# Table of Contents

# WDHCAL Simulation, Analysis and Reconstruction Code

## Common SVN Repository

All the source code related to the WDHCAL analysis should be stored in the common SVN repository. You can check it out using

```
svn co svn+ssh://svn.cern.ch/reps/clicdet/trunk/DHCalAnalysis
```

It contains several sub-directories for the different software packages (see below).

## Conversion of Raw Data

The raw data is converted into LCIO using the standalone program `dhcLcio` from the online directory of the DHCalAnalysis SVN. This program reads the raw binary data and creates the corresponding CalorimeterHit and TrackerRawData collections. It also stores the run meta data from the log book with each run header.

Building dhcLcio requires an installation of TinyXml.

```
cd DHCalAnalysis/online
bin/buildUser dhcLcio
```

Running the conversion program requires the configuration files in `DHCalAnalysis/online/cfg/dhcal/` for the channel to position mapping and the run list file (`DHCalAnalysis/DHCalRunList.txt`) for the run meta data. The syntax is

```
dhcLcio -f <raw/data/path> -w <output/path> -L <run/list/file> (-l <minimumNumberOfLayersHitToCre
```

The parameters in parentheses are optional. The default minimum number of layers hit is 1 and the default starting layer for the tail catcher is 39.

A submission script for ILCDIRAC to convert all available runs on the GRID is available: `DHCalAnalysis/PyRootCode/diracSubmission/submitConversion.py`.

## Detector Simulation (Mokka)

## Digitization and Reconstruction (Marlin)

The event reconstruction is performed in Marlin. The processors for the WDHCAL analysis are in the common SVN repository under

```
svn co svn+ssh://svn.cern.ch/reps/clicdet/trunk/DHCalAnalysis/DHCalMarlin
```

First set up the environment, create a build directory, let CMake create the make file and run the compilation with make.

```
source env.sh
```

```
mkdir build
cd build
cmake -C ../calicebuild.cmake ..
make install
cd ..
```

In order to use those processors in Marlin the library (libCaliceDhcal.so) has to be part of the MARLIN_DLL environment variable. This is already set in the env.sh. You can now add the DHCAL processors to your Marlin steering file and run it with

```
Marlin <steering.xml>
```

Example steering files can be found in the steer directory of DHCalMarlin.

This package contains several processors for reconstruction and analysis (see TungstenDHCALMarlinProcessors) as well as utility classes (see TungstenDHCALMarlinUtil).

# Running on the lxbatch

This is an example script to run a Marlin job on the CERN lxbatch farm.

```
#!/bin/bash
#BSUB -J singleParticles[1-399]
#BSUB -q 1nh
#BSUB -o /afs/cern.ch/user/j/jfstrube/work/public/DHCAL/stdout.%J_%I
#BSUB -e /afs/cern.ch/user/j/jfstrube/work/public/DHCAL/stderr.%J_%I
#BSUB -R "type==SLC5_64"

###### This top section contains instructions for the batch submission
###### (which environment, which queue (1nh), which name (399 jobs of name singleParticles))
###### obviously you need to change the path for the stdout and stderr

###### Now follows the actual job
###### First setup your environment
source /afs/cern.ch/eng/clic/software/x86_64-slc5-gcc41/Calice/devel/envCalice.sh
source /afs/cern.ch/sw/lcg/contrib/gcc/4.7.2/x86_64-slc5-gcc47-opt/setup.sh
export MARLIN_DLL=/afs/cern.ch/user/j/jfstrube/work/public/DHCAL/DHCalAnalysis/DHCalMarlin/lib/li

###### Now create your sandbox
mkdir LSB_${LSB_JOBID}_${LSB_JOBINDEX}
cd LSB_${LSB_JOBID}_${LSB_JOBINDEX}

###### This reads a file containing a list of runs
contents=($(cat /afs/cern.ch/user/j/jfstrube/work/public/DHCAL/DHCalAnalysis/DHCalMarlin/steer/fi
###### The filename is one particular line in that file
###### LSB_JOBINDEX is the number of one particular job (between 1 and 399 in this case)
filename=${contents[${LSB_JOBINDEX} - 1]}

###### Copy the run from castor into the sandbox
xrdcp root://castorpublic.cern.ch//castor/cern.ch/grid/ilc/user/c/cgrefe/calice/dhcal/lcio/Conver
###### Copy the Marlin steering file into the sandbox
cp /afs/cern.ch/user/j/jfstrube/work/public/DHCAL/DHCalAnalysis/DHCalMarlin/steer/BoxFinding.xml
###### And run Marlin on the run that was just copied
Marlin --global.LCIOInputFiles="${filename}" BoxFinding.xml
```

If you save the contents to a file name `bsubScript.sh`, you can submit the job using the `bsub < bsubScript.sh`

# Run Marlin on the Grid using ILCDIRAC

Marlin is one of the supported ILCDIRAC applications and can be used to submit grid jobs for the W-DHCAL analysis (see DiracUsage).

In order to make the DHCalMarlin processors available in the job the library has to be shipped in the input sandbox. Create a tar ball lib.tgz which has to contain a directory `lib/marlin_dll/` with all libraries that need to be appended to the MARLIN_DLL environment variable and can contain an additional directory `lib/lddlib/` for any other dependency that just needs to be added to the LD_LIBRARY_PATH.

A full example of a submission script is available: `DHCalAnalysis/PyRootCode/diracSubmission/submitUserJob.py`. In the same directory is a script called `createLibTarball.sh` to automatically create the lib.tgz directly from your local build of DHCalMarlin.

The submission script makes use of the run list tool and allows to submit jobs for all runs taken or specific sub sets of the test beam data. The basic syntax to run the submission script is

```
python submitUserJob.py -i <input/data/lfn/path> -o <output/data/lfn/path> -t <jobTitle> -x <stee
```

In order to select which data to process you can add `-a` for all runs, `-b` for beam data runs or/and `-n` for noise runs. You can also select only runs of a certain beam momentum using `-m 10` .

# Python & ROOT (Analysis, Plotting)

The python code for the high level analysis is in the common SVN repository

```
svn co svn+ssh://svn.cern.ch/reps/clicdet/trunk/DHCalAnalysis/PyRootCode
```

It contains the `pyDhcal` package with generic utility classes, e.g. the **`Run`** and **`RunList`** classes to access the run meta data (see TunstenDHCALData), as well as user code. In addition there is a **`diracSubmission`** package that contains examples for various submission scripts for ILCDIRAC.

# Event Display

## Raw Data

The event display needs the package tinyxml, which does not exist on lxplus, so you have to find a machine where you can install that. First, set up your environment. Then, build the code.

```
cd online/lib
make
cd ..
bin/buildUser edisp
```

The event display needs a run file as input. Unfortunately, it does not work with our LCIO data format.

## LCIO Files

The preferred event display for LCIO files is DRUID which is part of ILCSOFT

Source the environment script:

```
source /afs/cern.ch/eng/clic/software/x86_64-slc5-gcc41/ILCSOFT/v01-16/init_ilcsoft.sh
```

Running the event display:

```
Druid <data.slcio> (<eventNumber>)

Druid <data.slcio> <geometry.root> (<eventNumber>)

Druid <geometry.root>
```

Note that the order of the arguments is important. The event number is optional.

Obtaining the geometry file:

- The starting point is a GDML file which can be created by the simulation programs Mokka and SLIC
- This has to be converted into a TGeo hierarchy using the following commands in ROOT:

```
TGeoManager::Import( geometry.gdml );
gGeoManager->GetTopVolume()->Draw( ogl );
TFile *f = new TFile( geometry.root , recreate );
gGeoManager->Write();
f->Close();
```

By default only collections of the type MCParticle and ReconstructedParticle are displayed. To enable drawing of all collections (including CalorimeterHits) press the right button in the second row of buttons in the "Options" tab. You can then toggle drawing of the different collections in the "Eve" tab under "Event".

---

This topic: CLIC > TungstenDHCALCode
Topic revision: r8 - 2014-02-12 - ChristianGrefe