Nektar++ peer-review

Procedure

0. Author runs branch through buildbot and fixes errors
1. Author submits a merge request, can suggest appropriate reviewers
2. Editors will assign two reviewers (or return if obviously correction): typically one experienced, one less-experienced
3. Reviewers review code as described above – comments put directly on gitlab (in general comments section or associated with specific lines of code).
4. At end of review, write comment "@author I have completed the review, please look at my suggestions" to alert author.
5. Author addresses comments (responding to the original comments individually), re-run through buildbot and add a comment to the relevant editor that changes have been made.
6. Editor merges branch.

Checklist

Has it run through buildbot and passed?

High-level Functionality:

- Has additional functionality been added which is duplicated elsewhere in the code?
- Has the functionality been placed in the most appropriate location?
    - Solver-specific functionality should be with the solver, not the libraries.
    - Conversely, generic functionality should live in the appropriate library
    - Better to develop new functionality (e.g. filters, forcing) in a specific solver and then generalise later.
- Have data members been added in the correct place (e.g. do data members really need to be in StdExpansion)
- Is the new functionality sufficiently complex to warrant a higher-level design discussion?

Low-level functionality:

- Are there any functions which have been added but are never called?
- Is the use of temporary arrays justified?
- Have appropriate data structures and algorithms been chosen?

Documentation:

- Is there doxygen documentation for new functions?
- Are there appropriate additions to the user guide for new functionality?

Coding standard:

- Does the code meet the standard? (We will try to automate a lot of this)