Table of Contents

Upgrade Software Reviews

Introduction

As the proposed upgrade detectors are developed, we hope to encourage good coding practice from as early a point as possible.

Once a upgrade detector has a sufficient body of working code, it should be submitted for review before inclusion into a main project release.

Once the detectors which form the LHCb Upgrade are nearing a final state, it is envisioned that there will be a series of further reviews with the intent to bring everything together as harmoniously as possible

Aim

The main aims of these reviews are:

- To ensure that problems or potential problems are caught as early as possible.
- To ensure that we all are consistent in our coding and interfaces.
- To ensure that code is fit to be integrated into a release.
- To aid in information transfer between the detector groups (both current and upgrade).

Benefits

Once a group has a sufficiently functional body of code, we believe a review is beneficial to everyone:

- It allows the coordinators to have a better handle on what you are doing and the problems you face and it makes it clear if we need to change or provide new functionality to the common part of the code.
- It allows the other groups to learn from the issues raised in the review.
- It gives the authors the chance to discuss the various possible implementations available to them.

Scheduled Reviews

None.

Global Feedback

Refactoring/Rewriting of Existing Code

Where an upgrade detector shares many similarities with an existing detector, there is a strong temptation to simply rewrite (closer to refactoring in fact) the existing body of code. This can lead to bloat/legacy/unnecessary code.

For new developers, who may not be intimately familiar with the workings of the existing detector code, by all means use the existing code to prototype your package/class. However, it is strongly recommended that all unnecessary code is removed before submission. In some cases it may even be profitable to perform a re-write of the new detector code.

Wrapper Methods

- Avoid class methods which simply wrap up some other trivial method (sensorXBegin() == sensorX.begin())

Lean Code

- Do not add methods until they're actually used - i.e. do not add placeholders.
- Think about your code footprint - do you really need a vector of doubles, would floats do? Is a histogram necessary, would a single number in the output be better?
- Granulate monitoring, do not book expert monitoring histograms by default - add some kind of switch.

Existing Methods and Classes

- Do not be afraid to diverge from existing methods, in time we will converge once we know what the needs of each detector are.
- Your package should be reasonably self-contained.

QM Tests

- If you are adding QM test code, it should live in the same package so that future developers can immediately find it.

Verbose and Debug Statements

Protect verbose and debug statements like so:

```
if ( msgLevel(MSG::DEBUG) ) debug() << "The debug message" << endmsg;
```

You can test that you don't have any unprotected debug/verbose by adding the option

```
MessageSvc().countInactive=True
```

which will print a table of all components where there are such unprotected debug/verbose statements, but only when you run with the

```
$CMTDEB
```

build.

Review Framework

- One detector group, either for a current detector or an upgrade detector will be designated as the principal reviewer.
- The relevant code will be placed in a temporary svn location at least a week before the review date.
- The code authors will prepare and give a presentation covering the aim, functionality and problems they experienced with the code at the same time as submitting the code itself.
- Similarly, the principal reviewers will prepare and give a presentation critiquing the submitted code. (Note that this covers both positive AND negative aspects.)
- All detector groups and other interested parties are encouraged to attend - we will benefit from the collective experience.

Authors

The authors should supply the following on the relevant tWiki page at least one week before the date of the review:

- A list of all the packages submitted to SVN and their location.
- A recipe for building the code and running a simple test job with debug output.
- A description of the purpose of the code (to aid the reviewer), ideally this will be reasonably detailed.
- Contact details of those responsible.

Reviewers

The reviewers should supply the following on the relevant tWiki page:

- A list of general comments resulting from your review.

Code Guidelines

- Write within the conventions outlined here.
- Try not to be "clever", aim to be clear instead.
- Comment whenever you have to remind yourself what something does.
- Remember that others will have to edit your code at some point.
- Try to make the code as "lean" as possible, i.e. for a new detector cloned from the existing one, leave out legacy code and backwards compatibility present in the original.

Corrections to the C++ Conventions

- We do not officially support Windows based builds, so loop parameters can be defined at loop instantiation.
- We have moved to SVN, so the CVS macro on the first line of each file is unnecessary.
- Ignore convention R60. It is incorrect in that there are situations where the `?` opeator is more efficient. For instance inline methods where the use of `?` allows for efficient coding with a single return statement. Multiple return statements in inline methods are practically a garantee that the compile will **not** inline the method in question.

Review Guidelines

- Focus on constructive criticism, i.e. if something is plainly wrong, give the reason and the suggested correction.
- Compile a detailed list of corrections such that none will be forgotten when the time comes to apply the suggested changes.
- Make note of any broader questions which should be addressed and air them at the review itself.
- Focus on maintainability - at some point the code will be edited/maintained by someone else.
- Do not only focus on the code itself - comments in code are also important, they must be clear, correct and useful.
- Design issues:
    - Usage of C++ concepts
    - Efficiency for usage in code
    - Readability

Completed Reviews

- 20120612 FT and VeloLite (Agenda)
- 20120821 RICH, TT and VeloPix (Agenda)

Links

- C++ Conventions
- Python Conventions
- LHCb Coding Conventions (possibly out of date)
- Gaudi Guide (contains some python conventions, also old)
- Raw Event data banks
- Simulation Tutorials

-- PaulSzczypka - 24-May-2012

This topic: LHCb > UpgradeSoftwareReviews
Topic revision: r20 - 2012-09-19 - PaulSzczypka