Tcon 23 Jun 2017
Present: Casper da Luis-Costa, Ben Thomas, David Atkinson, Elise Emond, Kris Thielemans

The tcon was a discussion on how to make sure that we can go "back in history" reliably. We currently tag SIRF, SIRF-SuperBuild and CCPPETMR_VM, but there is no reference to explcit versions/git tags of the dependencies. We currently just checkout CCPPETMR forks of STIR, gadgetron etc.
See https://github.com/CCPPETMR/SIRF-SuperBuild/issues/24

A general preference was felt for option 3 presented by Casper (specify git hashes in `SIRF/version_config.cmake`), however this conflicts with the current SuperBuild mechanism which builds STIR etc before it even clones SIRF. Option 2 (`SuperBuild/version_config.cmake`) is therefore our recommended solution. This lead to a wider discussion on the role of the superbuild etc.

A problem with the SuperBuild is that it doesn't really generate a nice environment for developers. For instance, Visual Studio Solutions (and presumably XCode etc) generated by the superbuild provide "external projects" that are "bare" and not really useful for developers. In any case, there is also a danger of losing source modifications with the superbuild. This might be solvable for SIRF only if we make SIRF the main project (as opposed to an external project). However,this then still wouldn't work for all the other projects, so is not a viable solution.

Conclusions from the discussion follow now.

The SuperBuild is targeted as a general build mechanism for software useful for CCP-PETMR, including SIRF, gadgetron, various converters etc. We therefore make the SuperBuild the main mechanism to track versions of dependencies and to build a "coherent" set of tools where all these versions work together.
Implications:

- It does need to be renamed from SIRF-SuperBuild, and that SIRF is just one of the "external projects"
- SIRF-developers can use the Superbuild to build all dependencies, but would still want to clone their own SIRF repo and run CMake for SIRF only.

Version tracking:

- SIRF will use normal CMake mechanisms to ensure minimum versions of dependencies but not specify git hashes of dependencies.
- We chose option 2 in the github issue #24 (as option 3 is not possible via the proposed superbuild), i.e. the CCP-PETMR-Superbuild has a file with git tags for all projects. The developer can override the tags by specifying variables, and we also provide a "developer flag" that just uses "master" for everything.

VM:

- The VM update mechanism needs to switch to use the Superbuild
- We make the Vagrantfile "complete" (or at least as much as possible). Also, we specify a particular version of the superbuild in the Vagrantfile. This way one can rebuild VM with specific OS, compiler and all dependencies.
- We could have our own vagrant-box as basis as opposed to a standard Ubuntu box. We could even push different boxes for each release. However, this is probably not necessary if

Commented [Gu1]: If so, I recommend that someone investigate the use of packer: https://www.packer.io/

we solve the previous point. We will evaluate later if this is necessary. (Ben to check if current Ubuntu box is “frozen”)

Release file:

- We need to have a release file with steps to take to make a new release (SIRF, tagging, updating superbuild, update Vagrantfile, etc etc)

Notes added by Ben after the tcon:
By default Vagrant will pull the latest version of a box and will also check for updates:

> config.vm.box_check_update - If true, Vagrant will check for updates to the configured box on every vagrant up. If an update is found, Vagrant will tell the user. By default this is true.

We can, and probably should, turn this off. The boxcutter/ubuntu1604 that we currently used hasn’t changed for 6 months, but there’s nothing stopping the owner updating it or even revoking it. You can specify a box version number or min/max version number in the vagrantfile.

Options:

- Fix the version number for the existing box in the Vagrantfile.
- Derive our own box, push it back and fix the version number to use that.