CMT Action Requirements checklist

Functional Requirements for Package Management/Build System

See tables below for a detailed list of required and desired properties and functions for any replacement of CMT. Most come from the **Requirements/Goals** section of the main CMT Action Committee Confluence page. The tables may be used as checklists for the two proposed replacements, identified as **SCons** and **NPMS**. **SCons** is meant as shorthand for **SCons Enhanced**; that is, SCons plus additions written locally. Where appropriate the tables allow for checking off a requirement by implementing a standalone solution (Stnd), independent of (and callable by) both SCons and NPMS. Functions which should be callable by one or more clients (RM, MRStudio, individual developer) are indicated

like this . Functions or parts of functions which are desirable but not immediately required are indicated [like this] in the Description section.

CVS, Package Management

Name	Description	SCons	NPMS	Stnd
Package checkout	Check out requested tag (may have value HEAD) of a single package from CVS, installing in user's workspace in a manner consistent with current directory/subdirectory conventions.		Working	
Limited recursive checkout	Given a *container package* (one that refers to a list of other packages by name and CVS tag), check out all packages it refers to.		Working	
Dependency tree	Given a package, determine all packages it depends upon.		Working	
Find tags	Given a package, return list of its CVS tags [of a particular syntactic form]		Planned	

Requirements on Requirements File Replacement

Name	Description	SCons	NPMS
Inter-package dependence	Ability to express dependence on other packages, optionally with version (=CVS tag) specification		Not needed
Macro/include	A way to express common forms in a single place which may be referred to by many packages. Should have a way to supply arguments.		Planned
Per-target dependence	[Ability to specify dependence of individual target on other targets, both within the package under consideration or external to it.]		Working
Per-operation dependence	[Ability to specify that a particular dependence is in effect at compile time, at link time or at load time. Less interesting, maybe entirely redundant, if we already have per-target dependence specification.]		Not needed

Configure, build

Name	Description	SCons	NPMS
Configure	If a configure step ("establish environment, get ready to build") is needed at all, it should be callable and there should be a recursive variant. Whether configure options are set by an explicit operation or are inferred (e.g., from environment variables) it must be possible to indicate OS, compiler, debug, etc.		Not needed
Single package build, recursive build	Support request to build specified package and all it depends upon. [Would be better still if one could alternately specify 'build only current package; if external inputs are absent, stop immediately with error']		Working
Libraries	Build one or both of static library and shareable, as specified in requirements-replacement file. Further distinguish between Gaudi component and other shareables. For non-component shareable on Windows, make all public members (function members and data members) available at link time to other packages.		Planned

Export/Install, Run

Name	Description	SCons	NPMS
Install	The output of the build system should be in a form suitable for packaging for export and remote installation.		In Progress
Run	Run an application with supplied arguments. Any part of the build environment needed at run time should be transparently available then, without special action by the client.		Planned /Not needed

Developer Support

Name	Description	SCons	NPMS

Path management	It should be easy for developers to • establish a local writable working directory • develop against an existing read-only release in such a way that local packages take precedence over those in the release at compile, link and run time.	Planned
Configuration switching	It should be easy for a developer (or for a GUI application, on behalf of the developer) to switch from one environment to another. Here *environment* might include items such as debug/release setting and path as in previous item. [It should be possible to establish and use distinct environments simultaneously from different processes.]	Working
Debugging	All tools necessary for debugging an application built by the system should be available. For Windows this means project /solution files so the application can be run in Visual Studio (and patched from there?). For unix systems it should be possible to run applications from emacs/gdb or DDD (probably doesn't require anything special of build system beyond reasonably portable and self-contained run-time environment for applications).	Planned

J. Bogart