Proposals for Fixing Our Issues with CMT

May 8, 2007 Status

Due to a general lack of support from the CMT development community (we have sent various posts to their mailing list with no response), the difficulty to upgrade to CMT v1r18, and the sense that our use of user defined CMTPATHs is a thing of the past for CMT - we are planning to find other alternatives. Right now there are two:

SCons

"My preferred approach would be to write a python package that ingests the existing requirements files directly and generates the correct SConstruct files. This would be similar in spirit to Toby's pycmt package, but would not necessarily rely on cmt to get the interpackage dependencies. The benefit of this approach would be that this new python package could be made to interpret the requirements files in a more sensible way (i.e., private really does mean private) and we could extend the syntax to provide for specifying libraries that are only to be used in certain contexts (e.g., by the test programs) and not automatically exported." From Jim Chiang

versus

NPMS (Navid Package Management System)

Keep CMT and patch as necessary

Assuming we have too much invested in CMT - we decide to keep it for both package management and building. However, we must attempt to address as many of our problems with CMT as possible. We have already demonstrated an ability to circumvent the problems CMT causes by various workarounds:

- MRStudio checks out packages using cvs, not the cmt checkout .. but we rely on CMT to get the used packages of the given checkout package, in case of a recursive checkout, and we use CMT to get the list of tags of a given package
- Generation of solutions files for VS2003 is done internally to MRStudio, but also in this case CMT is used to know the dependencies between packages

Currently we're attempting to work out our problems with:

- too many environment variables
- Need to fix dumpbin for shared library usage
- Lack of VS2005 support

We would be stuck with

- CMT's mechanism to determine dependencies.
- We would be left to fix Intel Mac Makefile creation - though perhaps that would spark some interest from CMT-central

Another problem, is that there does not seem to be a coordinated effort to fix up our issues with CMT - so we patch when time allows, and other tools tend to avoid using CMT directly, resulting in various strategies for checking out and building our code.

PyCMT

https://confluence.slac.stanford.edu/display/SAS/Python+scripts+to+create+VS2005+files

Will we just address Windows issues with Python scripts or move on to Unix as well? And if so..why not SCons instead?

Who will be keeper of the python scripts? And do we have enough Python experience and interest to proceed along this path?

SCons Plus CMT

Continue to use CMT for package management (check outs and determining dependencies) but use SCons via CMT's document generator to create our makefiles.

Potential Issues:

- We will still need to provide our own means to generate Visual Studio files since SCons does not provide these?
- Can Windows use CMT's document generator at all?
- CMT will continue to determine package dependencies.

SCons without CMT

Use SCons as our build system and completely remove CMT from the picture by creating our own system that checks out the code consistent with our use of packages.

This could be seen as a goal, if we start with "SCons Plus CMT".

Go Our Own Way

Create our own package management system from scratch that handles both the checking out of code from CVS, determination of dependencies, and creating makefiles for all our supported systems.

We would need a working prototype and a full schedule to determine the level of effort this would require.