Table of Contents

Using Eclipse to work with the LHCb software

Eclipse is a multi-platform development environment. Its extreme flexibility makes it possible to use it in a lot of environments, from the Java development to the Web development.

It is possible to use Eclipse also in LHCb, even if with still some rough edges that will be smoothed out if there are enough requests.

Prerequisites

You need to be familiar with few LHCb developments concepts and tools:

- C++
- SetupProject
- getpack
- cmt

You also need the standard LHCb environment when you start Eclipse. So, if you are not running on `lxplus`, you need to call `LbLogin` before starting eclipse (at the current state of the local installation, you may need to specify the full path to the eclipse executable).

Basic introductions to Eclipse

This software tutorial concentrates on how to use Eclipse within the LHCb software environment. But first you should know what eclipse actually is, how to use it in general, and what different things it can be used for!

If you're already a user of eclipse, you can skip this part, if not, see the EclipseBasicTutorial.

Forewords

Before proceeding, you need Eclipse.

You can chose between a custom installation and the shared one

	Pros	Cons
Custom	you can install the plugins you like most and keep them up-to-date	you have to install it yourself and you will be probably limited to a single host
Shared	pre-defined set of plugins updated more or less regularly	can be run from lxplus, lxbuild or a desktop machine

This tutorial will make use of the shared installation I set up for LHCb for simplicity. Once you are familiar with Eclipse you can install your own copy (see EclipseConfiguration). I will highlight whenever something is specific to the LHCb installation with the symbol .

We will also assume we are starting from an empty cmtuser directory.

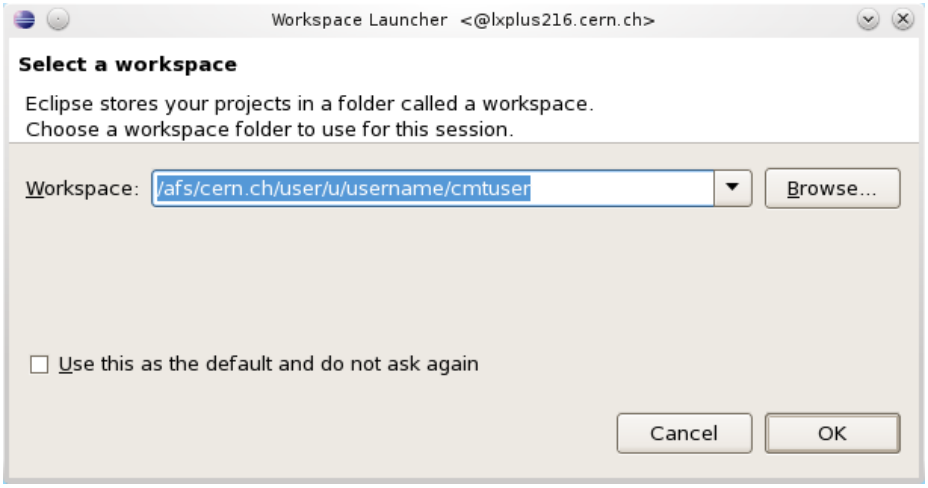
First time

screencast

To start eclipse (requires LbScript >= v5r4), just log on lxplus5 an type

```
eclipse &
```

A dialog will ask you to select your workspace. It will propose you the standard `cmtuser` directory in your home directory



Workspace selection

If you use a different location for your `$User_release_area`, you have to change the workspace location to point to it.

The first time you create a workspace, Eclipse will show you a welcome page from which you can explore its features. This time we can close it with the button highlighted in the image. Close the `Photran 6.0` (FORTRAN support) welcome page too.

First time, welcome view

You should **uncheck** the entry `Build Automatically` from the menu `Project`. It is not needed in general, but we need to have it unchecked for the tutorial.

Project / Build Automatically

From the `Window` menu select `Preferences`. Type "Python" in the filter box at the top of the left pane and select `Interpreter - Python`. For this tutorial you can click on the `Auto Config` button and accept the default. In some cases it will be necessary to use the actual Python interpreter (the one in the LCG externals), but for now the system one is enough. Click the `OK` button to store the changes to the configuration.

Now you that we made Eclipse create the configuration files in the workspace, you can go back to the command line prompt.

Create a User Project (`setenv<Project>` + `getpack`)

screencast

If you have followed the LHCbSoftwareTutorials, you know about `setenv<Project>` (AKA `SetupProject --build-env`) and `getpack`. Here we will use them to prepare a local project that we will work on using Eclipse.

In this example, we will work with the package `Tutorial/EclipseTutorial` using the project `LHCb`.

Check that the directory pointed to by `$User_release_area` is the one you used as workspace. If it is not the case, change the value of `$User_release_area`.

From the command prompt, prepare the local project with

```
SetupProject --build-env LHCb v31r8
getpack Tutorial/EclipseTutorial head
```

Let's go back to the Eclipse window (restart it if necessary).

From the menu `File` select `Import`, then `General/Existing Projects into Workspace`. In the dialog that

appears, click on `Browse...` and select the directory of your project (`LHCb_v31r8`), then click the button `Finish`.

Import Existing Project

Optionally (mainly if you want to work with many packages), you can go to the properties of the `C/C++ Build` group itself and in the `Behaviour` tab select the "parallel build" and the "optimal number of jobs".

Parallel build configuration

Exit from the `Properties` dialog with the `OK` button.

Building

screencast

Now we are ready to build: right-click again on the project and select `Build Project` (you can see the compilation going on in the `Console` tab at the bottom).

The package `Tutorial/EclipseTutorial` doesn't compile (on purpose). You can see the error in the bottom pane in the tab `Problems`. Double-clicking on the problem will open the problematic file highlighting the line that caused the error. If you click on the two horizontal arrows at the top of the `Project Explorer` view (*Link with Editor*), the project explorer will show where the file you are currently editing is in the project hierarchy.

Compile error

Correct the line (`m_freq` must be replaced with `m_frequency`), save the modified file (`CTRL+s` or `File->Save` or right-click in the editor) and build again.

Running the Tests

screencast

To run the tests of a package you need to call `cmt TestPackage` in the `cmt` directory of the package itself. You can tell Eclipse to do it by creating a custom "make target".

Right-click on the `cmt` directory of the package and, from the menu, select `Make Targets` -> `Create...`. Fill the dialog choosing the name `TestPackage`, uncheck `Use builder settings`, set the name of the builder to `cmt` and unckeck `Run all project builders`.

Custom make target for the tests

Now select the `Make Target` tab in the right pane (may be hidden), look for the directory `Tutorial/EclipseTutorial/cmt`, which will contain the target `TestPackage`. Double-click to run it.

The summary of the tests is visible in the `Console` view in the bottom pane, but you can use the internal web browser to display the HTML test summary that you can find in the left pane: `test_results/.../index.html`. Right-click on the file and select `Open With` -> `Web Browser`.

Result of the tests
(the HTML summary is available thaks to the environment variable `GAUDI_QMTEST_HTML_OUTPUT`)

When you run the tests from within an Eclipse project, the GaudiTestingInfrastructure creates launch configurations that can be use to execute the tests bypassing the QMTest wrapping (useful for debugging). You can access the configured launchers from the `Run` menu.

Debugging

screencast

Eclipse has got a user-friendly graphical front-end to `gdb`.

First we need to compile in debug mode (using the `Debug` configuration we set up), so select `Build Configurations` -> `Set Active` -> `Debug` from the project contestual menu (right-click). Build and run the tests (to regenerate the launcher configurations).

Open the file `CounterExampleAlg.cpp` from the `src` directory of the package. Using the `Outline` view in the right pane, look for `CounterExampleAlg::initialize` and double-click in the gray bar on the left of the editor window at the level of the function definition to set a breakpoint, a small blue dot will appear.

Setting a breakpoint

Unfortunately, there is a problem in the way the current version of Gaudi generates the launchers. To fix it select `Debug Configurations...` from the `Run` menu, go to the `Debugger` tab and close the dialog with the `Close` button.

From the `Run` menu select `Debug History` -> `eclipsetutorial.simple_test`. A dialog box will ask you if you want to switch to the `Debug Perspective`, tell it that you want and check the box to make it remember the decision.

The debugger will stop in the Python "main" function, and it will tell you that it doesn't have debug symbols for it. It's OK, let it continue by clicking the `Resume (F8)` button in the `Debug` view (top-left), which is a green triangle.

After some time, the debugger will stop at the breakpoint we defined before.

Hitting a breakpoint

Now you can follow the what is happening in your code. BTW, did you notice the spell checking in the comments?

Other Topics

Running Eclipse on lxplus via VNC (for slow connections)

Follow the instructions at RemoteLxplusConsoleHowTo then start Eclipse on the terminal window you get in the virtual display.

Development of LHCb Plugins

The LHCb Eclipse plug-ins are available for developers to allow for contribution.

To set up the development environment to work on the plugins, you have to install the Plugin Development Environment (PDE) in your Eclipse (you can find more informations on the eclipse web site). **warning**: you cannot use the shared installation on AFS because it doesn't have the PDE plugins.

Start Eclipse. I suggest to start from an empty workspace, at least the first time.

From the **File** menu, select **Import...**, then choose **Team Project Set** as project source in the dialog. Click **Next** and use this URL: http://cern.ch/lhcbproject/GIT/LHCbEclipsePlugins.psf. When you click **Finish**, Eclipse will get the sources of all the plugins as projects in your workspace (note that they are GIT repositories).

I cannot explain how to develop Eclipse plugins, for that you have the Eclipse documentation.

More Topics (TO-DO list)

- Interaction with Subversion
- Advanced debugging
- Templates
- PyDev

-- MarcoClemencic - 10-Jan-2011

This topic: LHCb > EclipseTutorial
Topic revision: r14 - 2012-07-12 - MarcoClemencic