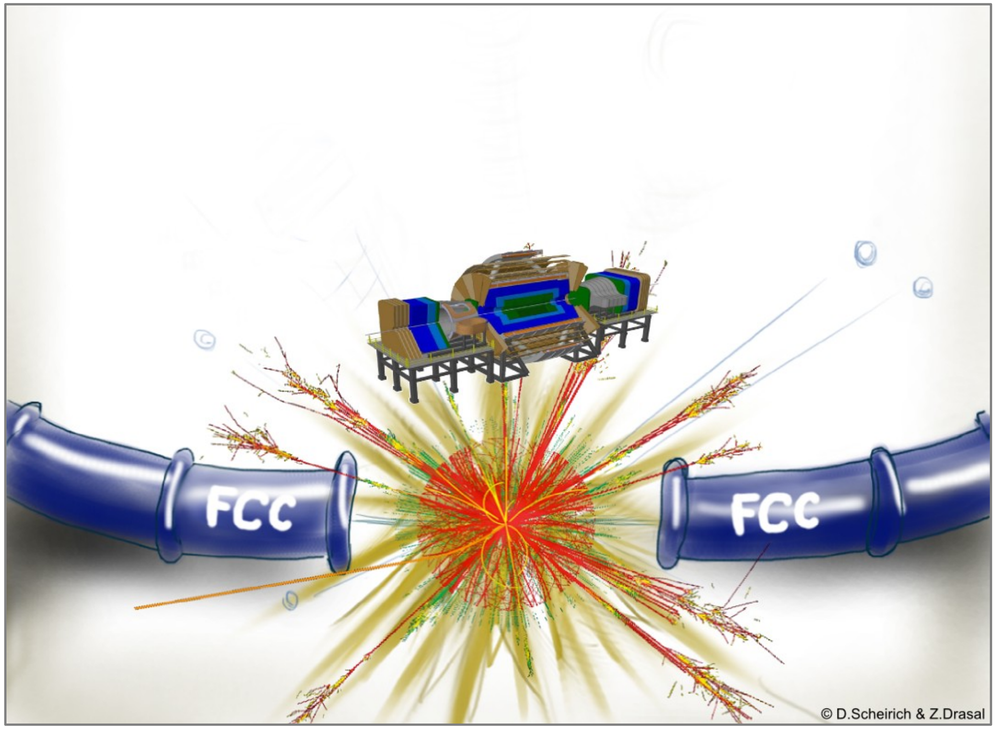
How to Start with Delphes within FCCSW



© D.Scheirich & Z.Drasal

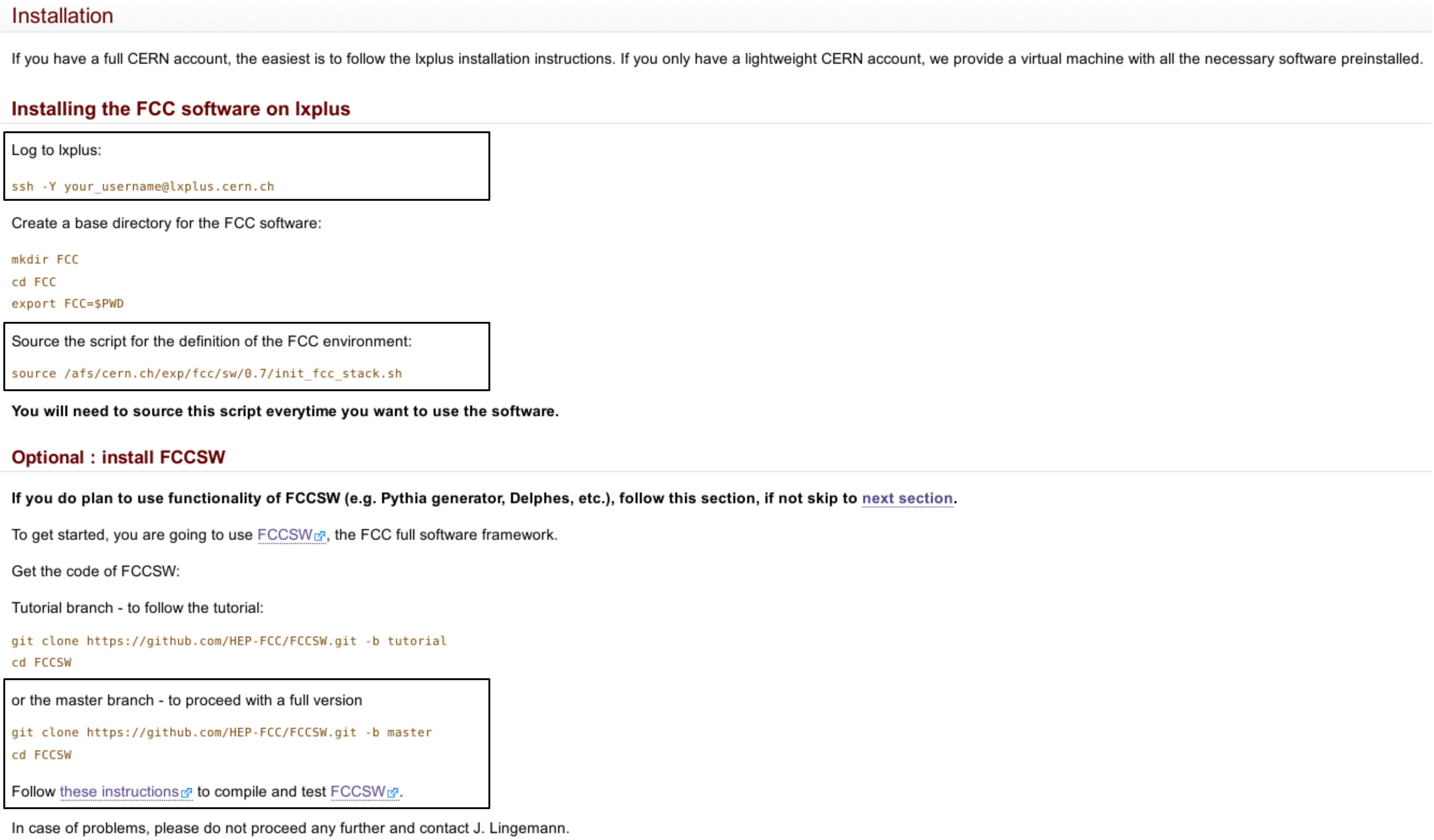
Zbyněk Drásal

CERN

Overview

- **FCCSW – Short How-To?**
- **Delphes within FCCSW**
  - Strategy & processing sequence
  - Input/Output using FCC-EDM (Event-Data model)
  - Configuration files
  - Documentation

FCCSW HowTo? Follow FCCSW Twiki



Installation

If you have a full CERN account, the easiest is to follow the lxplus installation instructions. If you only have a lightweight CERN account, we provide a virtual machine with all the necessary software preinstalled.

Installing the FCC software on lxplus

Log to lxplus:

```
ssh -Y your_username@lxplus.cern.ch
```

Create a base directory for the FCC software:

```
mkdir FCC
cd FCC
export FCC=$PWD
```

Source the script for the definition of the FCC environment:

```
source /afs/cern.ch/exp/fcc/sw/0.7/init_fcc_stack.sh
```

You will need to source this script everytime you want to use the software.

Optional : install FCCSW

If you do plan to use functionality of FCCSW (e.g. Pythia generator, Delphes, etc.), follow this section, if not skip to next section.

To get started, you are going to use FCCSW, the FCC full software framework.

Get the code of FCCSW:

Tutorial branch - to follow the tutorial:

```
git clone https://github.com/HEP-FCC/FCCSW.git -b tutorial
cd FCCSW
```

or the master branch - to proceed with a full version

```
git clone https://github.com/HEP-FCC/FCCSW.git -b master
cd FCCSW
```

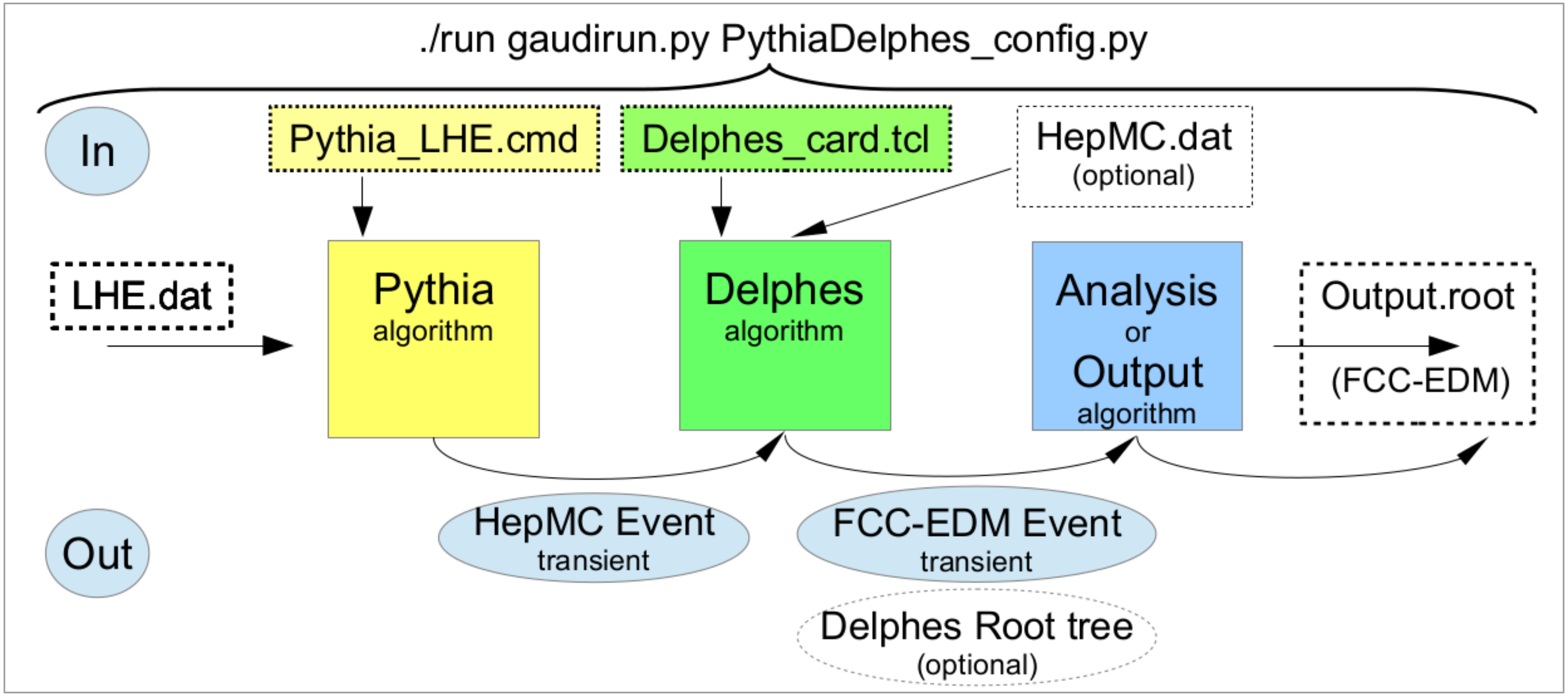
Follow these instructions to compile and test FCCSW.

In case of problems, please do not proceed any further and contact J. Lingemann.

https://twiki.cern.ch/twiki/bin/viewauth/FCC/FccSoftwareGettingStarted

Strategy & Processing Sequence

- FCCSW → based on modular structure of Gaudi framework
- Strategy:
  - modularize Pythia & Delphes as Gaudi algorithms
  - use PythiaDelphes_config.py - FCCSW command file (python script) to run Pythia & Delphes → set all GAUDI parameters to run Pythia, Delphes & Output module



Details of FCCSW Config File

- **PythiaDelphes_config.py**:
  - Defines a run sequence of Gaudi modules (Pythia+Delphes) through a Python script
  - Variables to be arranged:
    - `nEvents` → Events to be simulated
    - `messageLevel` --> GAUDI messaging verbosity: ERROR, WARNING, INFO, DEBUG
    - `pythiaconfFile` → Pythia config file: `Pythia_LHEinput.cmd/Pythia_standard.cmd`
    - `delphesCard` → Delphes TCL configuration file (use official Delphes card)
    - `delphesHepMCInFile` → Delphes input file (use "" to read HepMC directly from Pythia module, i.e. from transient data store)
    - `delphesRootOutFile` → Delphes output file (use "" to output data to transient data store → FCC-EDM objects automatically written out through Gaudi `out` module!
    - `delphes???OutArray` → Define which Delphes module objects are processed as FCC-EDM (Event Data Model) particles, where `???` stands for `muons, electrons, photons` …

Pythia Input/Output & Configuration

- **Pythia Data Input:**

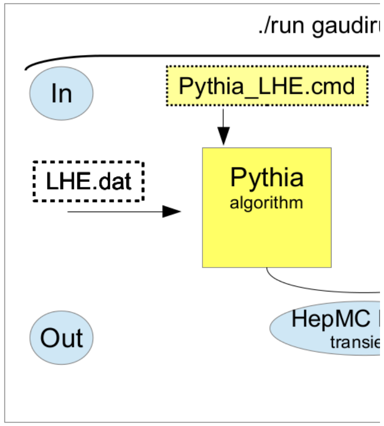
  1) **LHE (Les Houches Event)** data file (from e.g. Madgraph, ...)

     → Already simulated Les Houches Events are processed by Pythia

     → Pythia performs MPI, ISR, FSR, hadronization, decays ...

  2) No input, **Pythia simulated events** directly used

     → Pythia simulates physics events & performs MPI, ISR, FSR, ...



- **Pythia Data Output:**

  – Data output through transient memory data store using **HepMC event data format**

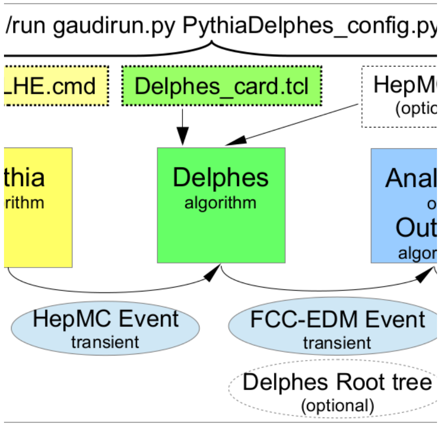
- **Pythia Configuration Files – 2 use cases prepared:**

  – **Pythia_LHEinput.cmd**: use Pythia-module to read-in the LHE file (generated by Madgraph, etc.), set Pythia run parameters

  – **Pythia_standard.cmd**: use Pythia-module to simulate physics events directly, define physics process to be simulated + Pythia run parameters

Delphes Input/Output & Configuration

- **Delphes Data Input:**
  - The Delphes data input is through the transient event data store in HepMC format (in principal, HepMC data file can be read-in too) → input read-in by `DelphesExtHepMCReader`
- **Delphes Output:**
  - The Delphes output objects (specified by `delphes???OutArray` variables in PythiaDelphes_config.py) are written out to FCC-EDM collections & associated collections (one-to-one relations)



- **`delphes???OutArrays`** (may be modified by user!):

```
delphesMuonsOutArray    ="MuonIsolation/muons"
delphesElectronsOutArray="ElectronIsolation/electrons"
delphesChargedOutArray  ="ChargedHadronMomentumSmearing/chargedHadrons"
delphesNeutralOutArray  ="Hcal/eflowNeutralHadrons"
delphesPhotonsOutArray  ="PhotonIsolation/photons"
delphesJetsOutArray     ="JetEnergyScale/jets"
delphesMETsOutArray     ="MissingET/momentum"
delphesSHTsOutArray     ="ScalarHT/energy"
```

FCC-EDM Output

- **Collections:**
  - `fcc::MCParticleCollection` --> generated particles `[genParticles]`
  - `fcc::GenVertexCollection` --> generated vertices `[genVertices]`
  - `fcc::GenJetCollection` --> generated jets `[genJets]`
  - `fcc::ParticleCollection` --> reconstructed muons `[muons]`, electrons `[electrons]`, charged particles `[charged]`, neutral particles `[neutral]`, photons `[photons]` and jet constituents `[jetParts]`
  - `fcc::JetCollection` --> reconstructed jets `[jets]`
  - `fcc::METCollection` --> reconstructed missing Et `[met]`
  - `fcc::IntTagCollection` --> flavour tag of generated `[genJetsFlavor]` or reconstructed jets `[jetsFlavor]`, i.e. PDG of leading constituent
  - `fcc::TagCollection` --> reconstructed tags - b-tags, tau-tags for jets `[bTags, tauTags]`
  - `fcc::TagCollection` --> reconstructed isolation tag info for electrons, muons and photons `[muonITags, electronITags, photonITags]`

- **Relations:**
  - `fcc::ParticleMCParticleAssociationCollection` --> relations of reconstructed object to MC particle for muons `[muonsToMC]`, electrons `[electronsToMC]`, charged particles `[chargedToMC]`, neutral particles `[neutralToMC]` and photons `[photonsToMC]`
  - `fcc::ParticleTagAssociationCollection` --> relations of reconstructed particles: muons, electrons and photons to their isolation tag info `[muonsToITags, electronsToITags, photonsToITags]`
  - `fcc::GenJetParticleAssociationCollection` --> relations of generated jet to MC particle `[genJetsToMC]`
  - `fcc::GenJetIntTagAssociationCollection` --> relations of generated jets to the flavour (PDG of the leading constituent) `[genJetsToFlavor]`
  - `fcc::JetParticleAssociationCollection` --> relations of jet to particle constituents `[jetsToParts]`
  - `fcc::JetIntTagAssociationCollection` --> relations of jets to the flavour (PDG of the leading constituent) `[jetsToFlavor]`
  - `fcc::JetTagAssociationCollection` --> relations of jets to reconstructed tags - b-tag, tau-tag `[jetsToBTags, jetsToTauTags]`

Some Practical Info

- Open the `PythiaDelphes_config.py` configuration file:

```
cd $FCCSW
vim Sim/SimDelphesInterface/options/PythiaDelphes_config.py
```

  - Change the verbosity level from `DEBUG` to `INFO`
  - Change the number of events to be studied: `nEvents` from 10 → 10k or 100k events
  - For simplicity, change the output from FCC event-data-model to simple Delphes output as a starting point for easy access to all saved info (after the quick check on data follow use the official FCC EDM output instead)

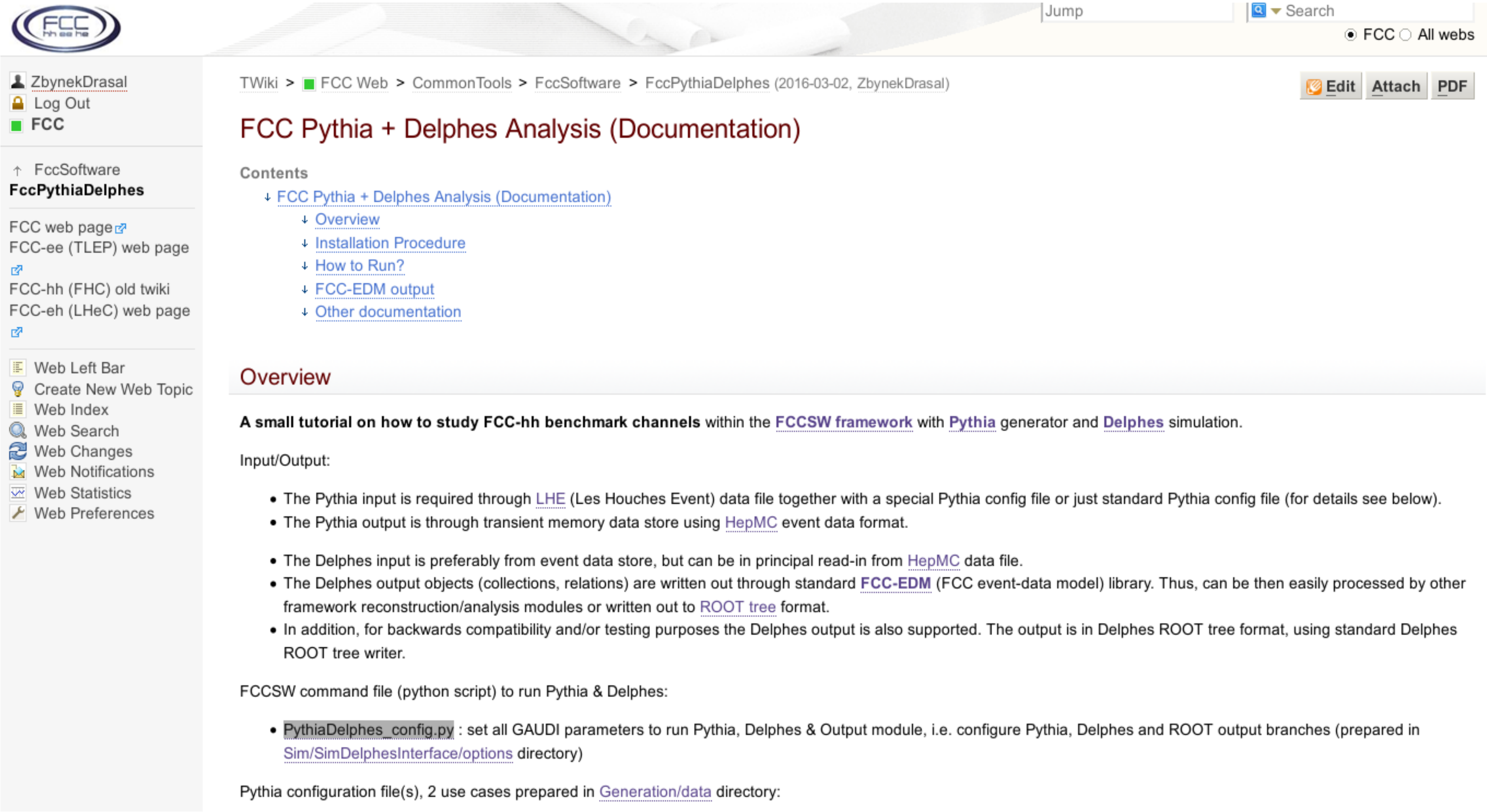
    → But, don't forget to read-in the `libDelphes.so` library (dictionary) in that case (ROOT needs to knows about defined Delphes objects)

```
root
.L /afs/cern.ch/exp/fcc/sw/0.7/Delphes/3.3.2/x86_64-slc6-gcc49opt/lib/libDelphes.so
TFile *_file0 = Tfile::Open("DelphesOutput.root")
TBrowser t
```

  - Follow tutorial on ROOT trees to find out how to process this data:
    https://root.cern.ch/root/htmldoc/guides/users-guide/ROOTUsersGuide.html#trees

Need more Information?

- For additional information follow the HowTo? at FCC Twiki page:
  - https://twiki.cern.ch/twiki/bin/view/FCC/FccPythiaDelphes



FCC Pythia + Delphes Analysis (Documentation)

Contents

Overview

A small tutorial on how to study FCC-hh benchmark channels within the **FCCSW framework** with **Pythia** generator and **Delphes** simulation.

Input/Output:

- The Pythia input is required through LHE (Les Houches Event) data file together with a special Pythia config file or just standard Pythia config file (for details see below).
- The Pythia output is through transient memory data store using HepMC event data format.

- The Delphes input is preferably from event data store, but can be in principal read-in from HepMC data file.
- The Delphes output objects (collections, relations) are written out through standard **FCC-EDM** (FCC event-data model) library. Thus, can be then easily processed by other framework reconstruction/analysis modules or written out to ROOT tree format.
- In addition, for backwards compatibility and/or testing purposes the Delphes output is also supported. The output is in Delphes ROOT tree format, using standard Delphes ROOT tree writer.

FCCSW command file (python script) to run Pythia & Delphes:

- PythiaDelphes_config.py : set all GAUDI parameters to run Pythia, Delphes & Output module, i.e. configure Pythia, Delphes and ROOT output branches (prepared in Sim/SimDelphesInterface/options directory)

Pythia configuration file(s), 2 use cases prepared in Generation/data directory: