at run time. For example, the amount of readout data may be temporarily increased for detailed monitoring, etc.

2 Requirements and Use Cases

The design of the database schema, the organisation of the database into separate files and the implementation of the data access libraries (DALs) have to satisfy the following use cases:

- load the whole system during the standard run control state sequence from "initial" to "running".
- load a partial system, eg a subset of the crates or crates with a subset of modules.
- stop a run and a start a new one loading a newly selected trigger menu and using any new calibrations which may have appeared (ie these operations must be completely handled just at the transition between "configured" and "running").
- take a calibration run and produce new calibration files.
- view/edit the calibration database.
- create test calibration databases, eg with all channels set to the same value.
- store/retrieve calibration data in/from the offline conditions database.
- change the trigger menu or produce a new trigger menu offline and distribute it to the processor system.

3 Data Access Libraries

It is expected that the online databases will typically be kept in OKS files. The OKS database includes both the schema and the data, however the OKS classes and objects are instances of generic OksClass and OksObject objects which hold descriptions of the classes and their instantiations.

To have the OKS schema and data available in a corresponding set of C++ classes and objects, we need to implement a data access library (DAL). This would provide a much more convenient access to the database.

The other advantage of using a DAL rather than accessing OKS classes directly is that the same API can be presented to higher levels of software if the underlying persistent database manager (ie OKS) is changed in the future. The same API can also be used offline where the data may be accessed from the offline "Conditions Database", or in the case of the trigger menu, even from an ATHENA "jobOptions" file.

In the Online software, the configuration database DALs provide read only access to the data. This may be fine also for the trigger menu, but for calibration data it may be more convenient for the calibration programs if the corresponding DAL can also create the OKS objects.

The important point is that all our other software sees the database as a collection of instances of a stable set of run time classes. The details of how these run time objects are read and saved to persistent form should be hidden from the rest of the system. This also applies to mechanisms whereby online databases are archived and retrieved to and from the offline database.

4 Database Organisation

The calorimeter trigger comprises three subsystems each of which is implemented in a number of VME crates. Each crate will have its own CPU and the whole system will be controlled by a small number of workstations (eg Linux PCs).

For the hardware and software configuration, we will follow the proposals of the Online software group. One or more OKS files will contain the hardware description: perhaps one file per subsystem and one file for common items such as worstations. A separate file will contain descriptions of the software. Other files will describe the DAQ partition(s) which draw together the required hardware and software setup.

The calibration data makes up the largest data volume. It is proposed that this is split into separate files for each crate and for each major type of calibration. Separation by crate means that each crate CPU only needs to load the data pertaining to its crate. Separate by type of calibration makes sense because the different calibrations will be run with different frequencies: some every day, others weekly, monthly, or yearly. It is possible that OKS will not be the best system for storing large amounts of data. If so, some other scheme can be used provided that the run time view remains the same.

The trigger menu is generated offline and will need to be distributed to the processor crates by some mechanism. This may be by copying OKS files, or possibly some dynamic database access will be implemented. If the trigger menu

is implemented in OKS, we expect a single file per trigger menu – at least for those parts relevant to the level 1 calorimeter trigger.

Run parameters will typically only be set and accessed via the Online software Information Service component. However the complete set of level 1 calorimeter trigger run parameters may be backed up to a single OKS file.

5 Hardware Configuration

The Online software configuration databases provide a "core" schema describing the basic infrastructure of crates and modules, etc. It will (probably) be useful for us to extend this be defining our own Crate and Module subclasses.

Examples of additional attributes we might need to add to the default classes include:

- links to calibration data for each object: ie connection between the hardware schema and the calibration data schema.
- geographical addressing of our crates (eg in the PreProcessor and ROD crates where this is not completely determined from the backplane).
- mapping from the electronics to the detectors and/or (phi,eta) coordinates.
- concept of FPGA programs: ie add firmware to the software schema.
- default values of run parameters?

5.1 Naming Conventions

In OKS, and other OO databases, objects must have a unique identifier. In OKS this identifier is a string. We will need a convention for naming each object (of the same class) uniquely. In particular, modules in the same slot in different crates need different identifiers. So a scheme which combines crate and module identifiers is required.

Although we have a proposed labelling scheme [4], this is not particularly suitable for crates. So for the purposes of the database, the following is suggested instead.

Crate identifiers could use the two or three character subsystem string, with a single digit suffix for the crate number within the subsystem. Eg `pp0` to `pp7`, `cp0` to `cp3`, `jep0` to `jep1`, `rod0` for the ROD crate(s) and `ttc` for the TTC crate.

Module identifiers could then be of the form `ccc-mmmnn`, where `ccc` is the above crate identifier, `mmm` is the standard three character module number (maybe using `cprod` and `pprod` instead of `crd` and `prd`?) and `nn` is the logical number of that kind of module within the crate. Examples might be `pp2-ppm11`, `pp3-pprod1`, `cp0-cpm5`, `jep1-jem15`, `cp3-cmm1`, `pp5-tcm`, `rod0-cprod4`. Although these are rather long, some modules appear in more than one subsystem, so either the full subsystem name must appear, or the crates must be numbered eg tt 0-`14` across subsystems which is not very memorable.

If subcomponents of modules are eventually entered into the configuration database (ie as opposed to HDMC parts files) then the above convention will need extending, presumably adding `-sssnn` for subcomponent type (`sss`) and number (`nn`).

6 Calibration

The overall schema for the calibration data is shown in figure 6. The major part of the calibration data is required in the preprocessor system. A more detailed schema for that is given in figure 6.

6.1 Overview

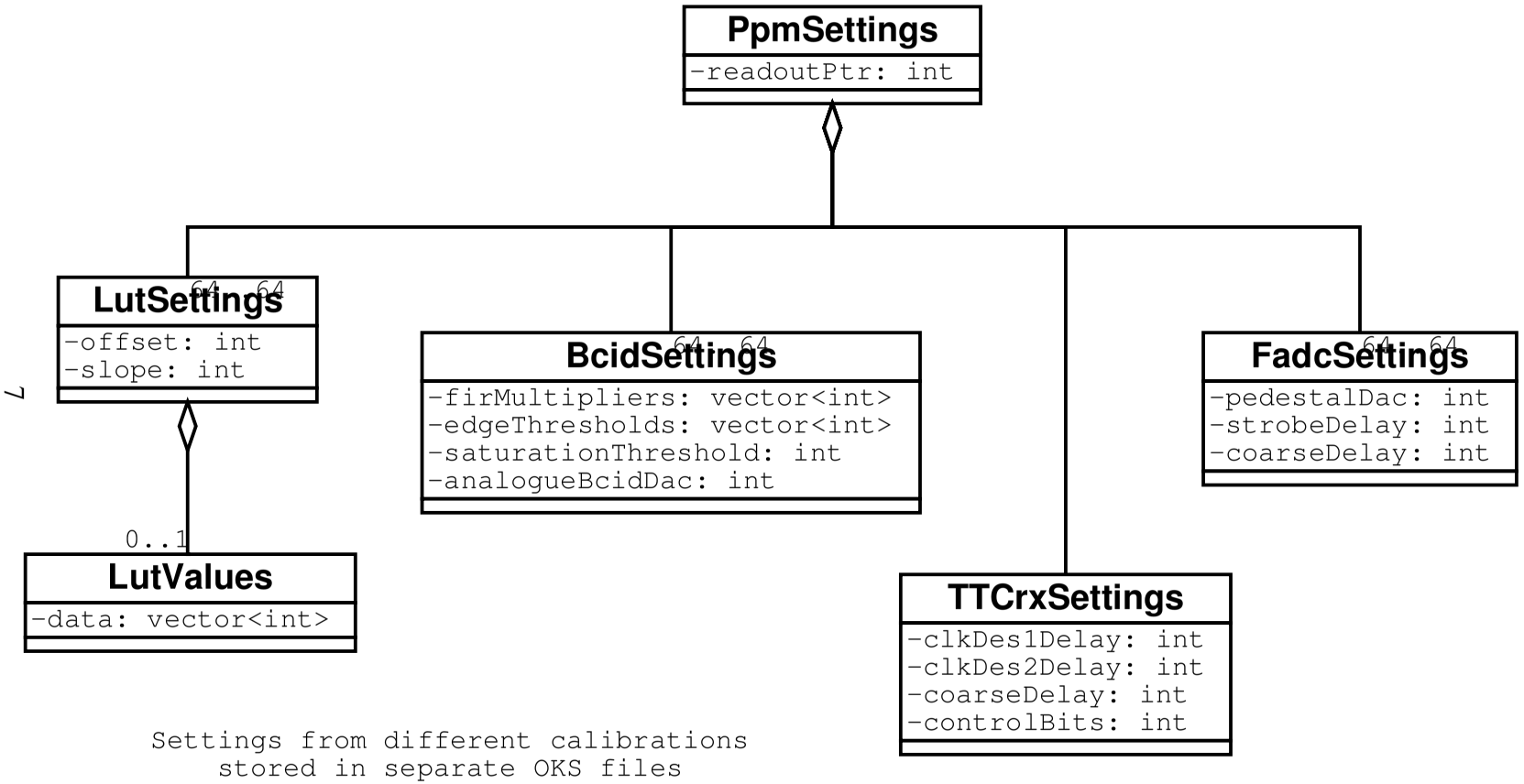
One `CalibConfiguration` object is the access point for all the calibration data in the crate. In the DAL this is the only object which uses OKS. It is responsible for creating all the other objects. This is the only class which would need replacing in the offline environment – all the other classes could remain unchanged.

There is one `XxxSettings` object for each module. This can be linked to the equivalent module object in the hardware configuration database. In OKS it would make sense for them to share the same unique ID (this only has to be unique within one class [**check!**]).

The settings object for each module then contains a number of objects with the data to be loaded into each distinct device on the module. It is expected that the software will implement hardware access to each device in separate objects of an appropriate class, so this division of calibration data into one chunk per device is probably the most appropriate – rather than an object per channel or per module.

However the PPrAsic needs to be downloaded with several different calibrations per channel, so here there is some further subdivision.

Overall calibration database schema. PpmSettings is shown in more detail in 6



Schema for PreProcessor calibration data.

The `CalibConfiguration` and module settings objects would probably be kept in one file, separate from the device settings. This file should be fairly static, while the device data files can change - as long as the OKS unique IDs of the device data objects in new calibration files match those expected in the top level file.

The following sections describe some of the calibration data classes in a bit more detail.

6.2 Energy Calibration

Every channel in the PPMs contains a 1024 entry lookup table (LUT) for the energy calibration. This can be filled by downloading all 1024 values. Alternatively, if the calibration is linear, the PPrASIC can fill the LUT itself, given an offset (pedestal) and a slope.

To minimise the data value, the LUT for a single channel should therefore be implemented as a simple object for the linear case with a link to a second object containing the explicit LUT values only if required.

6.3 BCID Settings

6.4 Input Timing Calibration

6.5 Internal Timing Setup

6.6 Readout Settings

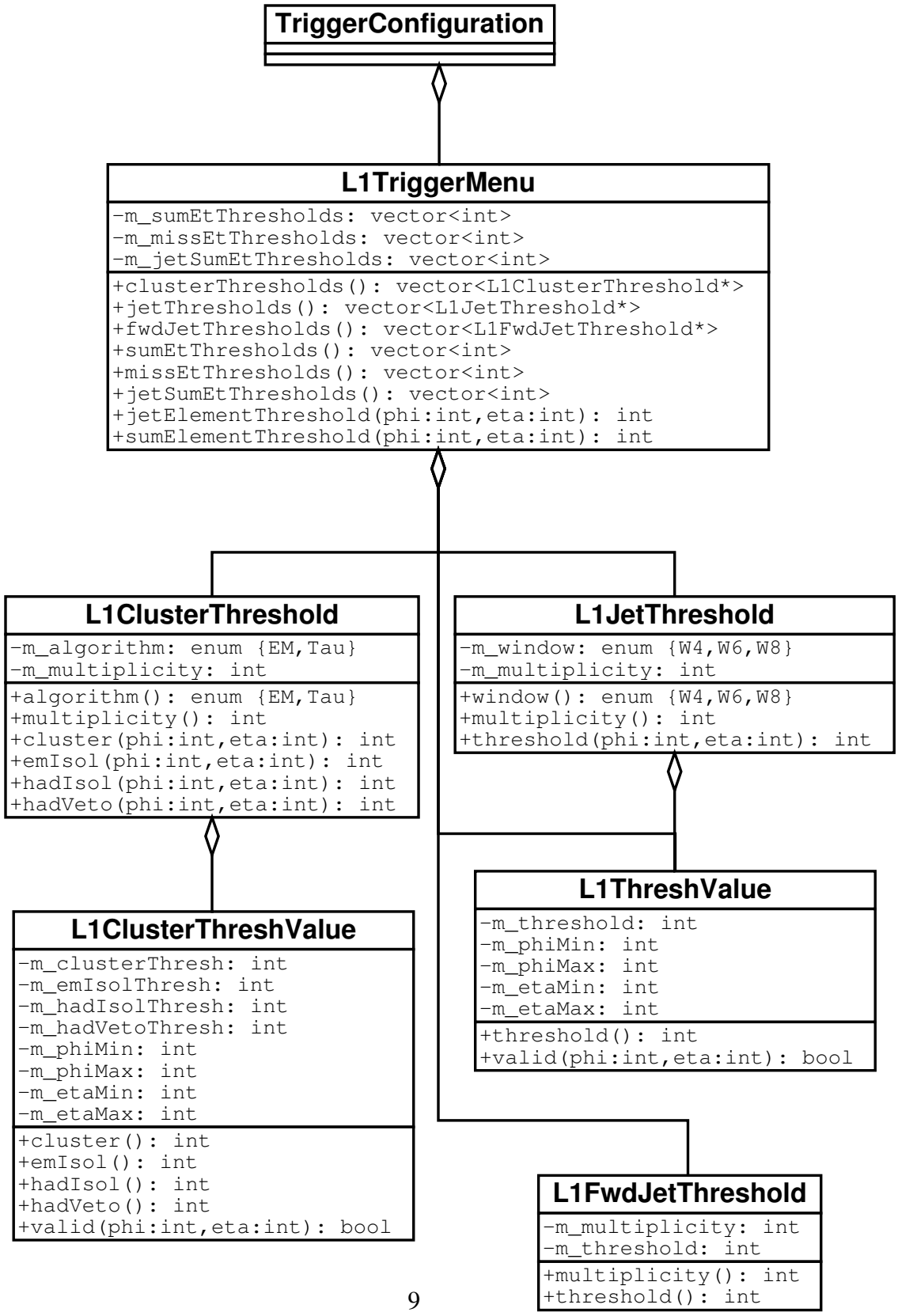
7 Trigger Menu

A partial schema for the level 1 calorimeter trigger menu is shown in figure 7.

7.1 CP Thresholds

Each CPM needs to be told which algorithm to run for each of the 16 configurable e/gamma or tau/hadron thresholds. This must be the same for all CPMs in the system. There should therefore be a single object (`L1ClusterThreshold`) per threshold for the whole system.

Each of the 16 thresholds actually consists of four separate values: the e/gamma or tau/hadron cluster threshold, the EM layer isolation threshold, the Hadronic layer isolation threshold and the Hadronic layer core veto threshold. The latter is only used in the e/gamma algorithm. Although in practice these values will be the



Level 1 Calorimeter Trigger Menu.

same for all channels of all CPMs, in principle they may vary across the eta range, and possibly to compensate for dead calorimeter channels.

The corrections for dead channels and transition regions between the different calorimeters could be applied at the point of loading the thresholds, or by the program which creates the threshold objects from the trigger menu. Even in the former case, there may be physics reasons in the trigger menu leading to an eta dependent threshold. So it is probably best to assume that the trigger menu may generate more than just a single value per threshold.

The complete collection of threshold data will need to be stored in the conditions database. To avoid storing excessive quantities of identical data, the threshold definition should probably be implemented as a list of threshold values and the {phi,eta} regions in which they apply. However the API for the `L1ClusterThreshold` should hide the implementation and the `L1ClusterThreshValues` should not be visible to the rest of the software.

7.2 JEP Thresholds

Similar considerations apply to the Jet thresholds. However here it is not so clear that the algorithm choice (jet window size) has to be constant over the whole system for a given threshold.

The Forward Jet algorithm is not yet certain and it is not yet clear if it will be necessary to describe it in a separate object or if the `L1JetThreshold` class can be used for forward jets as well.

The JEP system includes two thresholds which are applied in common to all jet elements and again at the entry of the jet elements to the total Et summing tree. Although they are common to all `L1JetThreshold`s they may still vary across {phi,eta} space.

The various Et thresholds are single system-wide values and can therefore be attributes of the menu object itself.

References

[1] ATLAS Level 1 Calorimeter Trigger: home page
`http://hepwww.pp.rl.ac.uk/Atlas-L1`

[2] Configuration Data for the ATLAS Level 1 Calorimeter Trigger (ATL-DA-EP-02)
`http://edmsoraweb.cern.ch:8001/cedar/doc.page?document_id=111349`

[3] ATLAS Online Software Configuration Databases
http://atddoc.cern.ch/Atlas/DaqSoft/components/configdb

[4] ATLAS L1Calo Trigger Labelling Guidelines
http://www.hep.ph.qmul.ac.uk/l1calo/doc/pdf/labelv032.pdf