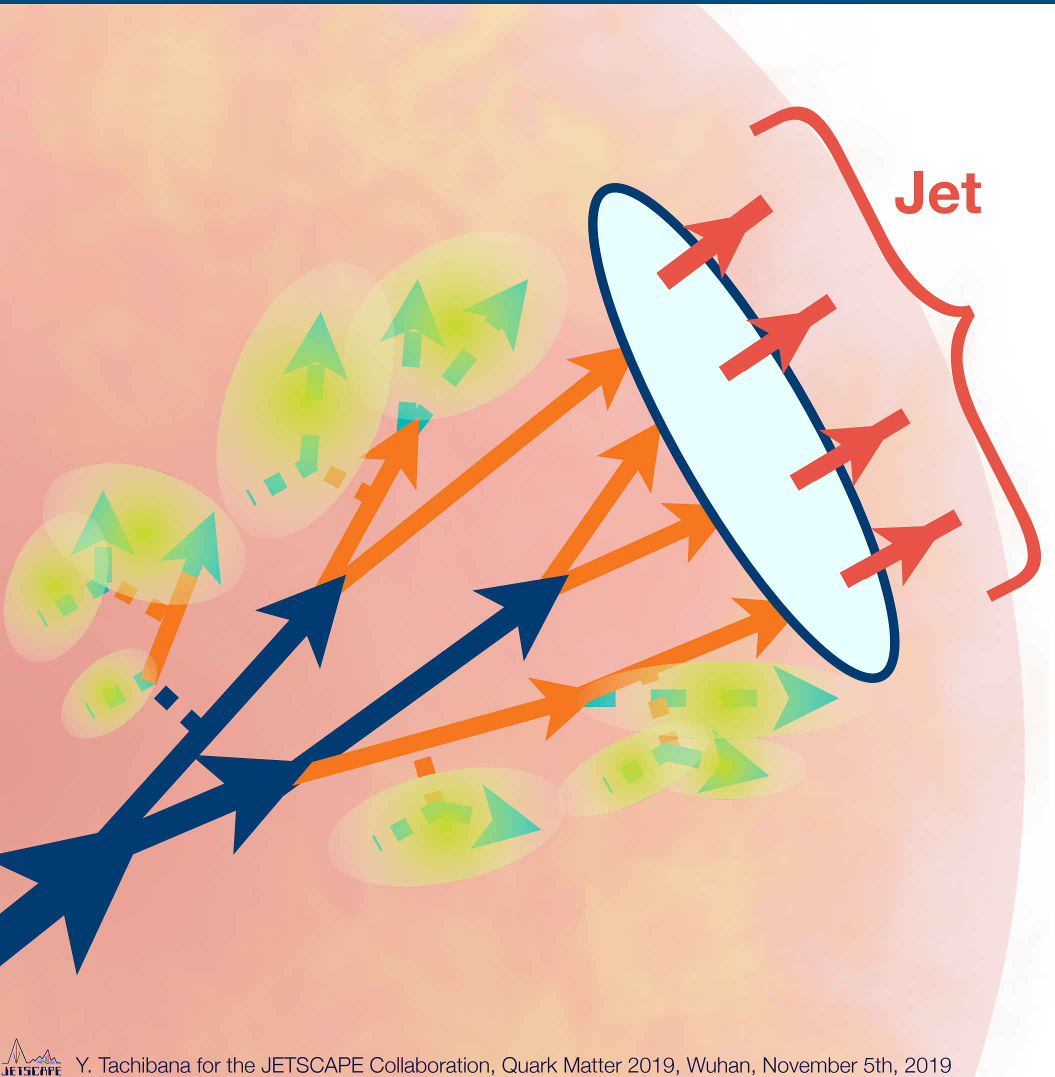
The JETSCAPE Framework



Hands-on session

James Mulligan
(UC Berkeley / LBNL)

JETSCAPE Online School
July 13 2020

Part 1
The basics of JETSCAPE

Part 2
Constructing an observable

Part 3
Implementing a custom module

Ask questions in slack channel: **#software-mulligan**

Part 1

The basics of JETSCAPE

Part 2

Constructing an observable

Part 3

Implementing a custom module

Review: Docker

https://github.com/JETSCAPE/SummerSchool2020

For this school, we require you to run JETSCAPE via docker

This allows everyone in the school to have a uniform software environment

Thanks to those who have helped others with the preparatory instructions!

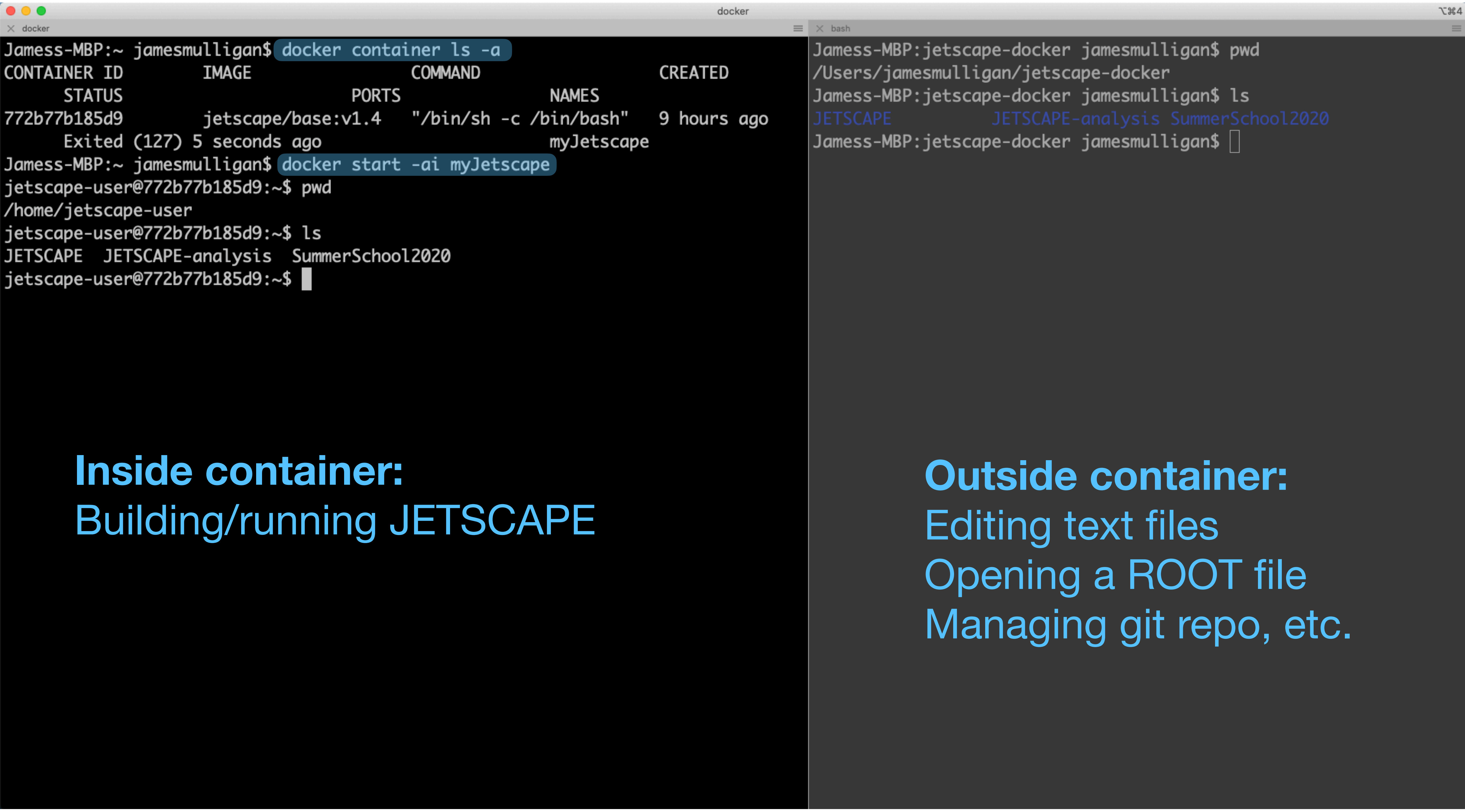
Tip:

Keep two terminals open: One inside the docker container, and one outside

When building/running JETSCAPE — **inside container**
When editing text files — **outside container**

Review: Docker

https://github.com/JETSCAPE/SummerSchool2020



```
Jamess-MBP:~ jamesmulligan$ docker container ls -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
      STATUS                        PORTS               NAMES
772b77b185d9        jetscape/base:v1.4   "/bin/sh -c /bin/bash"   9 hours ago
      Exited (127) 5 seconds ago                        myJetscape
Jamess-MBP:~ jamesmulligan$ docker start -ai myJetscape
jetscape-user@772b77b185d9:~$ pwd
/home/jetscape-user
jetscape-user@772b77b185d9:~$ ls
JETSCAPE  JETSCAPE-analysis  SummerSchool2020
jetscape-user@772b77b185d9:~$
```

```
Jamess-MBP:jetscape-docker jamesmulligan$ pwd
/Users/jamesmulligan/jetscape-docker
Jamess-MBP:jetscape-docker jamesmulligan$ ls
JETSCAPE          JETSCAPE-analysis SummerSchool2020
Jamess-MBP:jetscape-docker jamesmulligan$
```

Inside container:
Building/running JETSCAPE

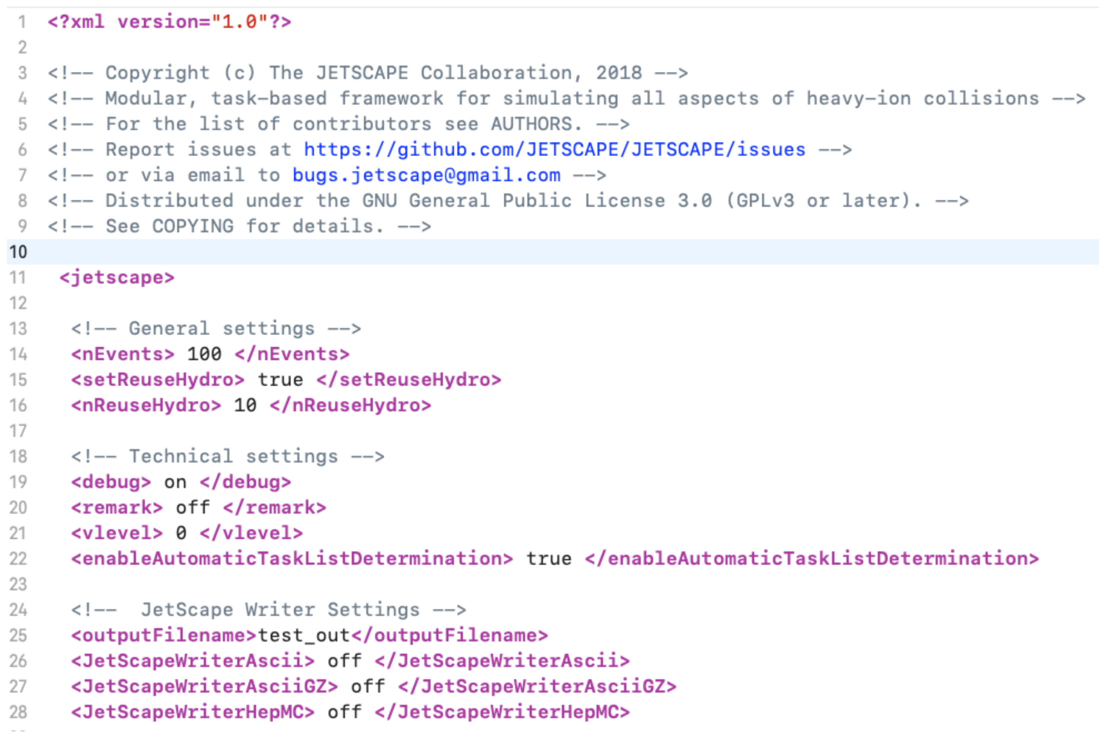
Outside container:
Editing text files
Opening a ROOT file
Managing git repo, etc.

The XML Configuration

All of the JETSCAPE settings are specified by two XML files:

- Master XML file: *you don't modify this*
    - Contains default values for every possible module and parameter
- User XML file: *you provide this*
    - Contains a list of which modules to run, and which default parameter values to override

Open the file and take a look: JETSCAPE/config/jetscape_master.xml



```
<?xml version="1.0"?>

<!-- Copyright (c) The JETSCAPE Collaboration, 2018 -->
<!-- Modular, task-based framework for simulating all aspects of heavy-ion collisions -->
<!-- For the list of contributors see AUTHORS. -->
<!-- Report issues at https://github.com/JETSCAPE/JETSCAPE/issues -->
<!-- or via email to bugs.jetscape@gmail.com -->
<!-- Distributed under the GNU General Public License 3.0 (GPLv3 or later). -->
<!-- See COPYING for details. -->

 <jetscape>

  <!-- General settings -->
  <nEvents> 100 </nEvents>
  <setReuseHydro> true </setReuseHydro>
  <nReuseHydro> 10 </nReuseHydro>

  <!-- Technical settings -->
  <debug> on </debug>
  <remark> off </remark>
  <vlevel> 0 </vlevel>
  <enableAutomaticTaskListDetermination> true </enableAutomaticTaskListDetermination>

  <!--  JetScape Writer Settings -->
  <outputFilename>test_out</outputFilename>
  <JetScapeWriterAscii> off </JetScapeWriterAscii>
  <JetScapeWriterAsciiGZ> off </JetScapeWriterAsciiGZ>
  <JetScapeWriterHepMC> off </JetScapeWriterHepMC>
```

...

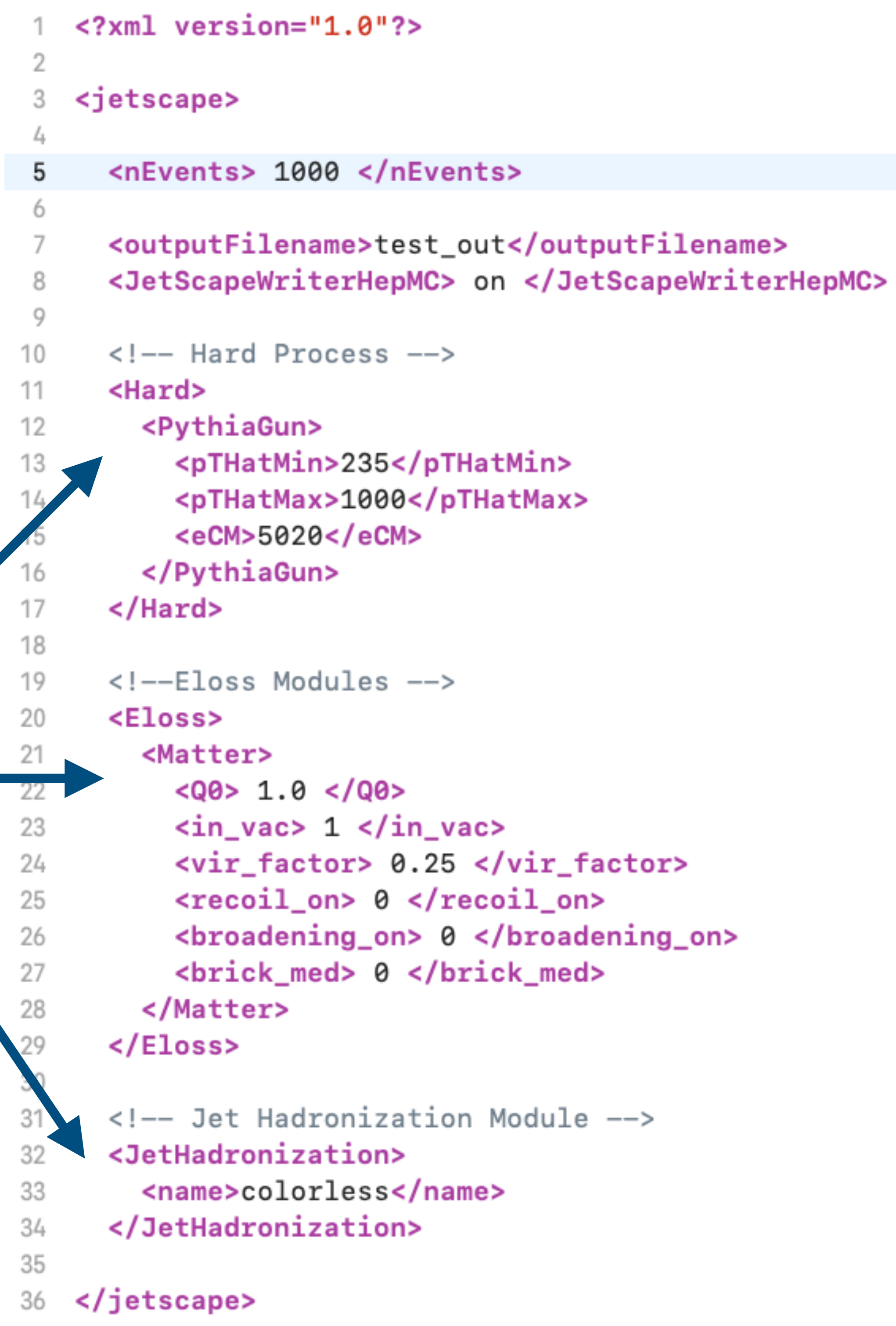
This is where you find all *possible* settings of all *possible* modules

As user, don't modify this!

User XML Configuration

Open the file:
JETSCAPE/config/jetscape_user_PP19.xml

Specifies which modules you want to run



```xml
<?xml version="1.0"?>

<jetscape>

  <nEvents> 1000 </nEvents>

  <outputFilename>test_out</outputFilename>
  <JetScapeWriterHepMC> on </JetScapeWriterHepMC>

  <!-- Hard Process -->
  <Hard>
    <PythiaGun>
      <pTHatMin>235</pTHatMin>
      <pTHatMax>1000</pTHatMax>
      <eCM>5020</eCM>
    </PythiaGun>
  </Hard>

  <!--Eloss Modules -->
  <Eloss>
    <Matter>
      <Q0> 1.0 </Q0>
      <in_vac> 1 </in_vac>
      <vir_factor> 0.25 </vir_factor>
      <recoil_on> 0 </recoil_on>
      <broadening_on> 0 </broadening_on>
      <brick_med> 0 </brick_med>
    </Matter>
  </Eloss>

  <!-- Jet Hadronization Module -->
  <JetHadronization>
    <name>colorless</name>
  </JetHadronization>

</jetscape>
```

User XML Configuration

Open the file:
JETSCAPE/config/jetscape_user_PP19.xml

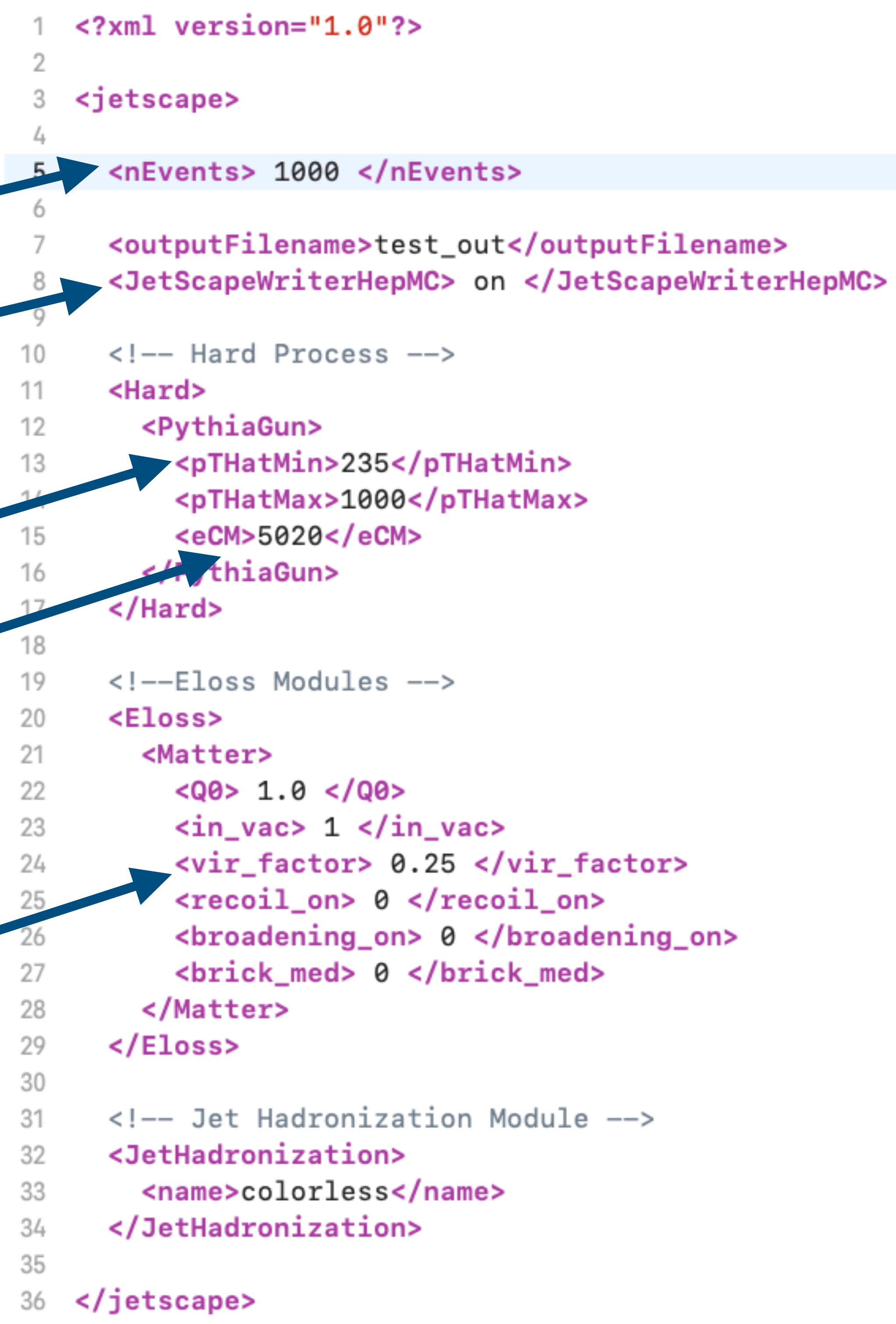
Set number of events

Set output format

Set $\hat{p}_{\mathrm{T}}$ bin edges

Set $\sqrt{s}$

Set any default parameter
that you want to override



```xml
<?xml version="1.0"?>

<jetscape>

  <nEvents> 1000 </nEvents>

  <outputFilename>test_out</outputFilename>
  <JetScapeWriterHepMC> on </JetScapeWriterHepMC>

  <!-- Hard Process -->
  <Hard>
    <PythiaGun>
      <pTHatMin>235</pTHatMin>
      <pTHatMax>1000</pTHatMax>
      <eCM>5020</eCM>
    </PythiaGun>
  </Hard>

  <!--Eloss Modules -->
  <Eloss>
    <Matter>
      <Q0> 1.0 </Q0>
      <in_vac> 1 </in_vac>
      <vir_factor> 0.25 </vir_factor>
      <recoil_on> 0 </recoil_on>
      <broadening_on> 0 </broadening_on>
      <brick_med> 0 </brick_med>
    </Matter>
  </Eloss>

  <!-- Jet Hadronization Module -->
  <JetHadronization>
    <name>colorless</name>
  </JetHadronization>

</jetscape>
```

Generate some events!

Let’s generate some pp events

Set to 200 events

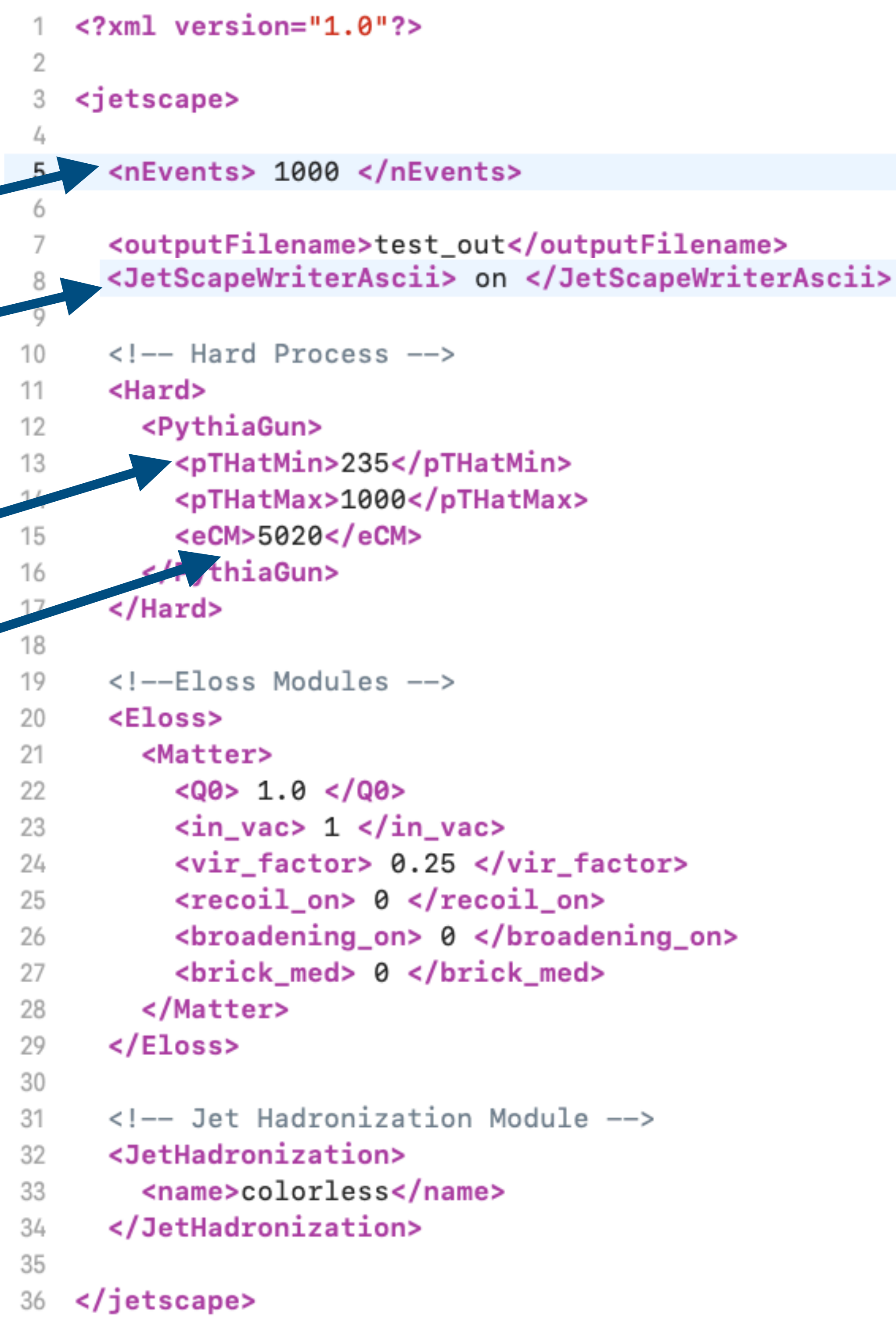
Set Ascii format

Set $\hat{p}_{\mathrm{T}}$ bin edges

Set $\sqrt{s}$

[inside container]

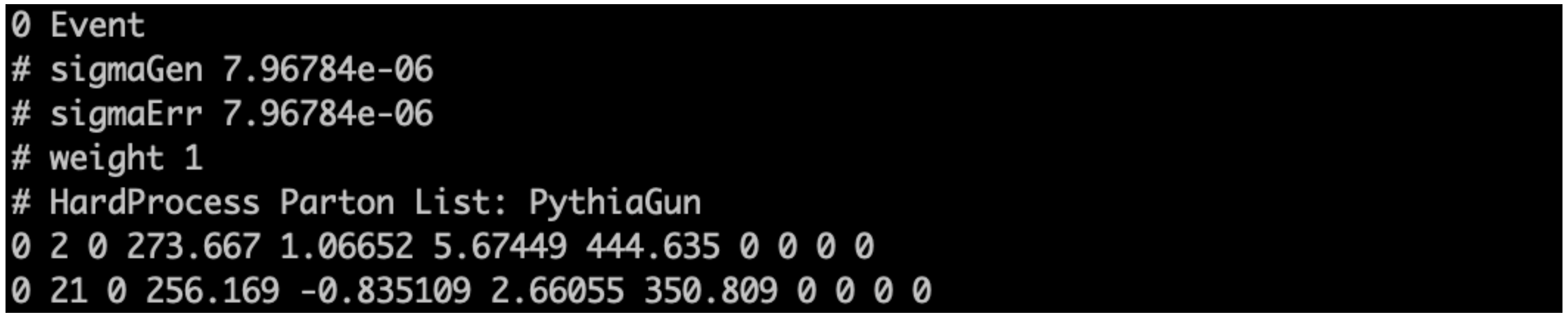
cd JETSCAPE/build
./runJetscape ../config/jetscape_user_PP19.xml



```
<?xml version="1.0"?>

<jetscape>

  <nEvents> 1000 </nEvents>

  <outputFilename>test_out</outputFilename>
  <JetScapeWriterAscii> on </JetScapeWriterAscii>

  <!-- Hard Process -->
  <Hard>
    <PythiaGun>
      <pTHatMin>235</pTHatMin>
      <pTHatMax>1000</pTHatMax>
      <eCM>5020</eCM>
    </PythiaGun>
  </Hard>

  <!--Eloss Modules -->
  <Eloss>
    <Matter>
      <Q0> 1.0 </Q0>
      <in_vac> 1 </in_vac>
      <vir_factor> 0.25 </vir_factor>
      <recoil_on> 0 </recoil_on>
      <broadening_on> 0 </broadening_on>
      <brick_med> 0 </brick_med>
    </Matter>
  </Eloss>

  <!-- Jet Hadronization Module -->
  <JetHadronization>
    <name>colorless</name>
  </JetHadronization>

</jetscape>
```

Ascii output

You should then find a file test_out.dat in the same directory

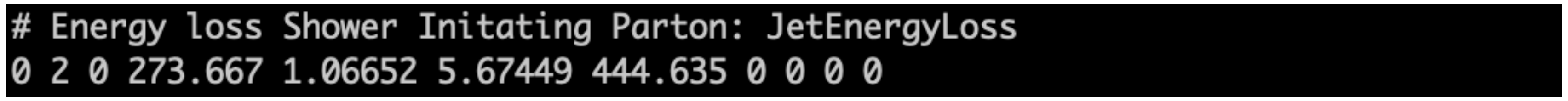


```
0 Event
# sigmaGen 7.96784e-06
# sigmaErr 7.96784e-06
# weight 1
# HardProcess Parton List: PythiaGun
0 2 0 273.667 1.06652 5.67449 444.635 0 0 0 0
0 21 0 256.169 -0.835109 2.66055 350.809 0 0 0 0
```

Event info

Initial partons

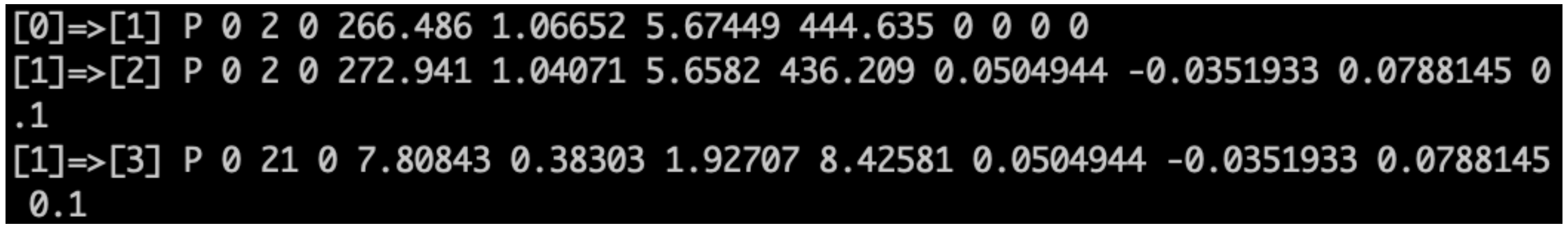
...



```
# Energy loss Shower Initating Parton: JetEnergyLoss
0 2 0 273.667 1.06652 5.67449 444.635 0 0 0 0
```

Shower-initiating partons

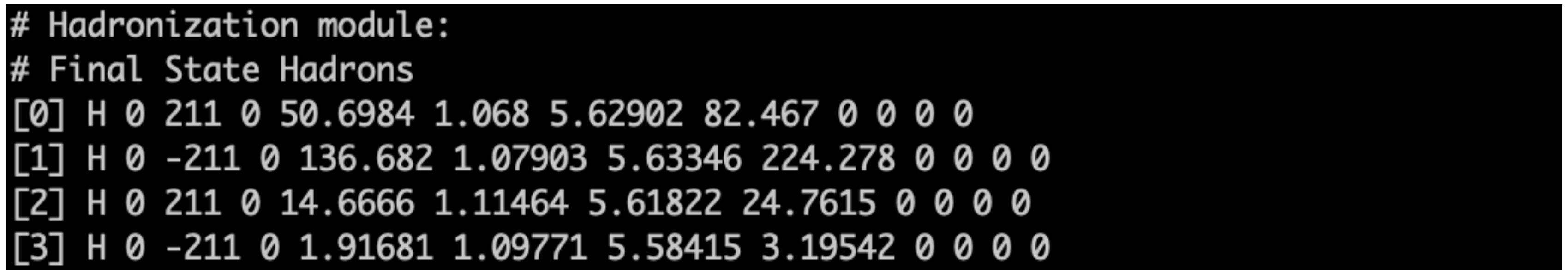
...



```
[0]=>[1] P 0 2 0 266.486 1.06652 5.67449 444.635 0 0 0 0
[1]=>[2] P 0 2 0 272.941 1.04071 5.6582 436.209 0.0504944 -0.0351933 0.0788145 0
.1
[1]=>[3] P 0 21 0 7.80843 0.38303 1.92707 8.42581 0.0504944 -0.0351933 0.0788145
 0.1
```

Parton shower history

...



```
# Hadronization module:
# Final State Hadrons
[0] H 0 211 0 50.6984 1.068 5.62902 82.467 0 0 0 0
[1] H 0 -211 0 136.682 1.07903 5.63346 224.278 0 0 0 0
[2] H 0 211 0 14.6666 1.11464 5.61822 24.7615 0 0 0 0
[3] H 0 -211 0 1.91681 1.09771 5.58415 3.19542 0 0 0 0
```

Hadrons

Generate final-state hadrons

From the Ascii output, we can generate a list of final-state hadrons

[inside container]

```
./FinalStateHadrons test_out.dat my_final_state_hadrons.txt
```

…

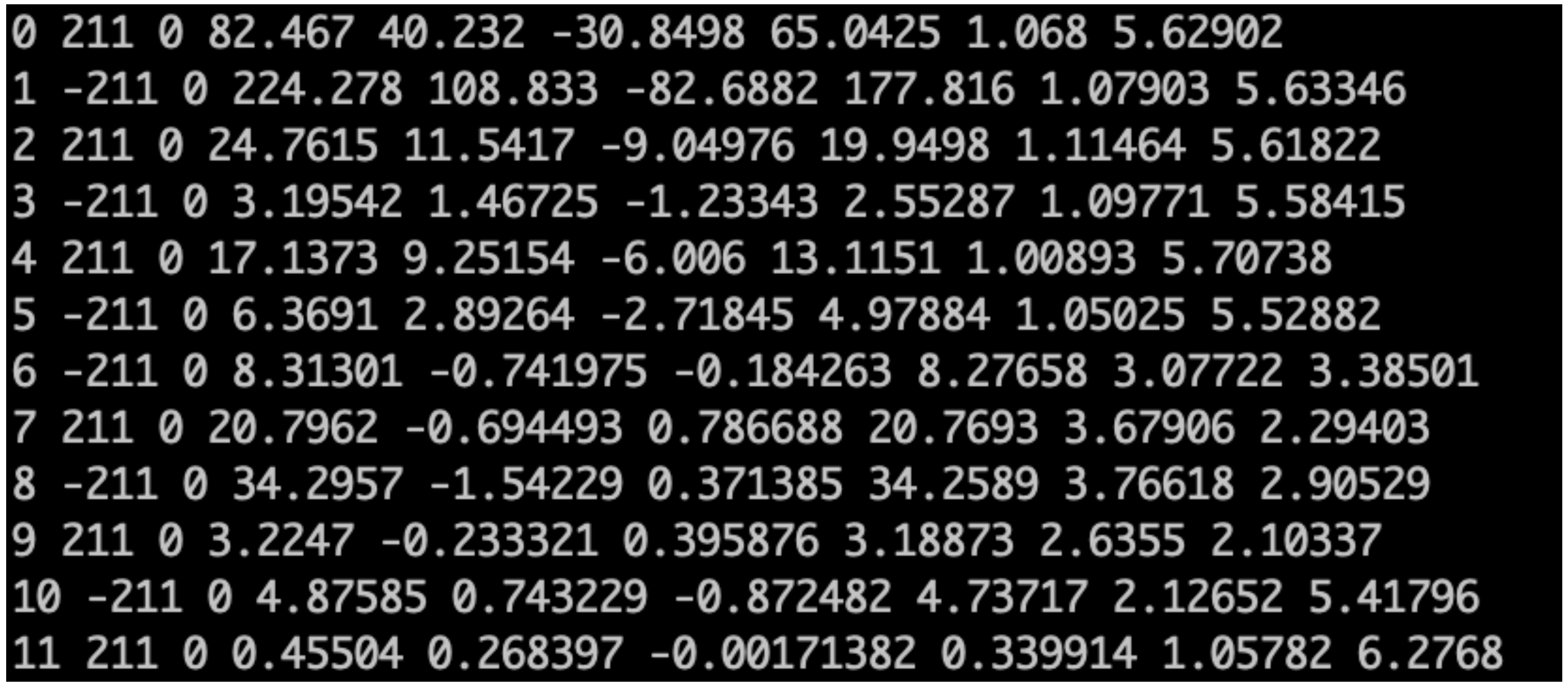
Generate final-state hadrons

From the Ascii output, we can generate a list of final-state hadrons

[inside container]

```
./FinalStateHadrons test_out.dat my_final_state_hadrons.txt
```

pdg E Px Py Pz Eta Phi



```
0 211 0 82.467 40.232 -30.8498 65.0425 1.068 5.62902
1 -211 0 224.278 108.833 -82.6882 177.816 1.07903 5.63346
2 211 0 24.7615 11.5417 -9.04976 19.9498 1.11464 5.61822
3 -211 0 3.19542 1.46725 -1.23343 2.55287 1.09771 5.58415
4 211 0 17.1373 9.25154 -6.006 13.1151 1.00893 5.70738
5 -211 0 6.3691 2.89264 -2.71845 4.97884 1.05025 5.52882
6 -211 0 8.31301 -0.741975 -0.184263 8.27658 3.07722 3.38501
7 211 0 20.7962 -0.694493 0.786688 20.7693 3.67906 2.29403
8 -211 0 34.2957 -1.54229 0.371385 34.2589 3.76618 2.90529
9 211 0 3.2247 -0.233321 0.395876 3.18873 2.6355 2.10337
10 -211 0 4.87585 0.743229 -0.872482 4.73717 2.12652 5.41796
11 211 0 0.45504 0.268397 -0.00171382 0.339914 1.05782 6.2768
```

...

Part 1

The basics of JETSCAPE

Part 2

Constructing an observable

Part 3

Implementing a custom module

Constructing an observable

You often don't just want to generate events…

You want to construct and plot a specific observable!

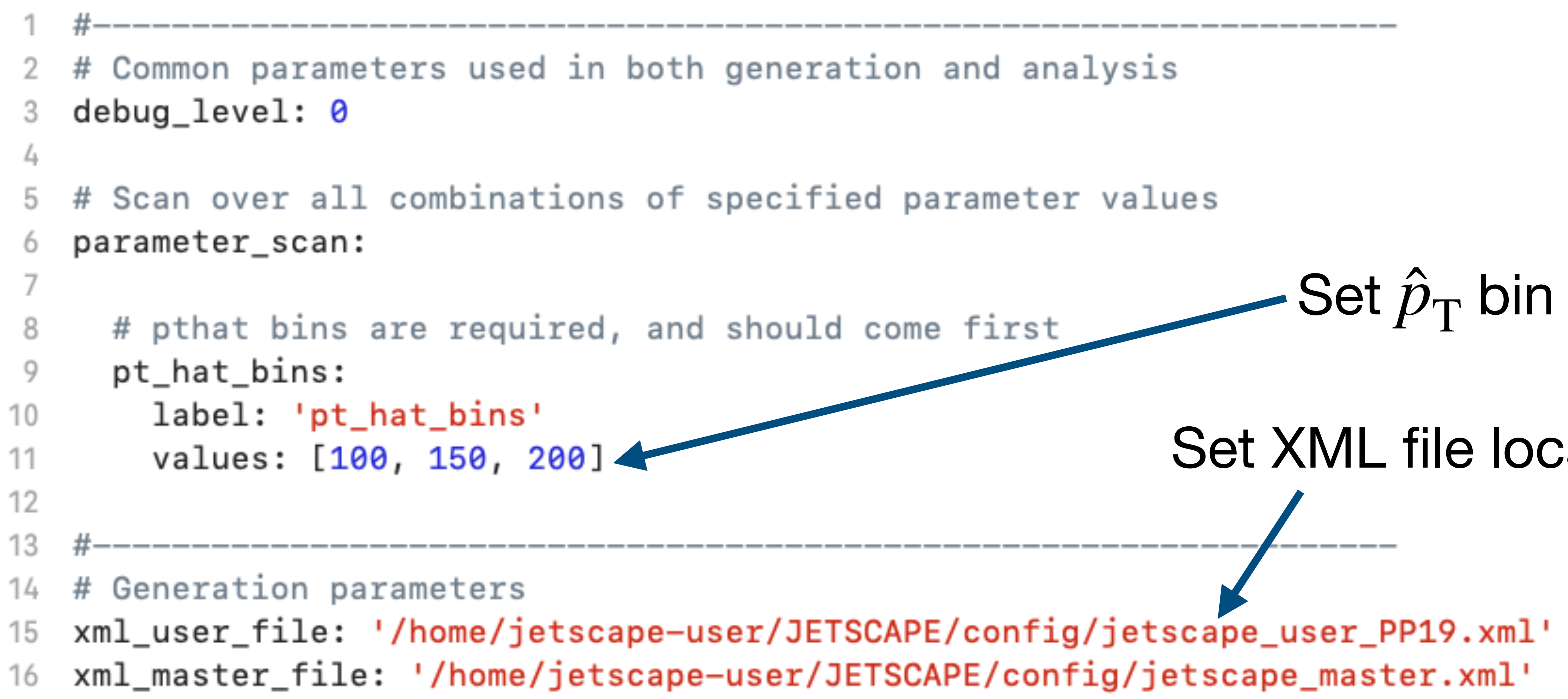
We will go through an example how one could do this:

https://github.com/jdmulligan/JETSCAPE-analysis

1. Generate JETSCAPE events for a set of $\hat{p}_{\mathrm{T}}$ bins
2. Analyze JETSCAPE output —> ROOT file

Generate events

Open JETSCAPE-analysis/config/example.yaml



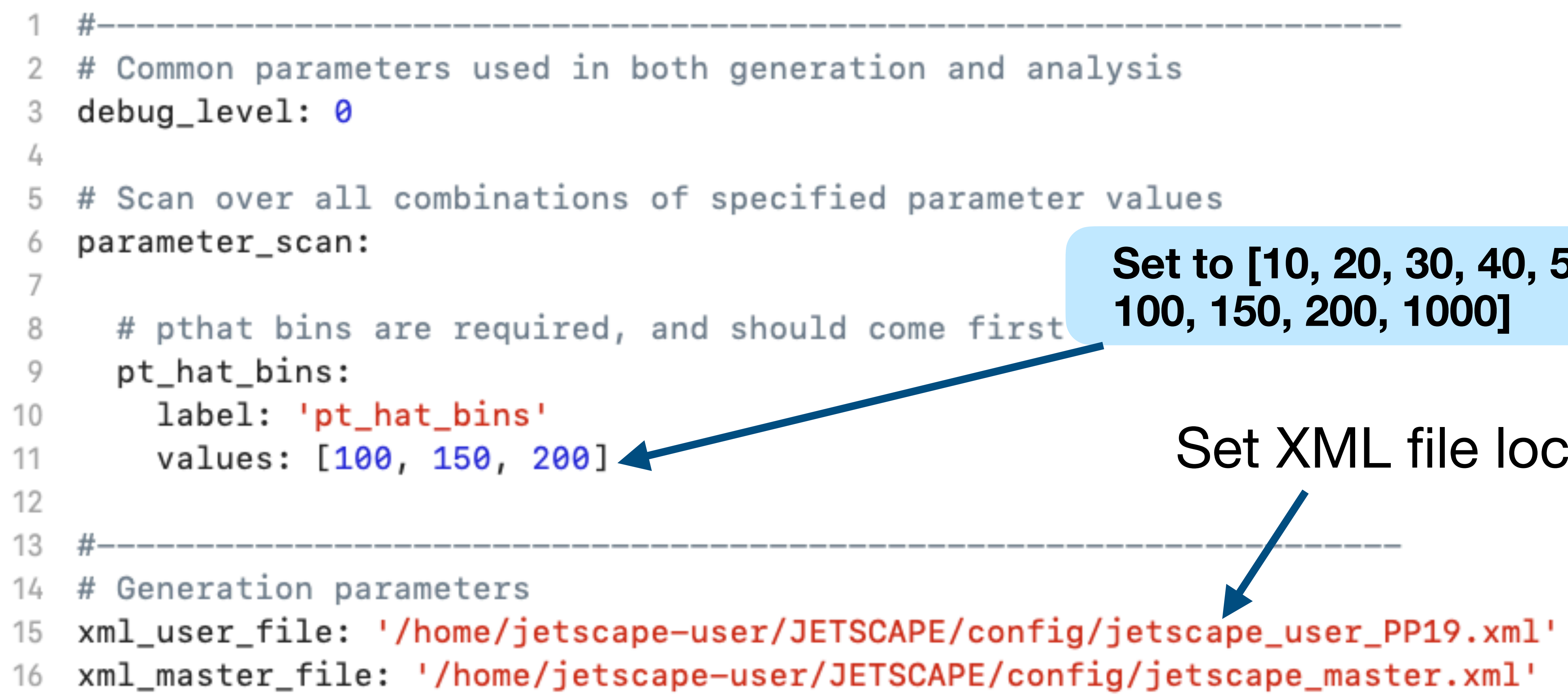
```yaml
#------------------------------------------------------------
# Common parameters used in both generation and analysis
debug_level: 0

# Scan over all combinations of specified parameter values
parameter_scan:

  # pthat bins are required, and should come first
  pt_hat_bins:
    label: 'pt_hat_bins'
    values: [100, 150, 200]

#------------------------------------------------------------
# Generation parameters
xml_user_file: '/home/jetscape-user/JETSCAPE/config/jetscape_user_PP19.xml'
xml_master_file: '/home/jetscape-user/JETSCAPE/config/jetscape_master.xml'
```

Set $\hat{p}_{\mathrm{T}}$ bin edges

Set XML file location

Generate events

Open JETSCAPE-analysis/config/example.yaml



```yaml
#------------------------------------------------------------------
# Common parameters used in both generation and analysis
debug_level: 0

# Scan over all combinations of specified parameter values
parameter_scan:

  # pthat bins are required, and should come first
  pt_hat_bins:
    label: 'pt_hat_bins'
    values: [100, 150, 200]

#------------------------------------------------------------------
# Generation parameters
xml_user_file: '/home/jetscape-user/JETSCAPE/config/jetscape_user_PP19.xml'
xml_master_file: '/home/jetscape-user/JETSCAPE/config/jetscape_master.xml'
```

Set to [10, 20, 30, 40, 50, 70, 100, 150, 200, 1000]

Set XML file location

Open the PP19 XML file and add a HepMC writer, nEvents = 500

Generate events

The script `jetscape_analysis/generate/jetscape_events.py` generates JETSCAPE events, including automated machinery to launch a set of pt-hat bins and optionally scan over any additional parameter(s).

[Inside the docker container]

```
cd JETSCAPE-analysis/jetscape_analysis/generate
python jetscape_events.py -c /home/jetscape-user/JETSCAPE-analysis/config/example.yaml
                          -o /home/jetscape-user/JETSCAPE-analysis-output
```

`-c` specifies a configuration file that should be edited to specify the pt-hat bins and JETSCAPE XML configuration paths,

`-o` specifies a location where the JETSCAPE output files will be written.

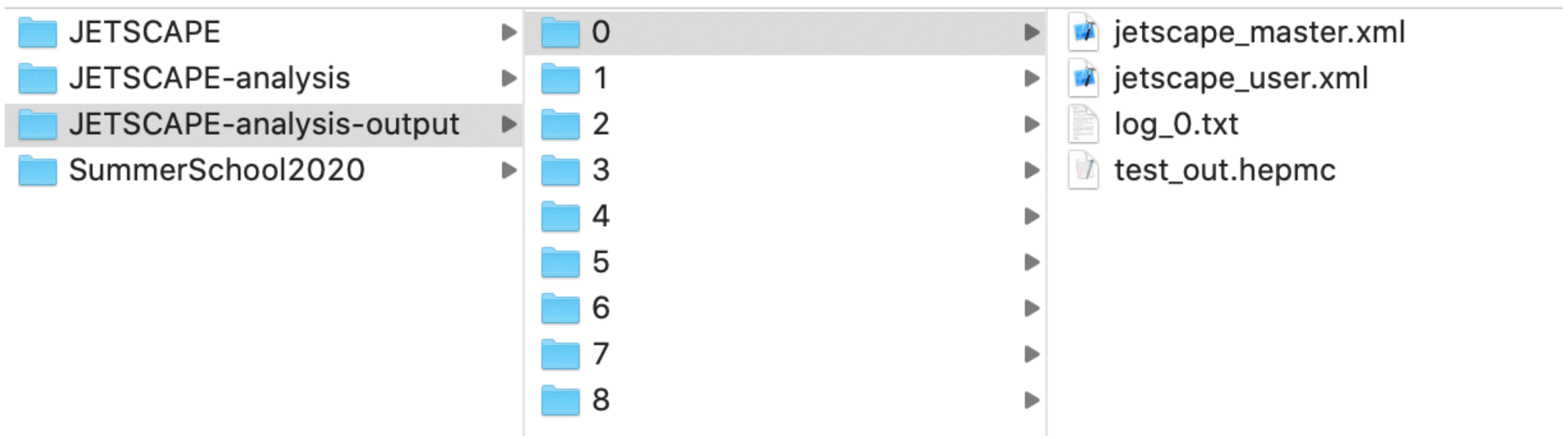
That's it! The script will write a separate sub-directory with JETSCAPE events for each pt-hat bin.

Generate events

The script `jetscape_analysis/generate/jetscape_events.py` generates JETSCAPE events, including automated machinery to launch a set of pt-hat bins and optionally scan over any additional parameter(s).

[Inside the docker container]

```
cd JETSCAPE-analysis/jetscape_analysis/generate
python jetscape_events.py -c /home/jetscape-user/JETSCAPE-analysis/config/example.yaml
                          -o /home/jetscape-user/JETSCAPE-analysis-output
```



```
JETSCAPE                  ▶   0  ▶   jetscape_master.xml
JETSCAPE-analysis         ▶   1  ▶   jetscape_user.xml
JETSCAPE-analysis-output  ▶   2  ▶   log_0.txt
SummerSchool2020          ▶   3  ▶   test_out.hepmc
                              4  ▶
                              5  ▶
                              6  ▶
                              7  ▶
                              8  ▶
```

Analyze events

We provide a simple framework to loop over the generated JETSCAPE output files, perform physics analysis, and produce a ROOT file. It also contains machinery to aggregate the results from the set of pt-hat bins, and plot the analysis results.

Open JETSCAPE-analysis/jetscape_analysis/analysis/analyze_events_example.py

```python
# ---------------------------------------------
# Initialize output objects
# ---------------------------------------------
def initialize_user_output_objects(self):

    # Hadron histograms
    hname = 'hChHadronPt'
    h = ROOT.TH1F(hname, hname, 100, 0, 100)
    h.Sumw2()
    setattr(self, hname, h)

    # Jet histograms
    for jetR in self.jetR_list:

        hname = 'hJetPt_R{}'.format(jetR)
        h = ROOT.TH1F(hname, hname, 300, 0, 300)
        setattr(self, hname, h)
```

Initialize some jet histograms

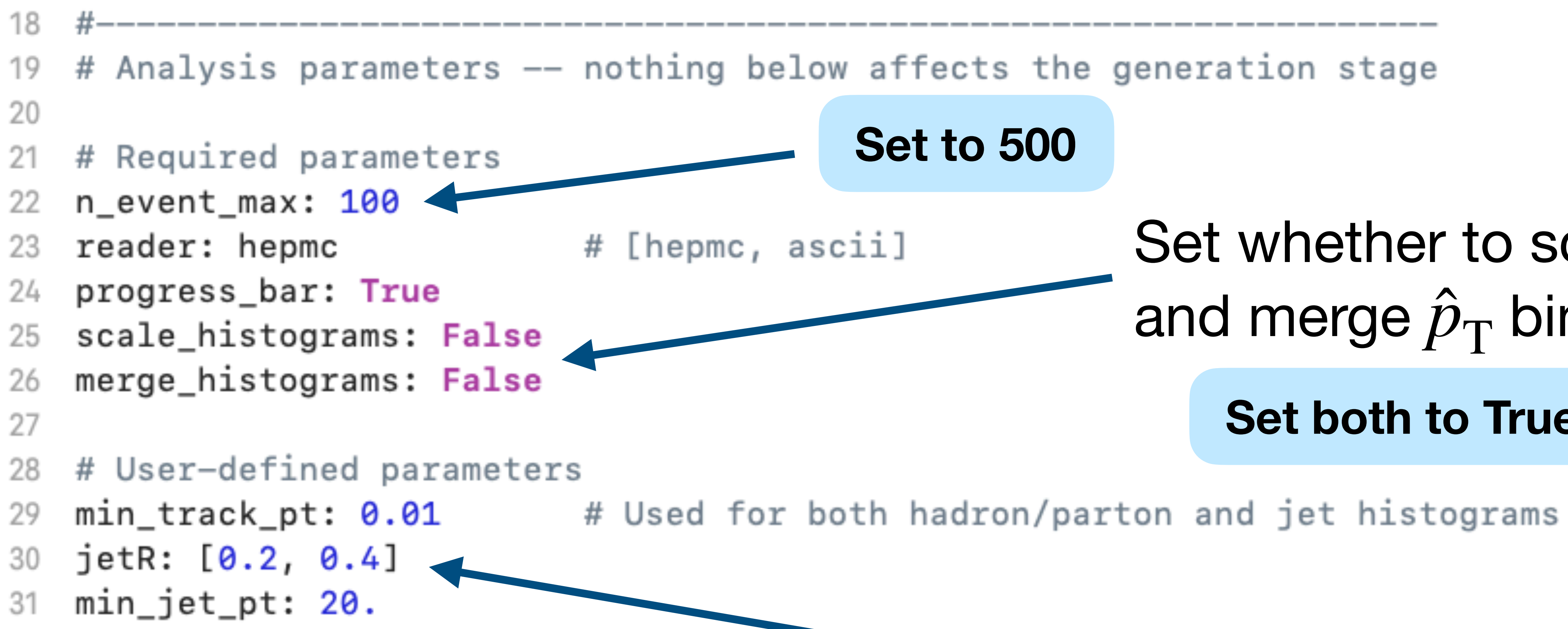
Get hadrons from JETSCAPE output

```python
# ---------------------------------------------------------------
# Analyze a single event -- fill user-defined output objects
# ---------------------------------------------------------------
def analyze_event(self, event):

    # Get list of hadrons from the event, and fill some histograms
    hadrons = event.hadrons(min_track_pt=self.min_track_pt)
    self.fill_hadron_histograms(hadrons)

    # Create list of fastjet::PseudoJets
    fj_hadrons = self.fill_fastjet_constituents(hadrons)

    # Loop through specified jet R
    for jetR in self.jetR_list:

        # Set jet definition and a jet selector
        jet_def = fj.JetDefinition(fj.antikt_algorithm, jetR)
        jet_selector = fj.SelectorPtMin(self.min_jet_pt) & fj.SelectorAbsRapMax(5.)
        if self.debug_level > 0:
            print('jet definition is:', jet_def)
            print('jet selector is:', jet_selector, '\n')

        # Do jet finding
        cs = fj.ClusterSequence(fj_hadrons, jet_def)
        jets = fj.sorted_by_pt(cs.inclusive_jets())
        jets_selected = jet_selector(jets)

        # Fill some jet histograms
        self.fill_jet_histograms(jets_selected, jetR)
```

Analyze events

We provide a simple framework to loop over the generated JETSCAPE output files, perform physics analysis, and produce a ROOT file. It also contains machinery to aggregate the results from the set of pt-hat bins, and plot the analysis results.

Open JETSCAPE-analysis/config/example.yaml



```
#----------------------------------------------------------------
# Analysis parameters -- nothing below affects the generation stage

# Required parameters
n_event_max: 100
reader: hepmc                 # [hepmc, ascii]
progress_bar: True
scale_histograms: False
merge_histograms: False

# User-defined parameters
min_track_pt: 0.01            # Used for both hadron/parton and jet histograms
jetR: [0.2, 0.4]
min_jet_pt: 20.
```

Set to 500

Set whether to scale and merge $\hat{p}_{\mathrm{T}}$ bins

Set both to True

Some analysis parameters

Analyze events

[Inside the docker container]

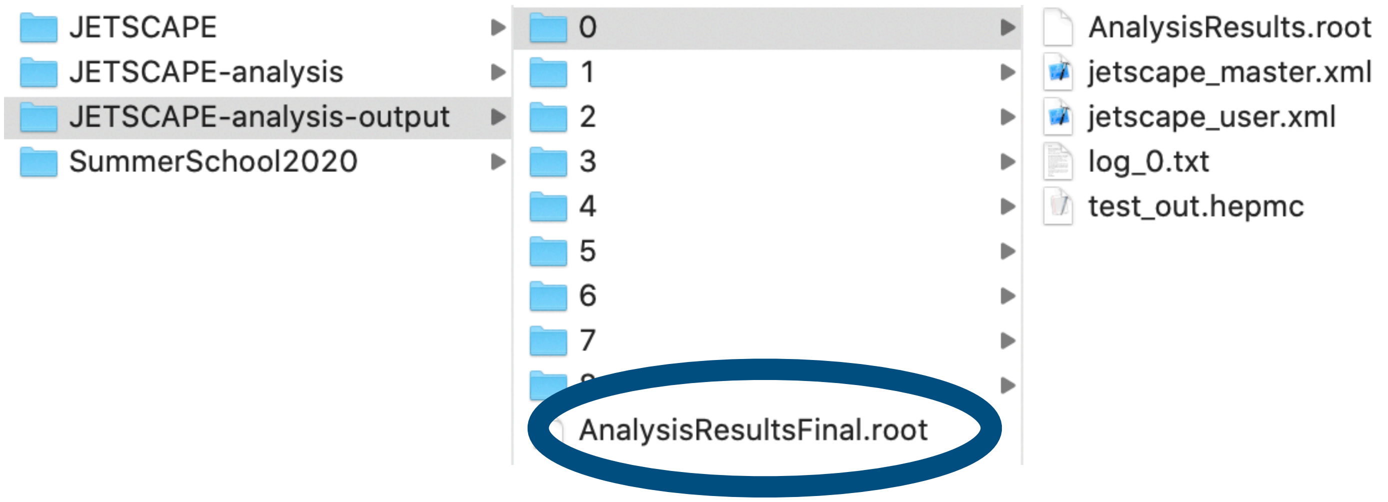
```
cd JETSCAPE-analysis
source init.sh

cd jetscape_analysis/analysis
python analyze_events_example.py -c ../../config/example.yaml
                                 -i /home/jetscape-user/JETSCAPE-analysis-output
                                 -o /home/jetscape-user/JETSCAPE-analysis-output
```

`-c` specifies a configuration file that should be edited to specify the pt-hat bins and analysis parameters,

`-i` specifies is the directory containing the generated JETSCAPE events,

`-o` specifies a location where the analysis output will be written.

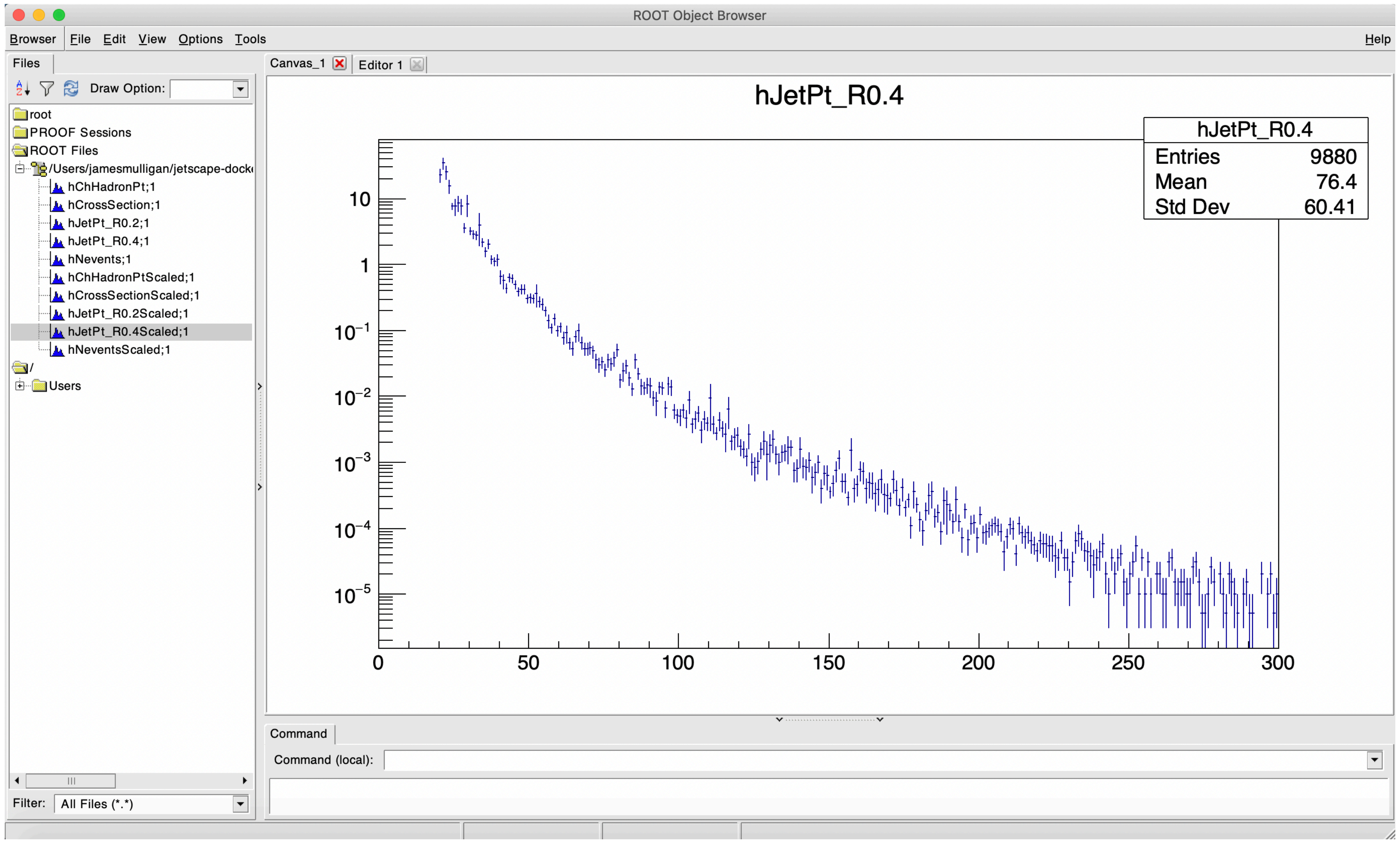
Analyze events



```
JETSCAPE                    0                        AnalysisResults.root
JETSCAPE-analysis           1                        jetscape_master.xml
JETSCAPE-analysis-output    2                        jetscape_user.xml
SummerSchool2020            3                        log_0.txt
                            4                        test_out.hepmc
                            5
                            6
                            7
                            8
                            AnalysisResultsFinal.root
```

[outside the container]

rootbrowse /path/to/AnalysisResultsFinal.root

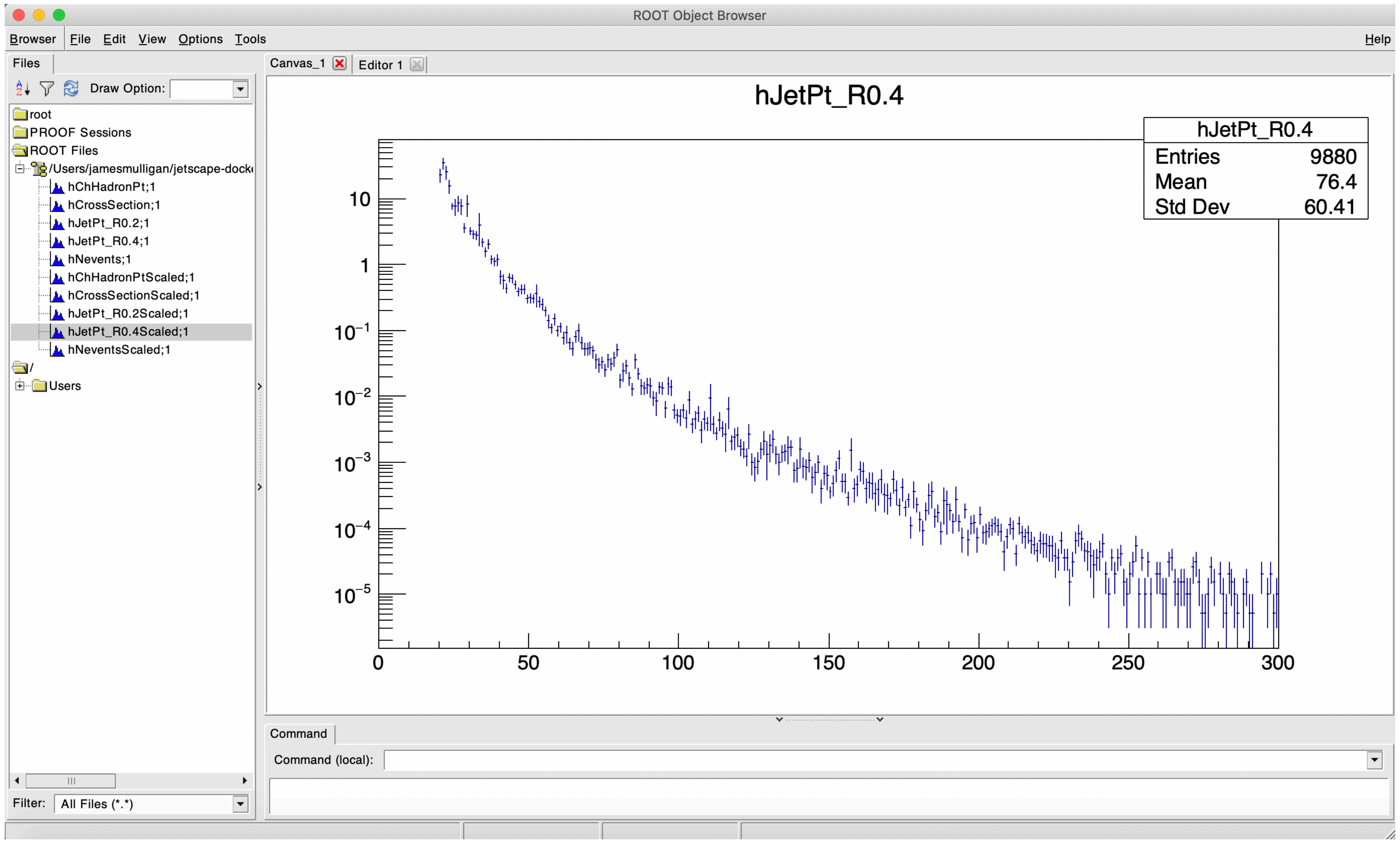


```
root
PROOF Sessions
ROOT Files
/Users/jamesmulligan/jetscape-dock
    hChHadronPt;1
    hCrossSection;1
    hJetPt_R0.2;1
    hJetPt_R0.4;1
    hNevents;1
    hChHadronPtScaled;1
    hCrossSectionScaled;1
    hJetPt_R0.2Scaled;1
    hJetPt_R0.4Scaled;1
    hNeventsScaled;1
/
Users
```

Analyze events

You can easily customize *analyze_events_example.py* by creating and filling your own histograms / trees

For further details, see https://github.com/jdmulligan/JETSCAPE-analysis



```
hJetPt_R0.4
Entries     9880
Mean        76.4
Std Dev    60.41
```

Part 1

The basics of JETSCAPE

Part 2

Constructing an observable

Part 3

Implementing a custom module

Writing a custom module

To develop a new JETSCAPE module, you should inherit from the relevant base class (InitialState, JetEnergyLoss, etc.) and implement the relevant initialization and execution functions, described in detail in The JETSCAPE framework Section 3.3.

First, update the SummerSchool 2020 repository

[Outside the docker container]

```
cd SummerSchool2020
git pull
```

Note: You may need a github account — which is highly recommended.
If you haven't set this up, you can also do:

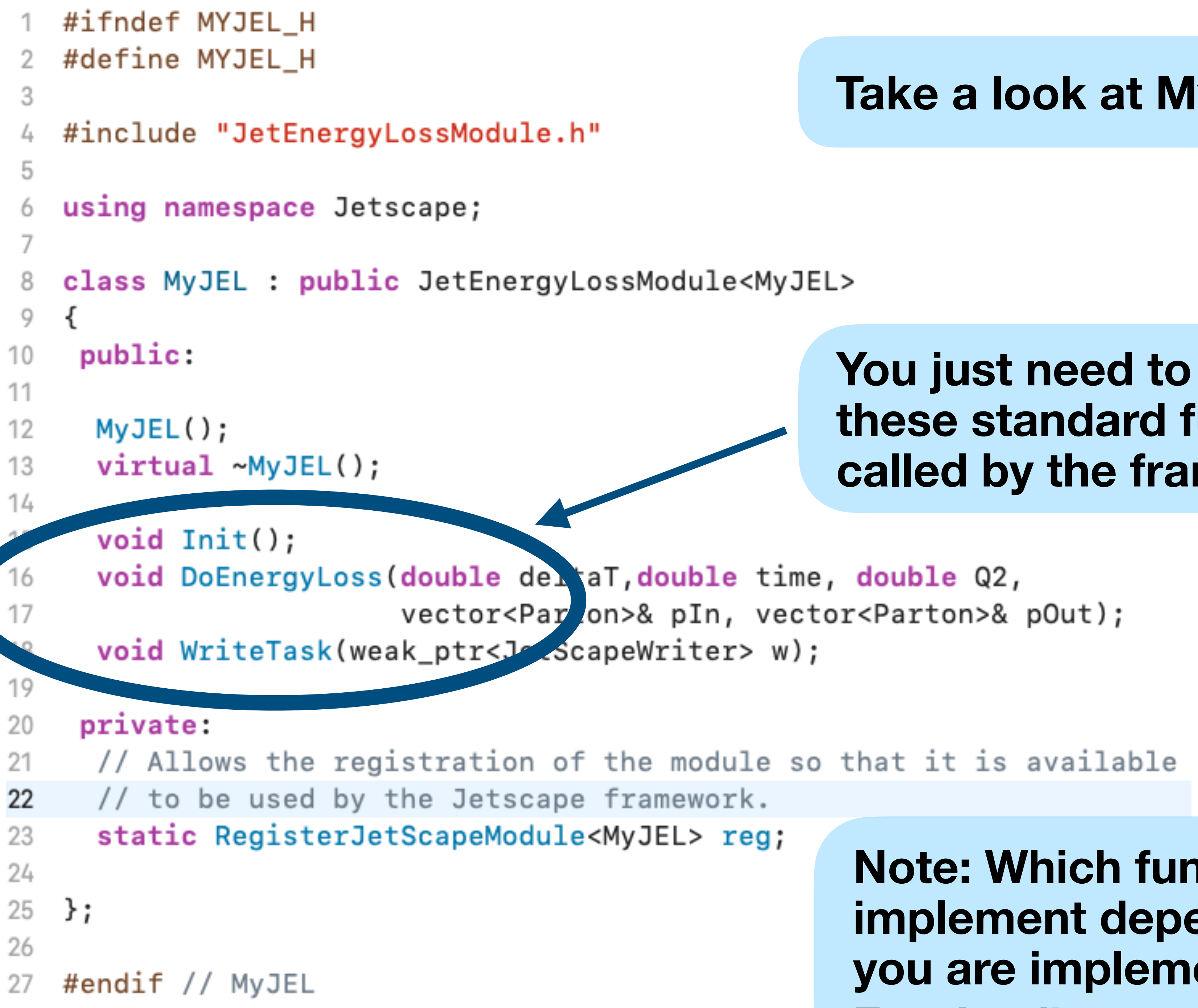
```
rm -r SummerSchool 2020
git clone https://github.com/JETSCAPE/SummerSchool2020.git
```

Then, copy the example custom module into the JETSCAPE src code

[Outside the docker container]

```
cp SummerSchool2020/framework_session/MyJEL* JETSCAPE/src/jet
```

Writing a custom module



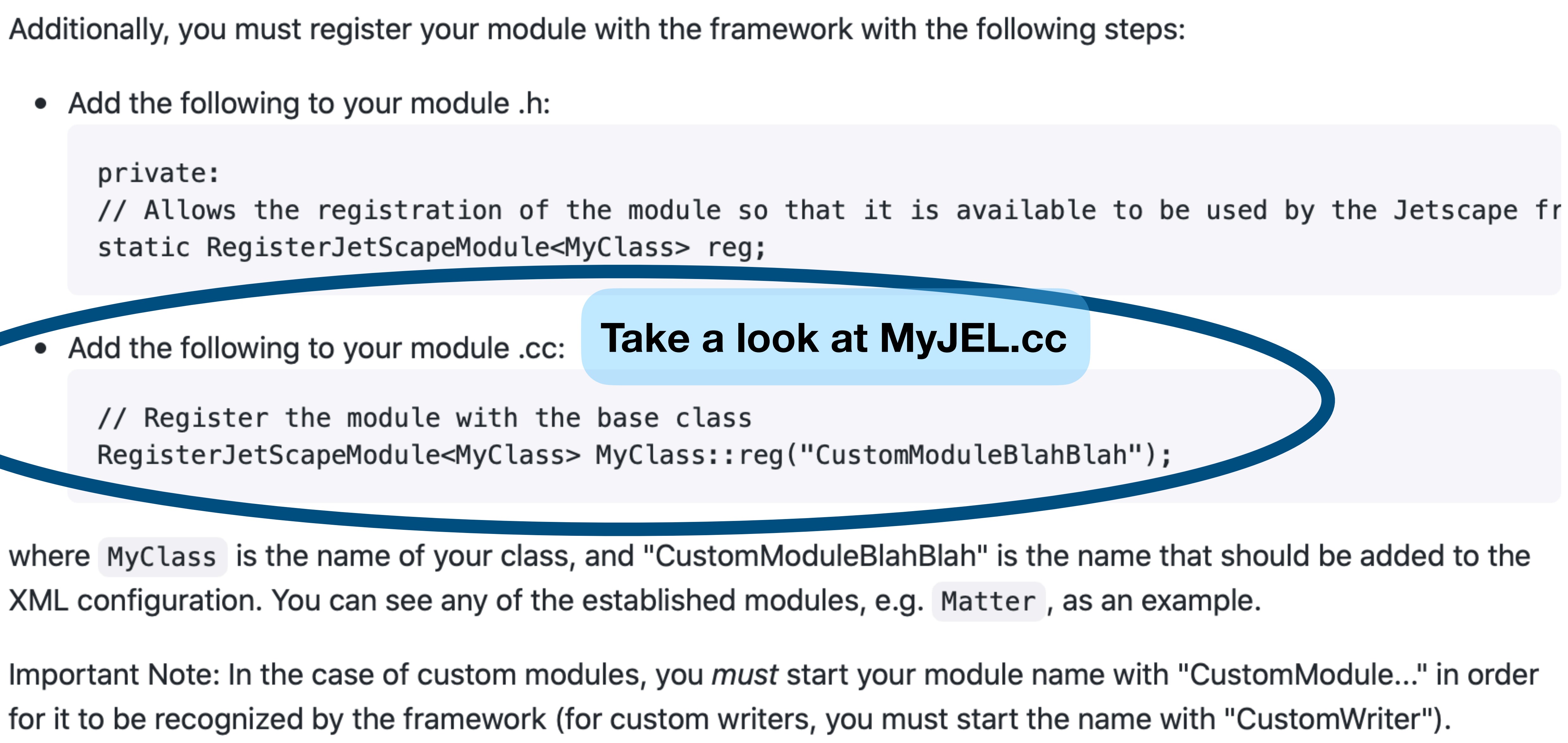
```
#ifndef MYJEL_H
#define MYJEL_H

#include "JetEnergyLossModule.h"

using namespace Jetscape;

class MyJEL : public JetEnergyLossModule<MyJEL>
{
 public:

  MyJEL();
  virtual ~MyJEL();

  void Init();
  void DoEnergyLoss(double deltaT,double time, double Q2,
                    vector<Parton>& pIn, vector<Parton>& pOut);
  void WriteTask(weak_ptr<JetScapeWriter> w);

 private:
  // Allows the registration of the module so that it is available
  // to be used by the Jetscape framework.
  static RegisterJetScapeModule<MyJEL> reg;

};

#endif // MyJEL
```

Take a look at MyJEL.h

You just need to implement your physics in these standard functions, which will be called by the framework

Note: Which function(s) you need to implement depends on what type of module you are implementing!
For details, see arXiv:1903.07706

Writing a custom module



Additionally, you must register your module with the framework with the following steps:

- Add the following to your module .h:

```
private:
// Allows the registration of the module so that it is available to be used by the Jetscape fr
static RegisterJetScapeModule<MyClass> reg;
```

- Add the following to your module .cc:

Take a look at MyJEL.cc

```
// Register the module with the base class
RegisterJetScapeModule<MyClass> MyClass::reg("CustomModuleBlahBlah");
```

where `MyClass` is the name of your class, and "CustomModuleBlahBlah" is the name that should be added to the XML configuration. You can see any of the established modules, e.g. `Matter`, as an example.

Important Note: In the case of custom modules, you *must* start your module name with "CustomModule..." in order for it to be recognized by the framework (for custom writers, you must start the name with "CustomWriter").

Build your custom module

[Inside the docker container]

```
cd JETSCAPE/build
cmake ..
make
```

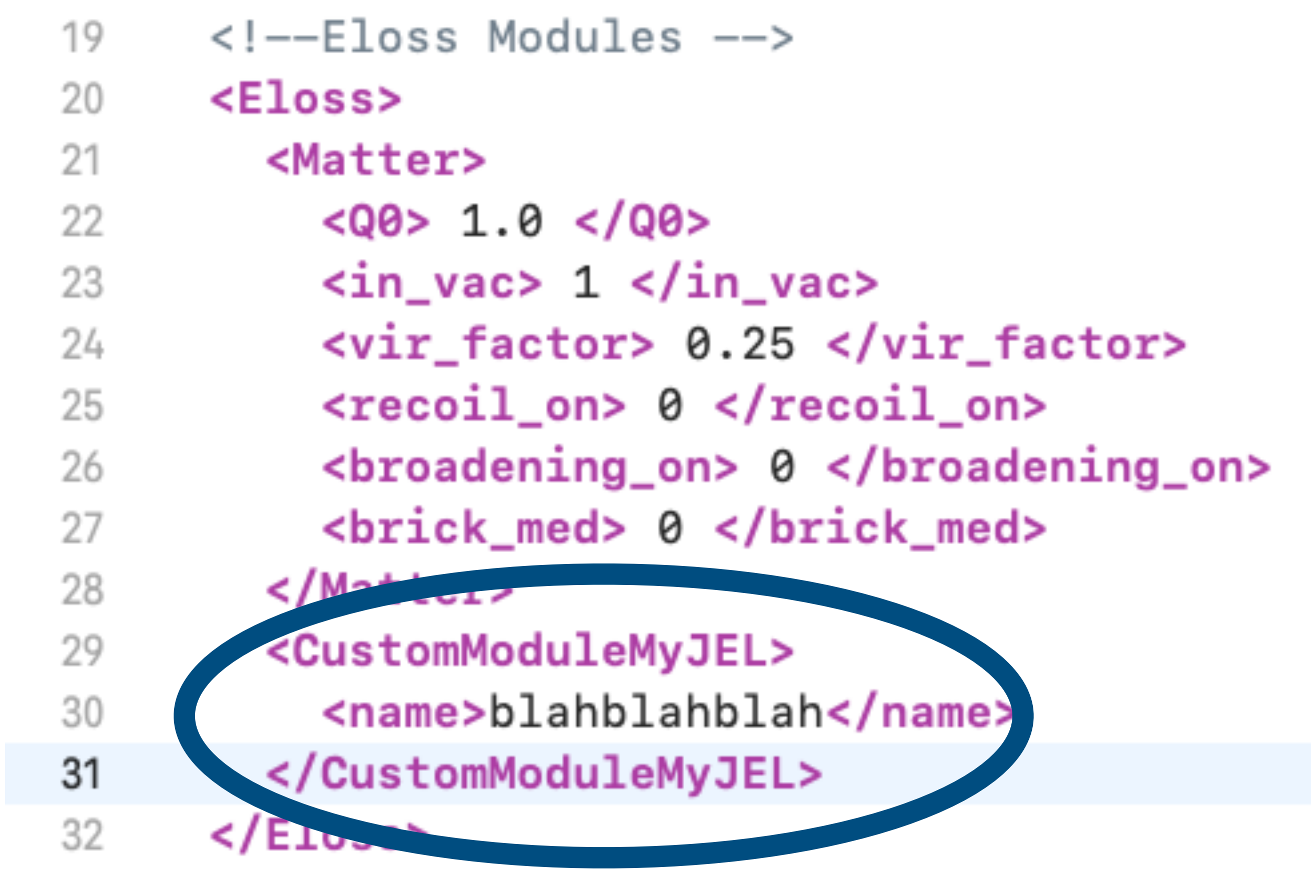
Build your custom module

[Inside the docker container]

```
cd JETSCAPE/build
cmake ..
make
```

Edit jetscape_user_PP19.xml
to add your module

It will then automatically be run by the framework

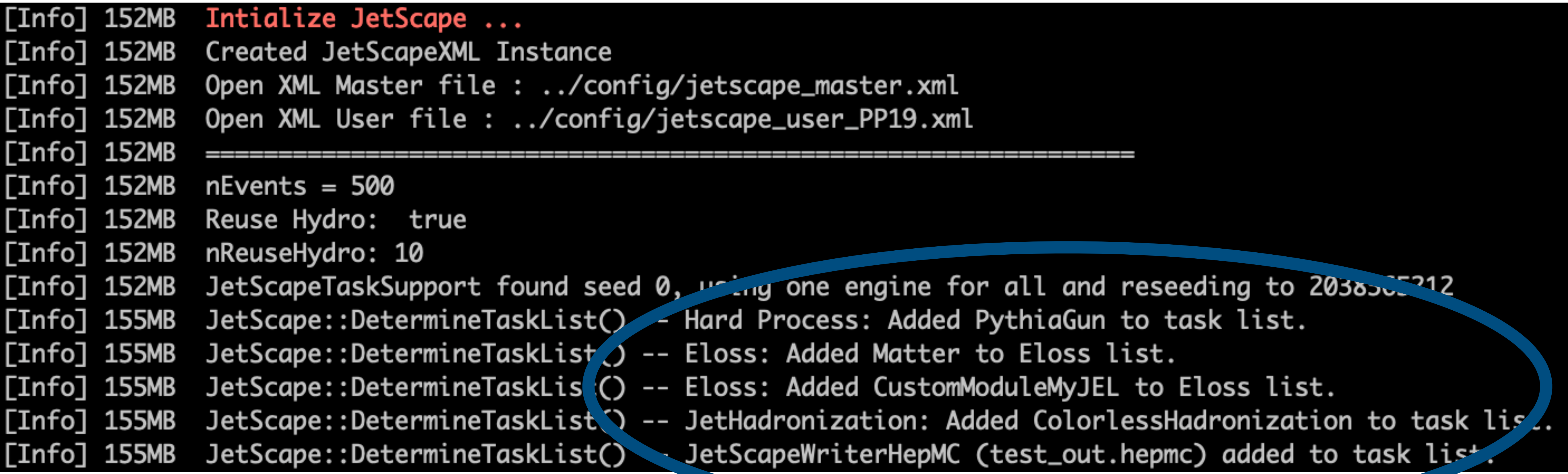


```
<!--Eloss Modules -->
<Eloss>
  <Matter>
    <Q0> 1.0 </Q0>
    <in_vac> 1 </in_vac>
    <vir_factor> 0.25 </vir_factor>
    <recoil_on> 0 </recoil_on>
    <broadening_on> 0 </broadening_on>
    <brick_med> 0 </brick_med>
  </Matter>
  <CustomModuleMyJEL>
    <name>blahblahblah</name>
  </CustomModuleMyJEL>
</Eloss>
```

Run your custom module

[Inside the docker container]

./runJetscape ../config/jetscape_user_PP19.xml



```
[Info] 152MB  Intialize JetScape ...
[Info] 152MB  Created JetScapeXML Instance
[Info] 152MB  Open XML Master file : ../config/jetscape_master.xml
[Info] 152MB  Open XML User file : ../config/jetscape_user_PP19.xml
[Info] 152MB  ================================================================
[Info] 152MB  nEvents = 500
[Info] 152MB  Reuse Hydro:  true
[Info] 152MB  nReuseHydro: 10
[Info] 152MB  JetScapeTaskSupport found seed 0, using one engine for all and reseeding to 2038[illegible]212
[Info] 155MB  JetScape::DetermineTaskList() -- Hard Process: Added PythiaGun to task list.
[Info] 155MB  JetScape::DetermineTaskList() -- Eloss: Added Matter to Eloss list.
[Info] 155MB  JetScape::DetermineTaskList() -- Eloss: Added CustomModuleMyJEL to Eloss list.
[Info] 155MB  JetScape::DetermineTaskList() -- JetHadronization: Added ColorlessHadronization to task list.
[Info] 155MB  JetScape::DetermineTaskList() -- JetScapeWriterHepMC (test_out.hepmc) added to task list.
```

Success!

Homework

In preparation for the physics sessions, please complete the following two slides before tomorrow's session

This is crucial for the upcoming physics sessions

Update JETSCAPE

There have been a couple recent updates — let's get the latest version of JETSCAPE

[outside container]

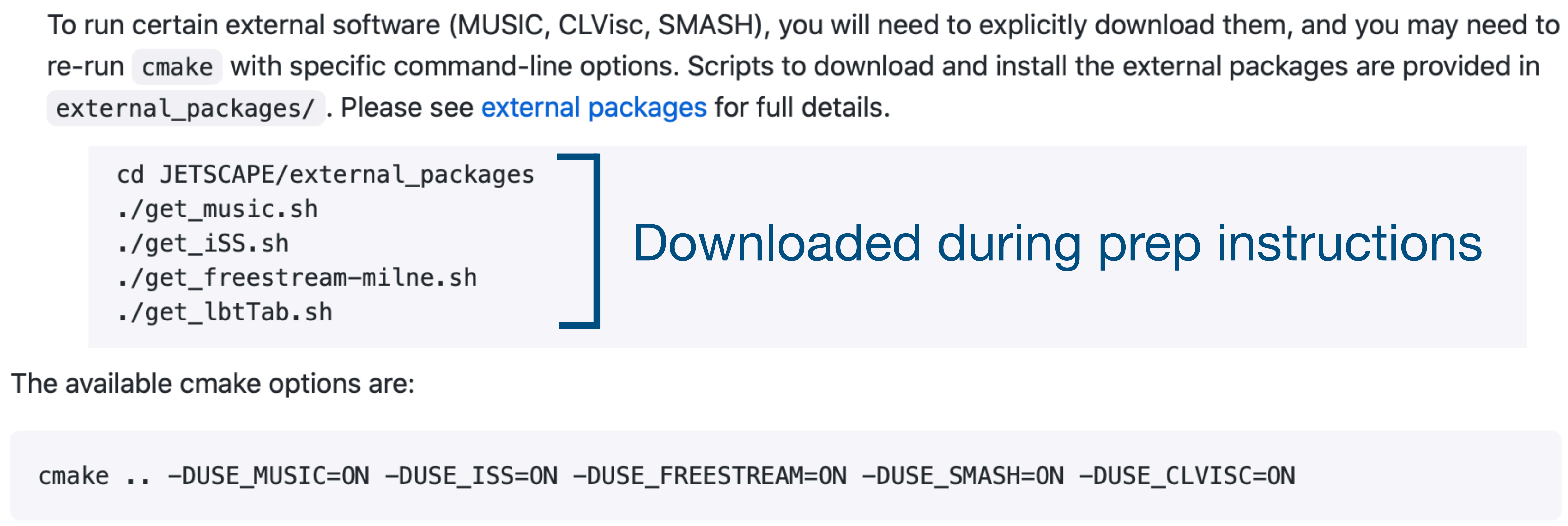
```
cd JETSCAPE
git pull
```

Note: You may need a github account — which is highly recommended.
If you haven't set this up, you can also do:

```
rm -r JETSCAPE
git clone https://github.com/JETSCAPE/JETSCAPE.git
```

Build JETSCAPE

with external packages enabled



To run certain external software (MUSIC, CLVisc, SMASH), you will need to explicitly download them, and you may need to re-run `cmake` with specific command-line options. Scripts to download and install the external packages are provided in `external_packages/`. Please see external packages for full details.

```
cd JETSCAPE/external_packages
./get_music.sh
./get_iSS.sh
./get_freestream-milne.sh
./get_lbtTab.sh
```

Downloaded during prep instructions

The available cmake options are:

```
cmake .. -DUSE_MUSIC=ON -DUSE_ISS=ON -DUSE_FREESTREAM=ON -DUSE_SMASH=ON -DUSE_CLVISC=ON
```

[inside container]

```
cd JETSCAPE/build
cmake .. -DUSE_MUSIC=ON -DUSE_ISS=ON -DUSE_FREESTREAM=ON

make -j4    # Builds using 4 cores; adapt as appropriate
```

The End!

Thank you to all of the TAs and chairs!