Using CMT with Daya Bay NuWa Software

Brett Viren

Physics Department



Daya Bay NuWa Offline Software Tutorial

Overview

1. CMT Basics
   - Packages
   - `requirements` File
   - Types of CMT Packages
   - CMTPATH
   - Projects
     - `CMTPROJECTPATH` and `CMTPATH`
     - User Projects
2. Anatomy of a CMT Package
   - Layout conventions
3. Exercises

Configuration Management Tool

CMT is a very flexible system that organizes:

- Software source code
  - packages and projects
  - versions (currently, not strongly used in Daya Bay)
  - consistent files and directories
- The build process
  - dependencies, internal and external
  - multiple platform support
  - partial software build
  - override "official" code with locally customized code
- Runtime environment
  - required variables like `PATH`
  - automatic and arbitrary software specific ones
  - idempotent setup scripts

CMT Packages

The basic unit of CMT work is a Package. A package is defined as:

- A directory, from which the package name is derived.
- A `cmt/` subdirectory containing (at least) a file called **requirements**
  - tells CMT what to do with the package.
- A set of files that CMT can operate on (eg. source code)

Trivial `requirements` file:

```
package PackageName
version v1r2p3
use OtherPackage v1r0
use AnotherPackage v* SomePath/ToFollow
# A comment
# ... Further CMT commands ...
```

CMT Packages and You

CMT always operates on one package at a time. You almost always need to be in `SomePackage/cmt/` when you run CMT related commands. The common things you will do:

cmt config will create the `setup.[c]sh` scripts and do other house-keeping on the package

cmt br cmt config as above, but also will "broadcast" this command to all dependency packages in this "project" (covered later)

source setup.sh will modify your environment to match instructions in the `requirements` file of the Package and in any dependencies.

make build just this package

cmt br make broadcast the make command to build this package and any dependencies packages in this project

Basic requirements File Elements

tags have a name, but no value. Their existence is used to imply something else. Usually set by CMT implicitly or via CMTEXTRATAGS.
Eg: Linux, gcc4 and dayabay, debug
macros assign some value to a name. The value can be some default or be based on the existence of a tag. Eg:

```
macro shlibsuffix    "so" \
      Darwin         "dylib"\
      WIN32          "dll"
macro_append <package>_linkopts " -lm "
macro libname lib<package>.$(shlibsuffix)
```

requirements File Advanced

path style env. var. can be modified like:

```
path_remove PATH "/<package>/bin/" WIN32 "\<package>\bin\"
path_prepend PATH "/<package>/bin/" WIN32 "\<package>\bin\"
```

pattern is used to expand multiple CMT commands.

```
# Defined elsewhere:
pattern exec_path \
 path_remove PATH "/<package>/bin/" WIN32 "\<package>\bin\";\
 path_prepend PATH "/<package>/bin/" WIN32 "\<package>\bin\"

# In some package's cmt/requirements:
apply_pattern exec_path
```

document is used to add support to new build types by supplying Makefile **fragments**. Too much for here, just know it exists.

Using "use"

use tells CMT this package needs to use some element of another package. This will include all **public** parts of that package's `requirements` file and ignore anything in a **private** section. This also sets up a dependency hierarchy that CMT will walk when doing a `cmt br` command.
Syntax:

```
use SomePackageName [version] [path]
```

Reasons to use **use**:

- Your package links to a library provided by another package.
- Running something in your package depends on environment variables produced by another.
- Your package requires a macro or pattern defined in another.

Full example

From tutorial/Detector/DetWalker:

```
package DetWalker
version v0
# What this package depends on:
use GaudiAlg
use DetDesc      v*       Det
use DetDescSvc   v*       Det
# Build basic library, files relative to src/ directory:
library DetWalker *.cc
# Gaudi hooks
apply_pattern component_library library=DetWalker
# The rest is only here, not by others
private
use Reflex v* LCG_Interfaces
use XmlDetDesc v* Detector
```

Types of CMT Packages

CMT supports a few types of packages, by convention. Some are:

Software holds source code, builds a library, may export public headers. Most packages are of this sort.

Glue used to make external packages look like they are manged by CMT. lcgcmt/LCG_Interfaces have many.

Policy defines specific configuration or CMT commands, support multiple platforms.

Configuration defines how to configure other packages, such as which version to use.

Release just a `cmt/requirements` that depends on several packages allowing them to be built and used together.

CMTPATH: How CMT finds packages

`CMTPATH`

- Environment variable CMT uses to find packages
- Lists one **or more** directories containing packages
- Set by hand for small CMT systems
- In Daya Bay **do not** set it by hand (see next).

CMT Projects

CMT collects related packages into a CMT project.
A project is defined as:

- A directory, from which the project name is derived
- A `cmt/` directory containing a **project.cmt** file
- A collection of packages in subdirectories.

Example `gaudi/cmt/project.cmt`:

```
project gaudi  # project name
use lcgcmt     # depend on another project
# Configure high level behavior
build_strategy with_installarea
setup_strategy root
# Package used to build entire project
container GaudiRelease
```

CMTPROJECTPATH and CMTPATH

CMTPROJECTPATH:

- Points to one or more directories containing projects
- Used by CMT to set CMTPATH when you do `source setup.sh` in some `cmt/` directory.
  - value is set based on the package from which you sourced setup.sh
- Just using the base release:

  ```
  CMTPROJECTPATH=/path/to/NuWa-RELEASE
  ```

- Use your own packages or override those in NuWa:

  ```
  CMTPROJECTPATH=$HOME/my-projects:/path/to/NuWa-RELEASE
  ```

- Order matters! CMT walks this path and will take the first instance of a given package it finds.
- Set by hand, or by local helper script

Where to put what file

All relative to PackageName/ directory

- `cmt/` CMT `requirements` file
- `src/` all C++ source code and **private headers**
  - `src/lib/` Gaudi-independent library code (optional)
  - `src/components/` code to "glue" lib into Gaudi (optional)
- `PackageName/` for **public headers only**. If you write an algorithm, MyAlg.h does **not** belong here but in `src/`. In our "legacy" code this is `include/`.
- `python/` any Python module files
- `share/` main Python scripts, options, misc.
- `doc/` change logs and any documentation files

Prerequisites

It is assumed you have already set up your account to use the tutorial project.

Make a project

Pick a place to put your projects (may be just your $HOME)

```
shell>  cd /path/to/my/area
```

Make a project directory

```
shell>  mkdir -p first_project/cmt
```

Create the `project.cmt` file (you can also use editor)

```
shell>  cat << EOF > first_project/cmt/project.cmt
>  project first_project
>  use dybgaudi
>  EOF
```

Add the top directory to the **front** of your project path. Note the use of backquotes in **‘pwd‘**.

```
shell>  CMTPROJECTPATH=‘pwd‘:$CMTPROJECTPATH
```

Congratulations, you now have an empty project ready to use.

Add a package

Make a package directory

```
shell>  mkdir -p first_project/MyFirstPackage/cmt
```

```
shell>  cd first_project/MyFirstPackage/cmt
```

Start a basic `requirements` file

```
shell>  cat << EOF>requirements
```

```
>  package MyFirstPackage
```

```
>  version v0
```

```
>  EOF
```

Need to do this also:

```
shell>  echo v0 > version.cmt
```

Add a package, continued.

Watch CMT modify your CMTPATH:

```
shell>  echo $CMTPATH | tr ':'  '\n'
/ihepbatch/dyb/dybsw/NuWa/NuWa-trunk/dybgaudi
/home/dyb/dybsw/NuWa/NuWa-trunk/lcgcmt
/home/dyb/dybsw/NuWa/NuWa-trunk/gaudi
/home/dyb/dybsw/NuWa/NuWa-trunk/lhcb
```

```
shell>  cmt config
Creating setup scripts.
Creating cleanup scripts.
```

```
shell>  source setup.sh
```

```
shell>  echo $CMTPATH | tr ':'  '\n'
/home/bes/guest09/first_project
/home/dyb/dybsw/NuWa/NuWa-trunk/dybgaudi
/home/dyb/dybsw/NuWa/NuWa-trunk/lcgcmt
/home/dyb/dybsw/NuWa/NuWa-trunk/gaudi
/home/dyb/dybsw/NuWa/NuWa-trunk/lhcb
```

Interrogate CMT

See what packages you are using

```
shell>  cmt show uses
#
# Selection :
use CMT v1r20p20070720 (/home/dyb/dybsw/NuWa/external)
```

Not many, lets add one:

```
shell>  echo use GaudiAlg >> requirements
```

```
shell>  cmt show uses
# use GaudiAlg * (no_version_directory)
#   use GaudiKernel v* (no_version_directory)
#     use GaudiPolicy v* (no_version_directory)
...
```

Broadcasts stay inside a project

When you type `cmt br <command>` it will send the command to all packages in the current project.

```
shell> cmt br pwd
# Now trying [pwd] in .../first_project/MyFirstPackage/cmt (16/16)
/home/bes/guest09/first_project/MyFirstPackage/cmt
```

```
shell> cd $GAUDIALGROOT/cmt
```

```
shell> source setup.sh
```

```
shell> cmt br pwd
# Now trying [pwd] in .../NuWa-trunk/gaudi/GaudiPolicy/cmt (11/15)
/home/dyb/dybsw/NuWa/NuWa-trunk/gaudi/GaudiPolicy/cmt
...
# Now trying [pwd] in .../NuWa-trunk/gaudi/GaudiAlg/cmt (15/15)
/home/dyb/dybsw/NuWa/NuWa-trunk/gaudi/GaudiAlg/cmt
```

```
shell> cd -
```

Note: output edited for readability.