Table of Contents

Working with CMT install areas

Starting with Gaudi v19r1 (released on 24th January 2007), the LHCb software is released with CMT Install Areas enabled. This page describes how to work in this environment (as opposed to the previous, now obsolete, environment without Install Areas).

Working with CMT Install Areas on lxplus

Choose which program and version you want to work with, and type the following command:

`setenv<project> <version>` (for example `setenvBrunel v31r0`)

This command is an alias for `SetupProject --build-env <project> <version>` (for example `SetupProject --build-env Brunel v31r0`). If in your environment the alias is not defined (e.g. on Windows, or for the specific project you are working with), you can use the `SetupProject` command directly

The command does the following actions:

- If it does not already exist, it creates a directory `$User_release_area/<project>_<version>` (for example `~/cmtuser/Brunel_v31r0`, assuming `$User_release_area` is set to its default value `~/cmtuser`) and populates it with a file `cmt/project.cmt`
- Sets the environment variable `CMTPROJECTPATH`. This variable replaces `CMTPATH`, which is no longer needed
- Puts you into the `$User_release_area/<project>_<version>` directory.
- Stay in this directory (i.e. not the parent directory `~/cmtuser`) when you `getpack` your packages

From now on, you can e.g. use `getpack`, `cmt make`, etc. - If you want to work with a different program or a different version, just repeat the instructions above to set yourself into a different `~/cmtuser/<project>_<version>` (for example `~/cmtuser/Boole_v13r0`). Note that, within a given project directory you will only see the packages contained in that directory in addition to those in the corresponding release area.

Note that `CMTPATH` should never be explicitly set by the users. If you experience problems with the environment try to `unsetenv CMTPATH`

Understanding the environment

Each project contains a file `cmt/project.cmt` which contains some lines as follows:

```
project Brunel_v31r0
use BRUNEL BRUNEL_v31r0
```

The first line is the `project` version (in this case `Brunel_v31r0`), which must be the same as the directory in which the file is located. This is followed by one or more `use <project>_<version>` statement which give the project(s) which the current project depends on. In this example the current project (`cmtuser/Brunel_v31r0`) depends on the `BRUNEL/BRUNEL_v31r0` project. If you look at the `cmt/project.cmt` file in the `BRUNEL/BRUNEL_v31r0`, you will see that it depends on `LBCOM_v6r0` and `REC_v4r0` and so on.

Each project contains an `InstallArea` directory which contains all public include files and all libraries built in the project. When you build a package with a given project, make will search for includes and link libraries first in the `InstallArea` of the current project, then in those of the projects it depends on - in the current example the search order is `cmtuser/Brunel_v31r0` followed by `BRUNEL/BRUNEL_v31r0` followed by `LBCOM_v6r0` and `REC_v4r0` and so on. Similarly when searching for plugin libraries at execution time. Note

that only libraries and include files in the `InstallArea` directory are searched for, not those in the individual packages of the project; when you build a package, the public includes and the libraries are copied to the `InstallArea`.

The projects are found by searching on the `$CMTPROJECTPATH`, whose default value is `$User_release_area:$LHCBPROJECTPATH`, with `LHCBPROJECTPATH` translating to `$LHCb_release_area:$Gaudi_release_area:$LCG_release_area`.

- If you wish to search also the `$LHCBDEV` area, you can use the `--dev` switch of `SetupProject` (e.g. `SetupProject --dev Brunel`). This modifies `CMTPROJECTPATH` putting `$LHCBDEV` as the **first** directory in the search path, ahead of `$User_release_area`. In many cases you might want to search the `$User_release_area` **ahead** of `$LHCBDEV`; in this case you can modify `CMTPROJECTPATH` explicitly, as follows: `setenv CMTPROJECTPATH ${User_release_area}:${LHCBDEV}:${LHCBPROJECTPATH}`
- If you wish to search any other area, you can use the `--dev-dir` switch of `SetupProject` (e.g. `SetupProject --dev-dir /afs/cern.ch/lhcb/software/nightlies/lhcb2/Mon Brunel`)
- You can see the hierarchy of the projects you are using (and from where) by using the command `cmt show projects`

Note also the names that are given by default to projects in the `$User_release_area` are simply following a convention, to make it easy to remember which released project they depend on. But you could use any name you like, for example `MyBrunelPatches`.

Migrating to a new release

From time to time, you will want to migrate an entire project in your `$User_release_area` to a new official release of the LHCb software.

When is a migration not advised?

Blindly copying packages can lead to inconsistent dependencies, incomplete or broken InstallAreas, libraries and python, and failing to pick up required changes. It is very much better not to blindly copy packages, but to start from scratch in the new project.

- The majority of the time, a user doesn't need to getpack anything. So there is no need to migrate anything.
- Sometimes users need to getpack something for a temporary patch which appears in the next release anyway. Again, no need to migrate.
- In the case a user or developer getpacks something to make their own changes, it is better to getpack the latest version of the package into the new project, because there may have been incompatible changes elsewhere which you need to pick up.
- In case you've made a lot of changes/updates/fixes/additions to one package, because you are a developer, or have created your own package, it may be OK to move the package around, but it's better to commit the changes for everyone else to use aswell, and then getpack the head into the new project, for simplicity.
- In the case the new version is sufficiently far along from the previous version, they may be incompatible, it is simpler not to migrate, but to start again.

Migration instructions

To do this it is sufficient to modify the dependency in your `cmt/project.cmt` to the new version, then rebuild. For example, if you want to move from DaVinci v26r2 to DaVinci v26r3 you just need to:

```
mv DaVinci_v26r2 DaVinci_v26r3
```

then edit `DaVinci_v26r3/cmt/project.cmt` updating the dependency to

```
project DaVinci_v26r3

use DAVINCI DAVINCI_v26r3
```

Unfortuna

Sharing developments between projects

From time to time (typically if you are developing some core software component) you may want to share the new development between several different application environments. For example you might want to test a change to `Det/STDet` in both the Boole and Brunel application environments. Since Boole and Brunel require two distinct `$User_release_area` projects, you might think that you need to put your `Det/STDet` modifications in both projects, but this is not necessary. You can instead:

- Create a `LHCb_v22r0` project, in which you put the modified `Det/STDet` package
- Modify the file `Brunel_v31r0/cmt/project.cmt` by adding the line `use LHCb_v22r0` **before** any other `use`
- Similarly, modify the file `Boole_v31r0/cmt/project.cmt` by adding the line `use LHCb_v22r0` **before** any other `use`

Now both application projects depend on your `LHCb_v22r0` patches, which will be picked up ahead of the equivalent packages in the `$LHCb_release_area`

Should you want to make all packages in your `$User_release_area` regardless of the project, you can use the following command: `cmt br -global -select=cmtuser gmake` This makes all packages in `CMT show uses` that contain `cmtuser` in the package path.

Experiences with the new CMT setup (C.Jones 07 Feb 2007)

After working with the new setup for a while, I thought I would share a couple of problems I encountered with the new system. In both cases, the root cause of the problems was the fact that in the new setup, the (public) includes that are used to compile a package are not those in the package itself, but from the install area.

Scenario One

You are editing a tool. The interface is in one package (say `RichKernel`) and the implementation in another (say `RichTools`). In the old CMT system if you changed the interface, since `RichKernel` does not use the interface itself, it just gives it a convenient home, I never needed to make `RichKernel`. So I didn't. `RichTools` would directly use the interface header file from my copy of the `RichKernel` package and all was OK.

With the new scheme this is no longer the case. The interface that is used by `RichTools` is that in my private install area, NOT my private package. Thus, you DO need to type make in `RichKernel` whenever you change the interface, in order to get it copied to the install area, ready to be used by `RichTools`.

Bottom line : It is probably safest to always type `make` in every package you alter a file in (To be very safe, you could always run `cmt br -global -select=cmtuser make` which will build all dependencies for a given package)

Scenario Two

This one is I think quite nasty. When it happened to me it took a while to spot. What happened is best described by the following series of events :-

- I had made many changes to a RICH package in my private area.
- At the same time preparations where being made for the migration of our software from Gaudi v18 to v19. Consequently I held off commiting these changes to CVS until the migration was complete.
- Once Brunel v31r0 was released, I then started merging my private changes with those in CVS.
- For one package, CVS created lots of conflicts. I decided the best way to deal with these, as I have many times before, was to move my modified version to some other location, `getpack` the current CVS head, and then work through adding back my changes.
- I forgot to update one header file (`XXX.h`). This header file was a public one and thus copied to the install area. The version in my area would not compile.
- Now, what I had managed to achieve with the above steps was for the version of `XXX.h` in the install area to have a newer time-stamp than that in my package area (I believe CVS checkouts preserve the time stamps of files from when they where checked in ?)
- So now, when you type `make`, it decides that the file in the package area is NOT newer that that in the install area and does not copy it, even though you have changed it (via `getpack`) since the last time you built the package.
- So, even though my package had an include in it that should not work, the package compiled fine since it was using the version in the install area.
- I thought all was OK, committed my changes to CVS. Marco checked them out and found the package would not compile, since `XXX.h` was the wrong version.
- I got confused since it worked for me. It took me quite some time to figure out the file in the install area was not being updated.

So, bottom line. Be careful to not end up with a situation where the install area is newer than your area package area.

Scenario Three (P. Koppenburg, 2 May 2007)

Be careful in which order you clean things. With the old setup it didn't matter. If you want to remove a package **don't do**

```
rm -r Hat/Pack
cd $DAVINCIROOT/cmt ; cmt br make clean
```

This will not remove the package library from the `InstallArea`. Only the package knows how to clean its libraries. So do:

```
cd $DAVINCIROOT/cmt ; cmt br make clean
cd ../../../../ ; rm -r Hat/Pack
```

Here the first step will have cleaned the `InstallArea`.

MarcoCattaneo - 24 Jan 2007

This topic: LHCb > InstallAreaWiki
Topic revision: r22 - 2010-11-16 - RobLambert