Table of Contents

Description of the L1DiMuon analysis code

Note: This page is still under development, so you may not find all the information which should end up here eventually.

Introduction

This page summarizes how to use the code developed to create look up tables for the MuCTPI hardware and simulation, and analyze the performance of these configurations.

The code was originally developed to run on single muon MC simulations, but has recently been upgraded to handle real ATLAS data. All of the necessary code is in the ATLAS user SVN repository. In the following the page gives a summary of how the code works internally, and then explains step by step how to use it.

The analysis method

Summarize how the MuCTPI works, how the look up tables are created, and how their performance is analyzed.

Getting the code compiled

The code is in the ATLAS user SVN repository. You can have a look at it in your browser here. The following explains how to retrieve, compile and use the code.

Initial checkout

If you start from scratch, you first need to create a working area for the packages. I use CMT to manage and compile the source code, as it made it possible to have one working area on AFS, and use the compiled applications both from my desktop and my laptop. CMT is the code management application used by the ATLAS offline code as well.

Create an empty directory somewhere. Take note that you'll need at least about 70 MB of space for the compiled code. (This is because by default the code is compiled with debugging symbols.) Go to this directory, and check out "working area code":

```
svn co svn+ssh://svn.cern.ch/reps/atlasusr/krasznaa/L1DiMuon/trunk ./
```

This will create the following layout in your working area directory:

```
cmt/
   project.cmt
Doxyfile
setup_CERN.sh
setup_STANDALONE.sh
checkout_LUTAnalysis.sh
```

The setup scripts take care of setting up your environment for compiling the code. Note that I developed the project on BASH, so there are no guarantees that it will work on anything else.

There are two separate setup scripts in order to make it simple to set up the project both on lxplus, and on any other supported system type. Compiling the code on lxplus and in standalone mode is discussed separately here.

Setting up the compilation on lxplus

Nowadays all regular lxplus machines run SLC5, but since I started writing this code quite a long time ago, the compilation system still fully supports SLC4. (I removed SLC3 support at one point.) Note that SLC4's default compiler is GCC 3.4, and you don't need to set up a different compiler to build the code.

Before sourcing `setup_CERN.sh`, make sure that if you're on SLC5, you have GCC 4.3 set up. SLC5 comes with GCC 4.1 by default. But most of the centrally provided libraries at CERN are only compiled with GCC 4.3 for SLC5. (For instance XercesC is not available with GCC 4.1 on AFS.) To set up GCC 4.3 on lxplus, do:

```
source /afs/cern.ch/sw/lcg/contrib/gcc/4.3/[platform]/setup.sh
```

Now source `setup_CERN.sh`. It will detect that you haven't checked out anything yet, and will tell you to at least check out the `L1DiMuonPolicy` package.

Setting up the compilation on a "standalone" system

In order to be able to compile the code on practically any kind of system, the compilation system supports a "standalone" mode. In this mode the user has to point the CMT glue packages to the location of ROOT and XercesC on the system using some environment variables. These variables are set in the `setup_STANDALONE.sh` file, and have the following meaning:

- `ROOT_ROOT`: The directory where you have your local version of ROOT installed.
- `PYTHON_ROOT`: In case ROOT was compiled against a different version of Python than the default on your system, you need to set this variable to point to this special version of Python. Usually it can be left to point to "/usr".
- `XERCES_ROOT`: The directory where you have your local version of XercesC installed.

The default values for these variables reflect the setup on my SLC5 desktop machine, which has ROOT installed under /usr/local/root by hand, and has XercesC installed from the SLC5 repository.

While it is possible to compile the code on practically any system as long as you set these variables correctly, unless you try to compile the code on one of these systems, internally CMT will name your system as "Unknown_system".

- SLC4 with GCC 3.4 on a 32-bit machine (`slc4_i686_gcc34`)
- SLC4 with GCC 3.4 on a 64-bit machine (`slc4_amd64_gcc34`)
- SLC5 with GCC 4.1 on a 32-bit machine (`slc5_i686_gcc41`)
- SLC5 with GCC 4.1 on a 64-bit machine (`slc5_amd64_gcc41`)
- SLC5 with GCC 4.3 on a 32-bit machine (`slc5_i686_gcc43`)
- SLC5 with GCC 4.3 on a 64-bit machine (`slc5_amd64_gcc43`)
- Intel OSX with GCC 4.0 (`osx_i386_gcc40`)
- Intel OSX with GCC 4.2 (`osx_i386_gcc42`)

The code will still work like that, but the directories created by CMT will look a bit frightening I guess.

Package checkout

The code is implemented using a number of separate CMT packages. Have a look at the SVN repository to see all the available packages. To make it easier for non-developers to check out the packages necessary to compile the `LUTAnalysis` package, there is a helper script `checkout_LUTAnalysis.sh`. You should just execute it in the directory where you checked out the "working area code". Unfortunately CMT no longer supports checkouts using a requirements file, as it would've been much more elegant to have CMT figure out which additional packages it needs to check out just based on `LUTAnalysis/cmt/requirements`. After running the checkout script, you'll end up with the following packages in your directory:

```
External/PROOF
External/ROOT
External/XercesC
L1DiMuonPolicy
LUTAnalysis
common/GoodRunsLists
common/analysers
common/cmdl
common/logging
common/ntuple
common/prooftools
common/tools
```

Compiling the code

The main package is called `LUTAnalysis`. To compile it together with all the packages that it relies on, go to `LUTAnalysis/cmt`, and execute:

```
cmt br make
```

This will make CMT visit all the packages used by `LUTAnalysis`, and build each one of them.

The purpose of the CMT packages

In this section I give a short explanation on what the different packages are there for.

- `L1DiMuonPolicy`: This is one of the tricky packages. It defines how the entire project should behave. To have all the packages behaving in a consistent way, they all have to use at least this package from the project. It defines the compilation rules, how the libraries and applications should be installed and where, and defines a document handler for installing any kind of file in the InstallArea (called `compiled`) of the project.
- `Extenal/ROOT`: Glue package to ROOT. More documentation later.
- `External/PROOF`: Extension of the `External/ROOT` package. It hosts the script that can create "virtual PAR packages" out of all the CMT packages of the project. More documentation later.
- `External/XercesC`: Glue package to XercesC. More documentation later
- `common/GoodRunsLists`: The ROOT-only code from the offline `DataQuality/GoodRunsLists` package, tweaked to fit into this project. Have a look at the physics workbook for some more hints on how to use this code.
- `common/analysers`: Simple classes for decoding the information stored in the various MuCTPI words. Could probably be merged with `common/tools`.
- `common/cmdl`: A command line options parser package. I just "stole" it from the TDAQ repository a long time ago. It provides a nice interface for creating C++ applications which receive complicated command line options.
- `common/logging`: The project uses a common way of printing messages on the console. This package holds the code necessary for doing all this sort of output generation.
- `common/ntuple`: Helper classes for reading files following the "MuctpiD3PD" data format.
- `common/prooftools`: Classes that come in handy when using PROOF(-Lite). These could come useful in any sort of analysis.
- `common/tools`: Some small helper classes, hardly worth creating a separate package for...
- `LUTAnalysis`: The main analysis package. It creates a number of libraries and applications, which are documented further down on this page.

Using the applications

In this section I summarize how to use the compiled applications of the project. In general however each application can print instructions about its command line options if you run it with the `-h` or `--help=` parameter. To set up your environment for running any of the applications, you always have to source the setup script of the package that holds the application. For the applications of `LUTAnalysis`, you would always have to execute the following in a fresh shell to set up your environment:

```
# Set up CMT somehow
# Set up the correct version of GCC somehow
source setup_STANDALONE.sh
source LUTAnalysis/cmt/setup.sh
```

l1dimu_createLUT

This is the main application of the project. It can analyze the "MuctpiD3PD" files and construct a Look Up Table XML file based on this analysis for the MuCTPI. It has a large number of command line parameters, which you can check by running:

```
# l1dimu_createLUT -h
Usage: l1dimu_createLUT [-v code] [-r filename] [-b value] [-e value]
                        [-c value] [-f value] -x filename -i filename
                        -g filename [-l filename] [-p] [-s]

Options/Arguments:
        -v code       Level of output verbosity
        -r filename   Name of the output ROOT file
        -b value      Value of the B-B cut between 0.0 and 1.0
        -e value      Value of the B-E cut between 0.0 and 1.0
        -c value      Value of the E-E cut between 0.0 and 1.0
        -f value      Value of the F-F cut between 0.0 and 1.0
        -x filename   Name of the output XML LUT file
        -i filename   Name of the XML file defining the input
        -g filename   Name of the XML file defining the sector layout
        -l filename   Name of the XML file defining the good runs list
        -p            Use PROOF-Lite for the job
        -s            Don't write description into the created XML

Description:
        This program can be used to extract look up table files
        for the MuCTPI (simulation) from "MuctpiD3PD" files. Enjoy!
                         Attila Krasznahorkay
```

-- AttilaKrasznahorkay - 14-Sep-2010

This topic: Main > L1DiMuonAnalysis
Topic revision: r3 - 2010-10-12 - AttilaKrasznahorkay