Module Services

Murrough Landon – 6 September 2001

http://www.hep.ph.qmw.ac.uk/~landon/talks

Overview

- General Aims
- Overall Schema
- Details
- Questions

Module Services: Aims

General Idea

- Will be the main way other software accesses the modules
- Provide a high level set of services for each of our modules and their main, complex, subcomponents
- Hide details of internal register and memory structure
- Maybe modules permit some access to major subcomponents?

Interfaces

- Based on HDMC Module and SubModule classes
- Modules will need to be given a Bus and ideally some connection to the PartManager (to be determined)
- Close connection between configuration database objects and Module Services configuration methods
- Run control states handled by run control package (calling module services to execute transition actions)
- Modules return status objects for hardware monitoring package
- Will test programs need more detailed access to module components?

Overall Schema

Expected Classes

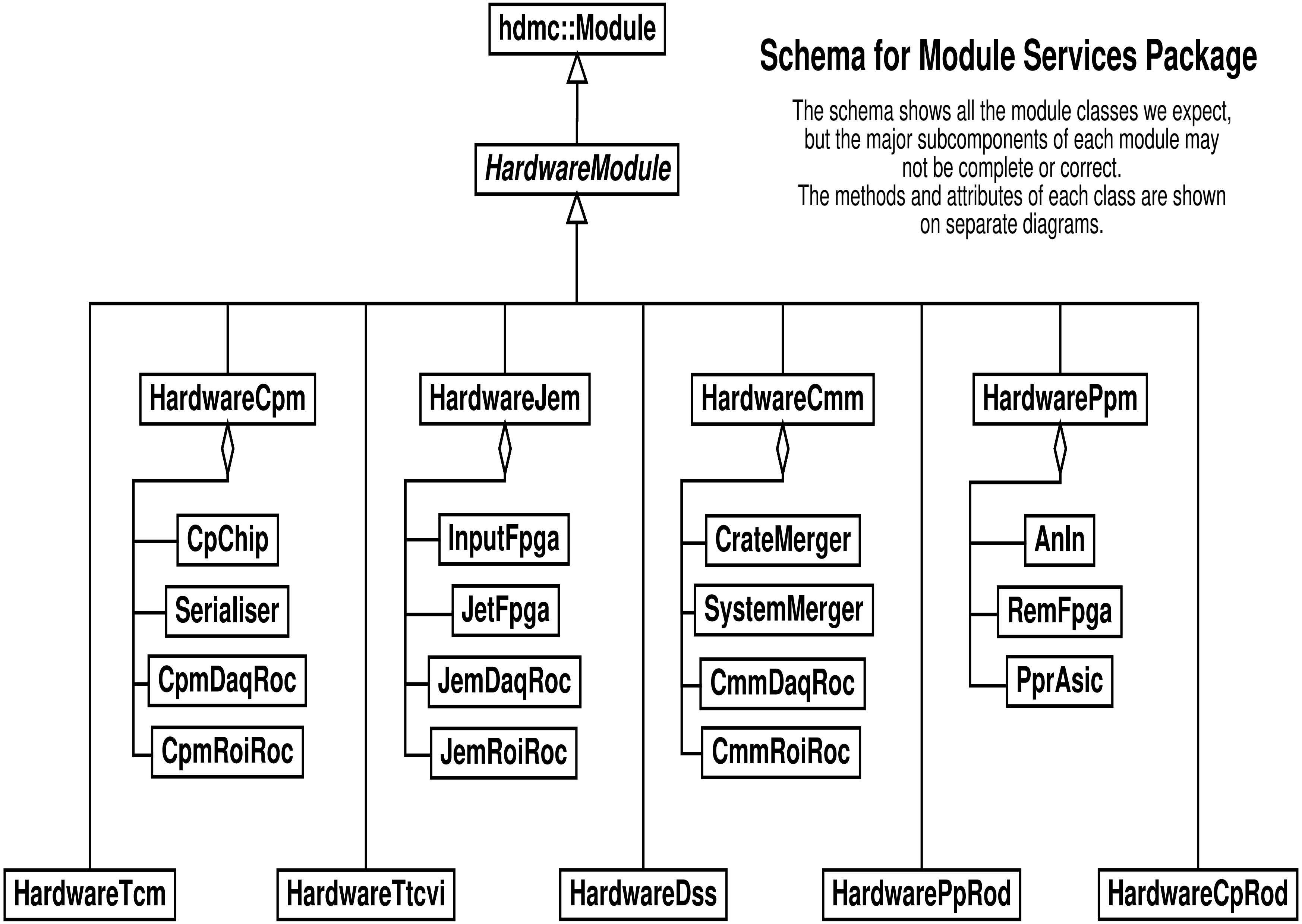
- May be useful to have single superclass for our modules? Ie other than HDMC Module class.
- Classes for our production modules (TCM, PPM, PPROD, CPM, JEM, CMM, CPROD)
- Also classes for test modules and external service modules (DSS, TTCvi, BUSY)
- Classes for major subcomponents - typically FPGAs, ASICs, etc

Module Services vs HDMC Hardware Parts

- Keep basic Parts in HDMC Hardware Access Library
- Use Module Services package for components which use objects from other packages (eg database)

Schema for Module Services Package

The schema shows all the module classes we expect, but the major subcomponents of each module may not be complete or correct.
The methods and attributes of each class are shown on separate diagrams.



Details: Typical Services (1)

Caveat

- I have mostly been thinking of configuring modules...
- ...so probably many gaps related to actually **using** the modules.

Creation

- Constructors: taking HDMC Bus and database parameters.
- Some thought required on convenient database object to describe each modules address.

Configuration

- Several methods to load calibration data.
- Typically aim for one method per type of calibration (eg timing, energy) per type of subcomponent (eg TTCrx, Serialiser, CpChip).
- Method(s) to load test vectors – depending on how clever we are in describing test vectors.
- Method(s) to capture configuration from hardware to database?

Details: Typical Services (2)

Status/Readout

- Method(s) to collect status information, eg link and parity errors, rates histograms etc. Should return objects.
- Method(s) to read back loaded test vectors.
- Collect event from spy buffer(s).
- What about other internal data??

Operation Modes

- Modules may have several operation or test modes, even with standard FPGA load. Need methods to set and query these modal states. Eg setting BCMUX modes on all channels at each end of the links. (NB Im not thinking of run control states here).
- Method to reset (at least control registers) to known state.

Miscellaneous

- Software L1Accept and software versions of TTC broadcast commands (though some of these will be setting operation modes).

Details: FPGA loading

HardwareModule base class

- Most of our modules (though not external modules) have FPGAs.
- Is it worth making FPGA loading a common base class function?
- Requires access to FPGA Parts of subclasses...
- ...and would be unused for several of our modules.

FpgaSpec

- Devise database description of how an FPGA should be loaded.
- Need path to a file and version number/string reported by a register in the loaded FPGA to check. Descriptive name for each FPGA type (eg Serialiser) also useful.
- Database could have a collection of all the FPGA programs relevant to a given type of module.

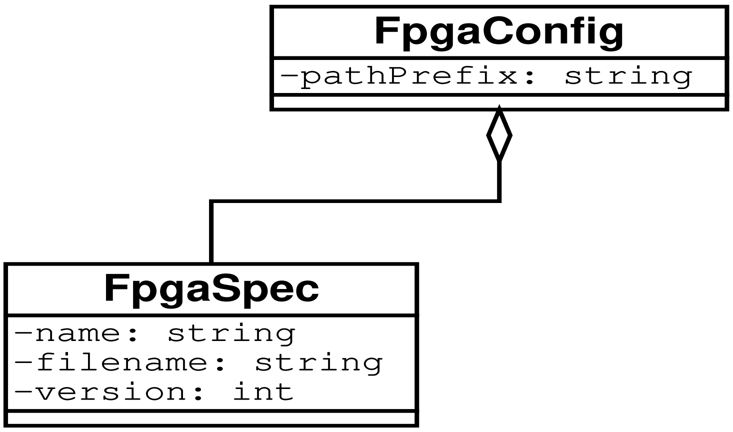
HardwareModule Class

Most of our modules contain FPGAs,
so a method to reload FPGA code into
on board flash memories is useful.
The method is passed an object structure
obtained from the database which specifies
the pathname of each FPGA program.
Can use the same object structure to
return currently loaded FPGA versions
(though some members would be unused).

HardwareModule

```
+HardwareModule(bus:Bus*,db:Dbmodule*)
+loadFpgaCode(db:FpgaConfig*): void
+getFpgaVersions(): FpgaConfig*
```

Q: if loadFpgaCode() is a superclass method
then subclass FPGA parts must be accessible
to the superclass...
...so each subclass needs to override some
virtual superclass methods allow access
to its FPGA parts
Maybe it would be easier to just implement
loadFpgaCode() separately in each subclass?



HDMC has some support for loading FPGAs. Probably needs
extending to cover different types of FPGA and different
ways they can be loaded in the different types of module
(ie how registers are accessed to do it).

Details: Various Modules

CPM and JEM

- Not sure if readout controllers are complex enough (from outside view) to require separate subcomponents. But probably sensible.
- Most configuration methods are fairly straightforward decomposition of calibration and trigger menu objects to the relevant subcomponents.

RODs

- They have TTCrx chips – but arent sensitive to delay settings?
- Little to configure? Apart from source ID?

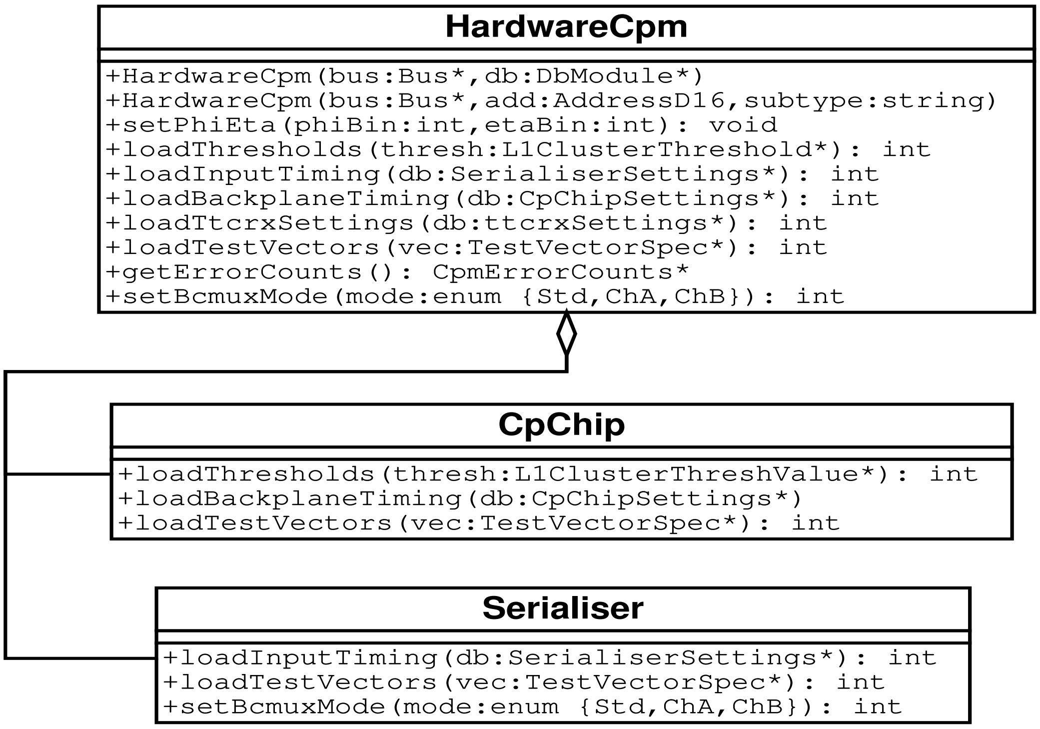
CMM

- Subclass of SystemMerger specific for Et and Jet (with method to load thresholds which arent needed for CP variant).

PPM suggestions

- Diagrams even more sketchy than the others...
- Mapping of calibration data to subcomponents may not be optimal.

HardwareCpm Class



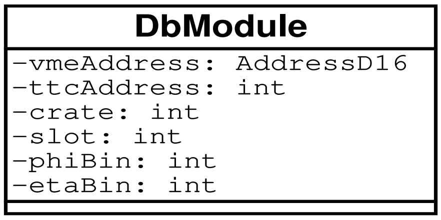
Creation: need some HDMC info (bus, partmgr?), also address etc.
Q: how to cope with module subtypes (eg different FPGA loads).
Should we have different class variants which provide different services, or handle all in one class, accepting that some methods will be meaningless some of the time?

Initialisation: several methods passing objects typically read from databases (or created manually in test programs).
Also provide methods to return loaded data...
Present suggestions for thresholds imply that a CPM knows its phi,eta location. Maybe included in complex DbModule object?

Loading test vectors: pass "intelligent" object which specifies what to load where? Or do we need a bit more detail?

Status: method(s) to return summary of errors (eg link errs, parity errs, etc). Return single object which can be asked detailed or global questions (eg still no errors?)

Readout: dump internal data? Maybe for alternate CpChip debug program. Can we freeze and read other memories?

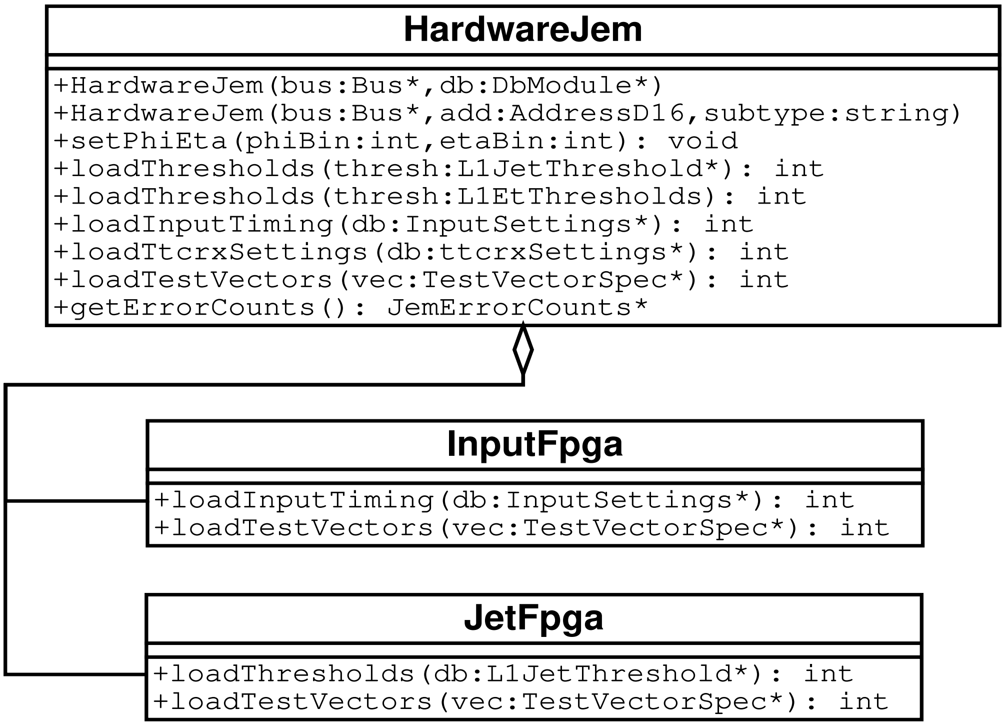


Many DbModule attributes are derivable from each other. Should probably define some unique address and provide methods to return the rest.

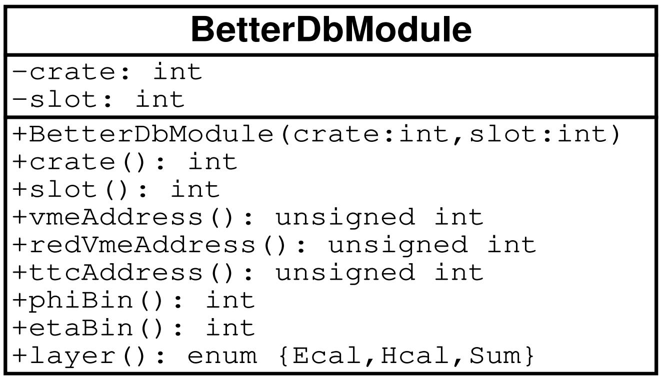
Probably want separate DbModule subclasses for each (or most) of our module classes to provide specific parameters (eg ROD ID, module subtypes?)

NB this example DbModule should be subclass of Online ConfdbModule (probably with more specific name).

HardwareJem Class



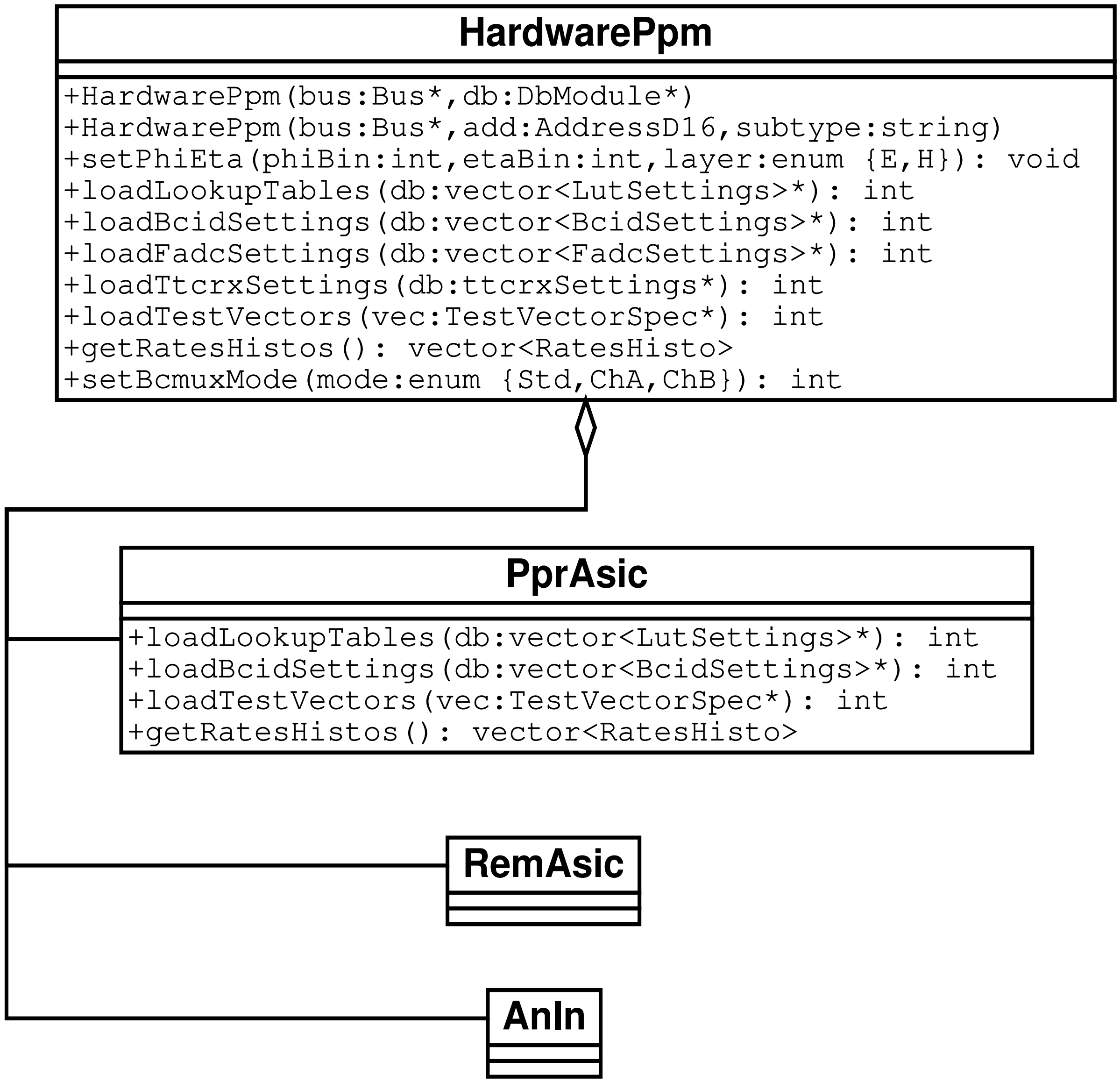
Rather similar to CPM obviously...
NB present TriggerMenu suggestions put all global Et thresholds in top level TriggerMenu class.
This is not so convenient for passing to each JEM so its may be better to change the TriggerMenu although that may not be so nice elsewhere?



May like separate subclass per module type for different address allocation algorithms?
Possibly TCM has different algorithms depending on which type of crate its in (adapter link card) though maybe methods to give VME address in standard crates and also reduced VME crates may be sufficient?

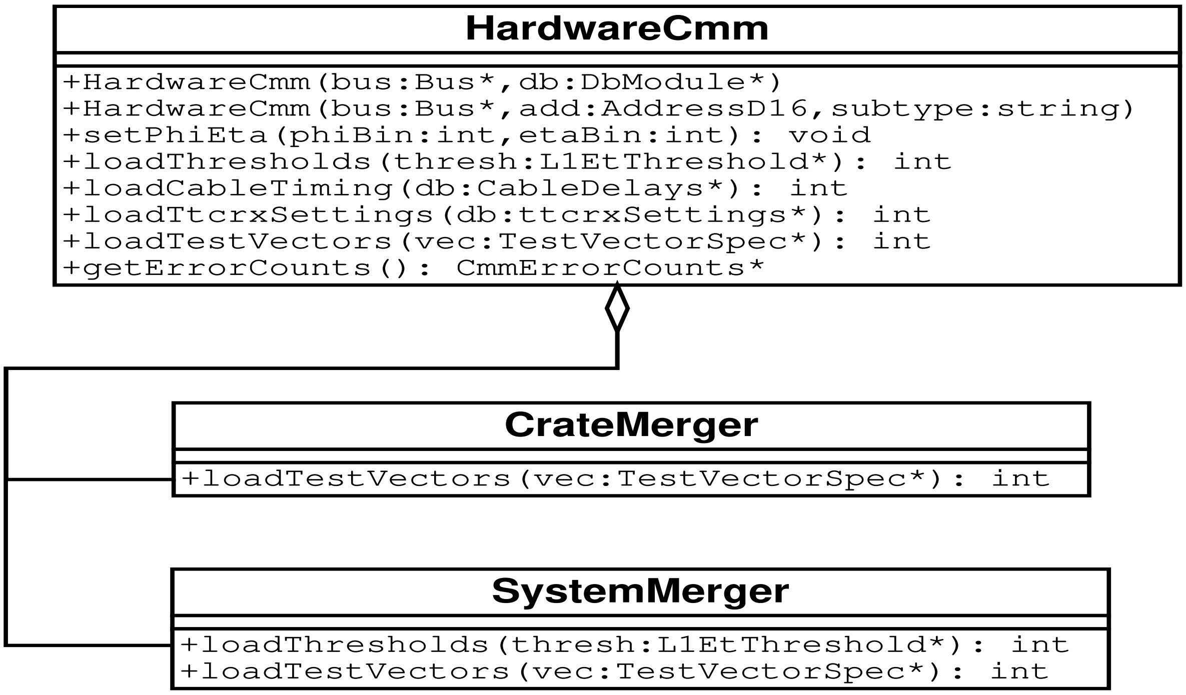
There may anyway be subclasses of DbModule for many types of module so we can include any extra parameters specific to that module.
For example CMM, ROD firmware variants, DSS daughtercard config, etc

HardwarePpm Class



Again some similarities with other modules...
Collections of calibration data may usefully be more sophisticated than simple vectors (eg in cases where many channels have the same calibration).
Im not sure where the FadcSettings need to go.
If both PprAsic and AnIn card need some parameters of the proposed FadcSettings structure, maybe it should be split?
Anyway the proposed PPM calibration data structures are doubtless missing many parameters.

HardwareCmm Class



NB trigger thresholds are only required in the SumEt and Jet variants of the CMM.
Doubtless many other operations and configuration information (eg suppressed channel masks etc).

TestVectorSpec

Want to give a single test vector specification to a module (possibly/probably subclasses of some TestVectorSpec class to cope with different types of component to be loaded?)

Also may like to have a specification for the test vectors to be loaded into a set of modules under test.

Want a simple way for module to dispatch relevant parts of the test vector spec/collection to its subcomponents.

Various ways to provide test vectors: single file/stream, multiple files/streams, generation on the fly according to some seed parameters or description, random data, and arbitrary mixtures of these (channel by channel?)

At the point of use, the test vector provides a simple byte stream (or stream of records N bytes wide).

Questions

How much should modules know?

- Should modules know about run state (and variations depending on different run types)?
- Should modules (and CpChips?) know their phi,eta address? (Implied by present suggestions for completely general eta and phi dependent trigger thresholds)
- Should they use some global mapping service?
- *My answers: no, yes, not sure*

Parts, PartManager, etc

- Should modules hard code creation of their subcomponents in their constructors or read them from parts files?
- How do we cope with modules which have variable subcomponents eg different pluggable daughtercards?
- How do we cope with modules which have very different FPGA loads (eg debug version of CpChip code). Separate classes with different sets of methods or single more complicated class?
- *My feelings: probably a bit of mix and match depending on each module. Some variations can be specified in the database, others may be better with separate classes.*

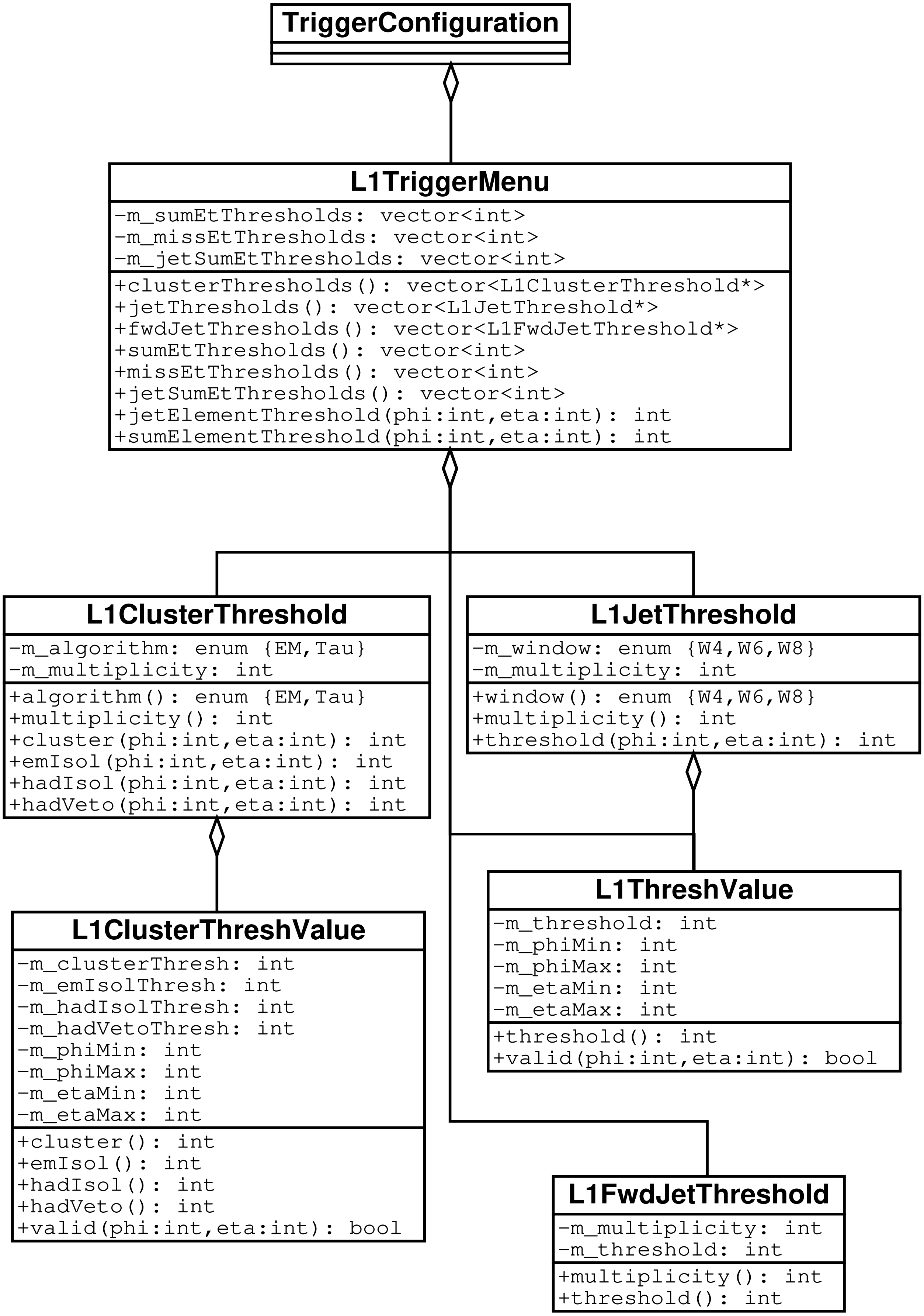
Granularity

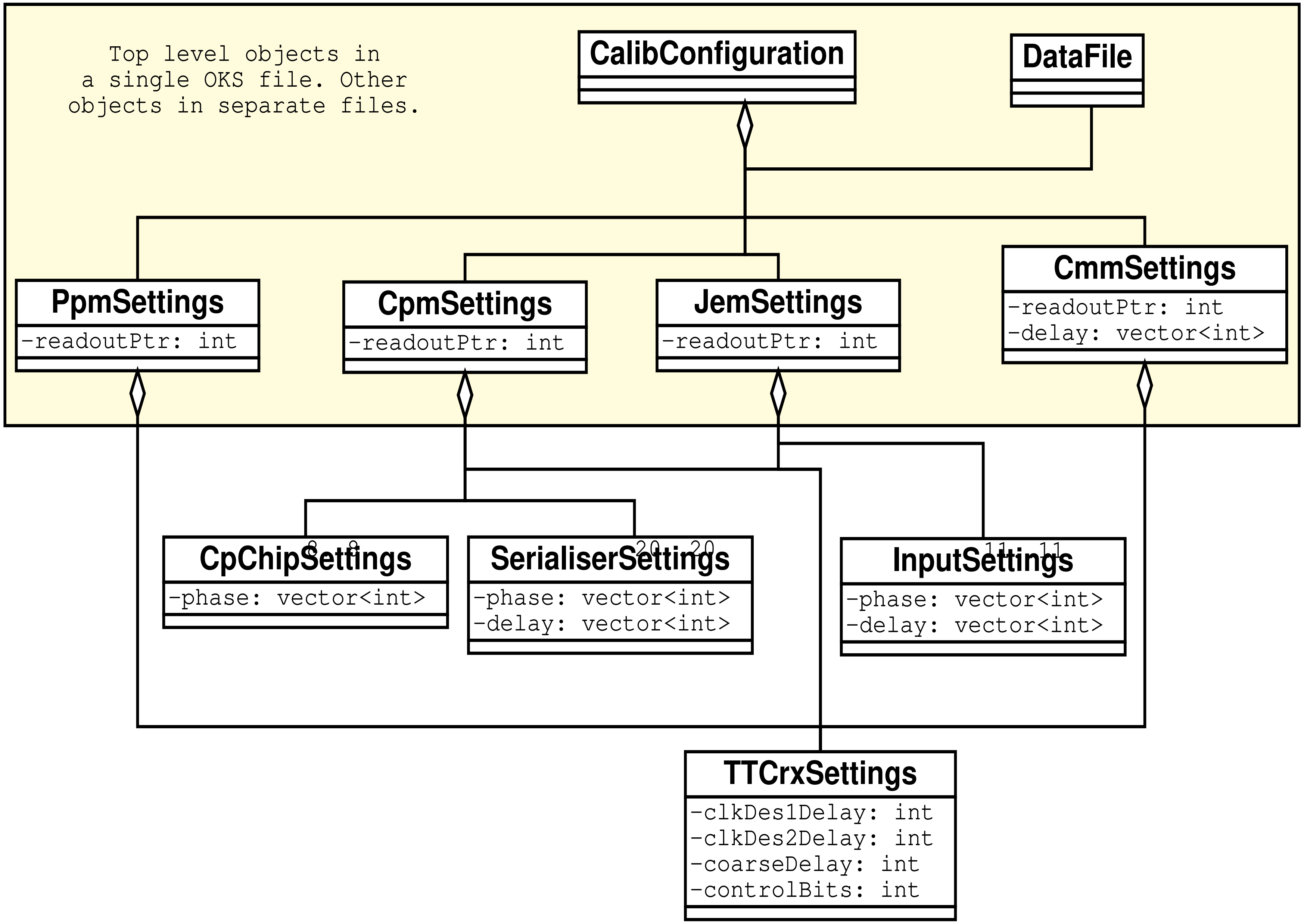
- Do we need access or detailed control of individual subcomponents of modules? (Eg change BCmux mode on a subset of channels). Or are global modes enough?

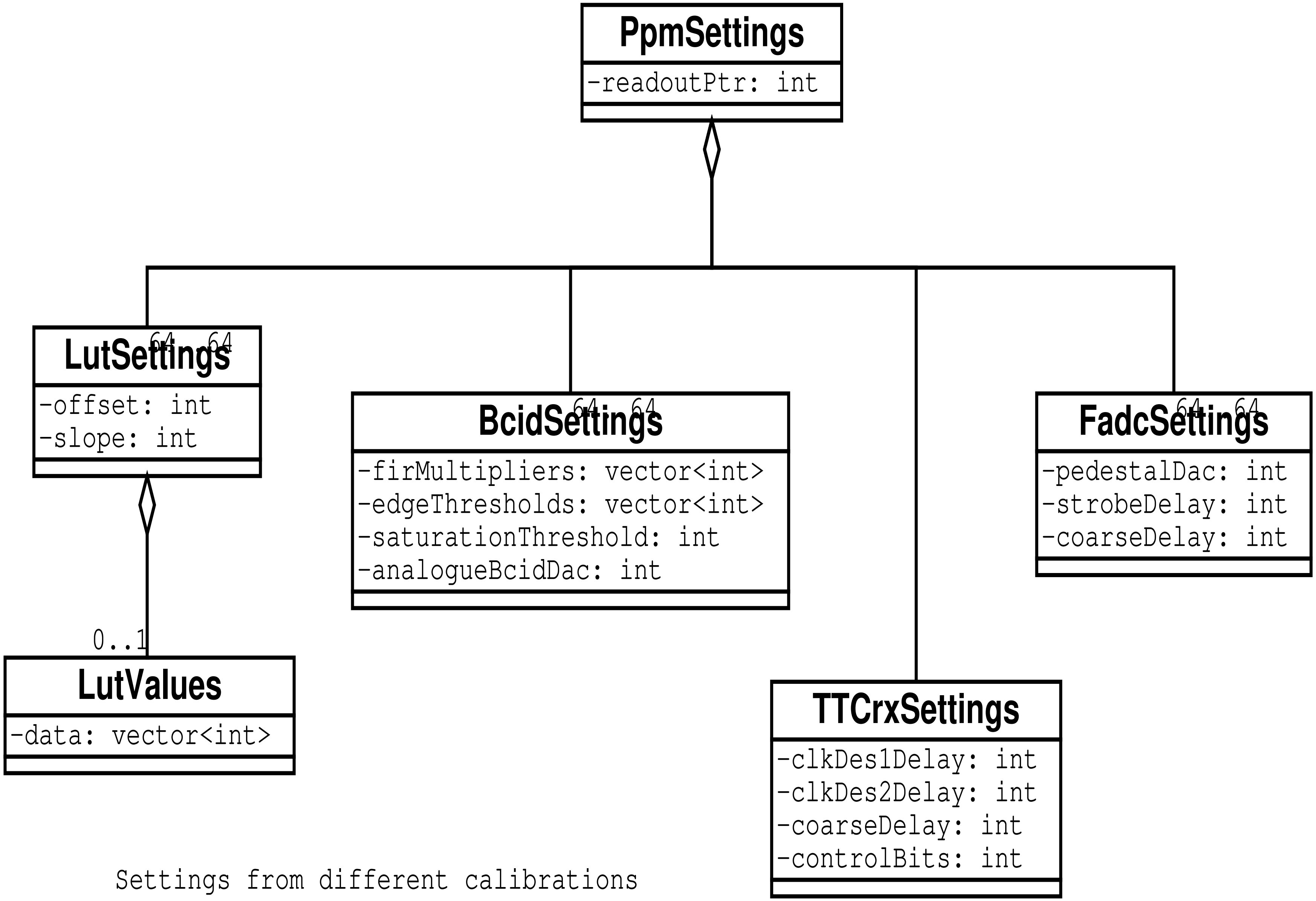
Database Objects

Quick Summary

- Draft document (SW note 005)
  `http://www.hep.ph.qmw.ac.uk/l1calo/doc/pdf/ConfigDatabase.pdf`
  has suggestions for trigger menu and for calibration data. The diagrams are included here for convenience.
- Version of trigger menu schema is already in use offline but may be changed.
- Calibration data certainly needs more work.
- Nothing yet on extending the existing hardware and software database schema (from the Online group) to describe the configuration of our modules.
- Also nothing on run parameters.
- General idea for both trigger menu and calibration data is to create a tree of objects. The top level object can be asked to find any lower level object or collection of objects, eg the calibration data for a single CPM.


TriggerConfiguration
L1TriggerMenu
-m_sumEtThresholds: vector<int>
-m_missEtThresholds: vector<int>
-m_jetSumEtThresholds: vector<int>
+clusterThresholds(): vector<L1ClusterThreshold*>
+jetThresholds(): vector<L1JetThreshold*>
+fwdJetThresholds(): vector<L1FwdJetThreshold*>
+sumEtThresholds(): vector<int>
+missEtThresholds(): vector<int>
+jetSumEtThresholds(): vector<int>
+jetElementThreshold(phi:int,eta:int): int
+sumElementThreshold(phi:int,eta:int): int
L1ClusterThreshold
-m_algorithm: enum {EM,Tau}
-m_multiplicity: int
+algorithm(): enum {EM,Tau}
+multiplicity(): int
+cluster(phi:int,eta:int): int
+emIsol(phi:int,eta:int): int
+hadIsol(phi:int,eta:int): int
+hadVeto(phi:int,eta:int): int
L1JetThreshold
-m_window: enum {W4,W6,W8}
-m_multiplicity: int
+window(): enum {W4,W6,W8}
+multiplicity(): int
+threshold(phi:int,eta:int): int
L1ThreshValue
-m_threshold: int
-m_phiMin: int
-m_phiMax: int
-m_etaMin: int
-m_etaMax: int
+threshold(): int
+valid(phi:int,eta:int): bool
L1ClusterThreshValue
-m_clusterThresh: int
-m_emIsolThresh: int
-m_hadIsolThresh: int
-m_hadVetoThresh: int
-m_phiMin: int
-m_phiMax: int
-m_etaMin: int
-m_etaMax: int
+cluster(): int
+emIsol(): int
+hadIsol(): int
+hadVeto(): int
+valid(phi:int,eta:int): bool
L1FwdJetThreshold
-m_multiplicity: int
-m_threshold: int
+multiplicity(): int
+threshold(): int


Top level objects in
a single OKS file. Other
objects in separate files.
CalibConfiguration
DataFile
PpmSettings
-readoutPtr: int
CpmSettings
-readoutPtr: int
JemSettings
-readoutPtr: int
CmmSettings
-readoutPtr: int
-delay: vector<int>
CpChipSettings
-phase: vector<int>
8 8
SerialiserSettings
-phase: vector<int>
-delay: vector<int>
20 20
InputSettings
-phase: vector<int>
-delay: vector<int>
11 11
TTCrxSettings
-clkDes1Delay: int
-clkDes2Delay: int
-coarseDelay: int
-controlBits: int



Settings from different calibrations
stored in separate OKS files