LFC Guidelines

Daniel Secrieru
(dsecrieru@hotmail.com)

Following are a set of working guidelines for LFC developers. This is still a draft paper a may suffer modifications. However, all the team members should read these guidelines and try to follow them as close as possible. Thank you for your understanding! :-)

1 Guidelines

- this is a friendly team; we all work voluntarily and because we love to code

- nobody will be forced to do anything that he/she doesn't want. Everybody will work on whatever he/she decides, but will finish that task and will not rely on someone else to do his job, unless he/she has good reasons for that and agrees with another team member to get help

- all discussions concerning LFC development will be held on the LFC-development list

- if you aren't already, get familiar with using tools like cvs, ssh, doxygen, latex, not to mention the compilers (gcc, bcc)

- time, time, time... time is everything! Each task should have a clear timeframe planification, and everybody should strive to hit the deadlines (without sacrificing the quality of comments, documentation, tests, ...). The dependencies between tasks should be very clear. Team members are expected to come with some realistic (not optimistic) estimations of the time necessary to finish a task and should take this in account when asking to work on a task to match his/her available time. Tasks that take longer than 3-4 days should be broken into subtasks that can be finished in no more that 3-4 days for the purpose of scheduling

- the development process of a LFC module should take the following steps:
  - every developer should think about the overal design of the task he/she is working on and post a proposal on the list, that is, sort of a briefing
  - big decisions like design are usually taken by the team and when the developer gets a green light from the team concerning his/hers proposal, he/she should start working on the interface for the class/module he/she is working on. When done, the developer should post a first RFC on the list, so that the team may comment the design. More

RFCs could follow if the developer must make modifications to the design, based on the team's comments. Consult /doc/papers/rfcSample.tex to see how an RFC is supposed to look like. This is probably the most important step in the development of a task and keep in mind that the RFCs are the tool with which one acomplishes this step

  - if everything is alright, the next step is implementation. When done, the developer should post a code review on the list, containing an archive with the code
  - only when the code review(s) have passed one team member approval (or more, if the task/modifications are more complex), the developer can commit the code into the repository. The code reviewing can be efficiently achieved by following a file diff (use a diff utilitary)

- here are a few important things that should accompany the development process of a LFC module:
  - every developer should maintain the doxygen documentation for its module and update it if necessary
  - every developer will write some tests for its module, which will be located in /tests
  - every developer will write some tutorials and a detailed latex doc on how to use his/hers module, if not trivial
  - when working on some task, every developer should maintain a doc with whatever resources he/she used in the developing process (links, articles, white papers, books, ...)
  - every checkin should have a brief description about the contents, modifications, testes etc.

- all developers should follow the LFC coding style, which is to be found in /doc/papers/codingStyle.tex

- a few cross-platform development related issues:
  - LFC is cross-platform, so any cross-platform task like sockets/networking or GUI support will be developed cross-platform. That means that every developer should have acces to both Unix and Windows systems, or at least those working on cross-platform tasks. However, if that's not the case, a cross-platform task can be split between developers on different platforms
  - the design of the PALs should take into account all the targeted platforms

  - every and all features of a cross-platform module should be throughly tested on all supported platforms

- all the comments/suggestions of team members to list postings/proposals/RFCs/code reviews should be concise and easy to understand. Criticism is allowed, only if it's constructive . Unnecessary 'hostile' behaviour should be avoided. Opinions should be kept objective and not rely on subjective tastes. If there's no other way, please warn that a personal opinion is following and should be viewed as such