# ldmx-sw Tutorial Course: Basics

Tom Eichlersmith

University of Minnesota

eichl008@umn.edu

May 27, 2021

# Outline


LDMX

# Pre-Requisites

# Pre-Requisites


- A **container runner** [1] installed
- git version control installed
- Patience.

[1] `docker` is common for personal computers and `singularity` is common for shared computing clusters.

# Introduction

# Big Idea


LDMX

Simulate/Reconstruct/Analyze **event-by-event**

- Particles/Hits/Objects are put onto **event bus** (so they are **passengers** )
- **Processors** look at one event bus at a time in a certain **sequence**
- Generally, processors take passengers off the bus to **analyze** and/or **produce** passengers to put onto the bus
- **Process** knows the sequence and feeds the event buses to the processors

## Event Independence

Each event is **indepedent** of all other events in the software. This isn't necessarily true in real-life ( **pileup** ), but we have other methods for handling this.

# Compiling ldmx-sw

# Compiling ldmx-sw


LDMX

## Demonstration

I'm going to run these commands while sharing my screen. They are here for your future reference.

```
git clone --recursive https://github.com/LDMX-Software/ldmx-sw.git
cd ldmx-sw
source scripts/ldmx-env.sh
mkdir build
cd build
ldmx cmake ..
ldmx make install
```

# Configuration Scripts

# Purpose


LDMX

We have to tell the software **a lot** of things...

- How many events?
- Where is the data to input (if any)?
- Where should we put the output data (histograms or events)?
- Which processors to run?
- ...and much more!

## Configuration Script

A `python3` script which is run by the software before grabbing the necessary parameters.

**Today** : A basic configuration script that runs a simulation of electrons entering the LDMX detector design and then puts those events through some of our reconstruction pipeline.

# Process

- The pass name
- Input and output files
- Histogram file
- Maximum number of events
- Logging frequency

```
from LDMX.Framework import ldmxcfg
# create my process object
p = ldmxcfg.Process('sim')
# how many events to process?
p.maxEvents = 10
# we want to see every event
p.termLogLevel = 0
p.logFrequency = 1
# we also only have an output file
p.outputFiles = [
        "myFirstSim_"
        + str(p.maxEvents)
        + "_events.root"
        ]
```

# Minimum Simulation

```
from LDMX.SimCore import simulator
import LDMX.Ecal.EcalGeometry # geometry required by sim
mySim = simulator.simulator('mySim')
# get the path to the installed detector description
mySim.setDetector('ldmx-det-v12')
# Get a pre-written generator
from LDMX.SimApplication import generators as gen
mySim.generators = [ gen.single_4gev_e_upstream_tagger() ]
# add your configured simulation to the sequence
p.sequence.append( mySim )
# can still edit parameters after adding to sequence
mySim.verbosity = 1
```

# Reconstruction


LDMX

```
# import chip/geometry conditions
import LDMX.Ecal.ecal_hardcoded_conditions
import LDMX.Hcal.HcalGeometry
import LDMX.Hcal.hcal_hardcoded_conditions
# import processor templates
import LDMX.Ecal.digi as ecal_digi
import LDMX.Hcal.digi as hcal_digi
# add them to the sequence
p.sequence.extend([
    ecal_digi.EcalDigiProducer(),
    ecal_digi.EcalRecProducer(),
    hcal_digi.HcalDigiProducer(),
    hcal_digi.HcalRecProducer()
    ])
# View configuration before actually running
p.pause()
```

# Running `fire`

# Let's Run It!


LDMX

## Demonstration

I'm going to run these commands while sharing my screen. They are here for your future reference.

```
1 ldmx fire basic.py
```

# Common Pitfalls in Config Files


- Misspelling parameter names
  - ldmx-sw gives each parameter a sensible default, so the process will run if a parameter is not given
  - Misspelling a parameter means that your parameter value is just silently ignored
- Attempting to access files outside of directories mounted to container
  - Will see "Not found/doesn't exist" errors even though you can see the file
  - Try mounting the directory using `ldmx mount`
- Obviously lots more...
  - Many quirks of ldmx-sw that we all need to learn
  - Please put questions in slack and/or reach out to me
  - If you can imagine a C++ solution, put your ideas in the GitHub issue tracker

# Helpful Tips


- Never type out file paths or file names. Input files can be fed on the command line and output files can be formatted based off of input names and parameters.
- `os.path.realpath` to get full path without symlinks to files
- `argparse` to configure your run by changing processors or passing input files or changing parameters
- `print` ldmx classes to see what parameters are given to it after your python nonsense *but before they have been taken for processing*
- Define python functions (maybe even write your own python module) containing settings/processors/sequences/processes that you use a lot

**Overall**

Remember that the configuration script is run as a python script *before* the parameters are taken for processing, so you can automate a lot of stuff.

# Python Analyses

# Python Analyses


LDMX

## Good Starting Point

- Easier to write and interact with ✓
- Quick start-up and easy to develop ✓

**Today** : I will use a helper python module and write an analysis live! (The file will be shared after-the-fact for your reference.)
The helper python module ▸ EventTree is uploaded and is also available in the `ldmx-analysis` repository.

# C++ Processors

# Why?!?


**Tom, you just showed us how cool Python analyses are. Why use C++?**

## Reasons

- Access to more tools that Python (e.g. Detector ID Decoding)
- Generally easier to scale to larger samples

## My General Workflow

1. Make sure configuration for generating data sample runs
2. Write short Python analysis to start looking at generated data
3. Switch to C++ when Python analysis is "getting out of hand" or I need a C++-only tool

# Starting a C++ Analysis


LDMX

## General Outline

- Clone the `ldmx-analysis` repo and make your own branch (e.g. *myusername-dev*)
- Copy an existing analyzer's header (in `include/Analysis`) and source (in `src/Analysis`) to your own files in the same directories.
- Clean out the other analyzer code and start writing your own!
- Start make histograms/ntuplizing/skimming!

## To Build (in README of `ldmx-analysis`)

```
1 mkdir build; cd build; #make a build directory
2 ldmx cmake ..
3 ldmx make install
```

# Analysis


LDMX

Lots of examples in `DQM` module and `ldmx-analysis` repo.

**Features to Look At:**

- `HistogramPool`: Manage histograms by name instead of managing histogram pointers yourself
  - Helpful for creating *lots* of histograms
  - `EcalPN` analyzer in `DQM` module
- `NtupleManager`: Create tree of *once per event* values in histogram file
  - Helpful for higher-level physics variables when you don't know which you want to compare yet
  - `ECalVetoAnalyzer` in `ldmx-analysis`
- `setStorageHint`: Skim events based on criteria in your processor
  - Helpful for picking out specific (maybe frustrating?) events to look at in more detail
  - `TriggerProcessor` in `Recon` module
- `Process::keep`: drop or keep event objects to save space in your output files
  - Can be done only in python configuration