JAS4pp – Java Analysis Framework for Particle Physics

S.Chekanov (ANL)

Snowmass21 Computational Frontier Workshop, CompF5: End user analysis

August 10-11, 2020

LOI: https://www.snowmass21.org/docs/files/summaries/CompF/SNOWMASS21-CompF5-001.pdf

Jas4pp - a Data-Analysis Framework for Physics and Detector Studies

S.V. Chekanov[a], N. A. Graf[b], G. Gavalian[c]

[a] *HEP Division, Argonne National Laboratory, 9700 S. Cass Avenue, Argonne, IL 60439, USA.*
[b] *SLAC National Accelerator Laboratory, Menlo Park, CA 94025 USA.*
[c] *Thomas Jefferson National Accelerator Facility, Newport News, Virginia 23606, USA.*

Physics and detector studies with JAS4pp

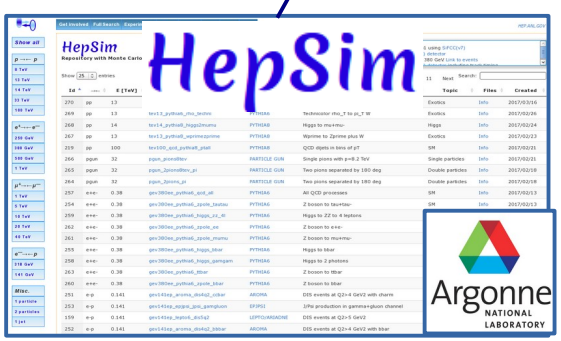
- JAS4pp is a modified version of Jas3 (SLAC) for physics analysis in particle physics ("4pp") for future experiments. Developed at HEP/ANL with Lesser GPL license
- Many new features compared to JAS3 + improved speed and memory usage
- Integrated with the **HepSim** repository
- Wiki documentation, but no comprehensive description

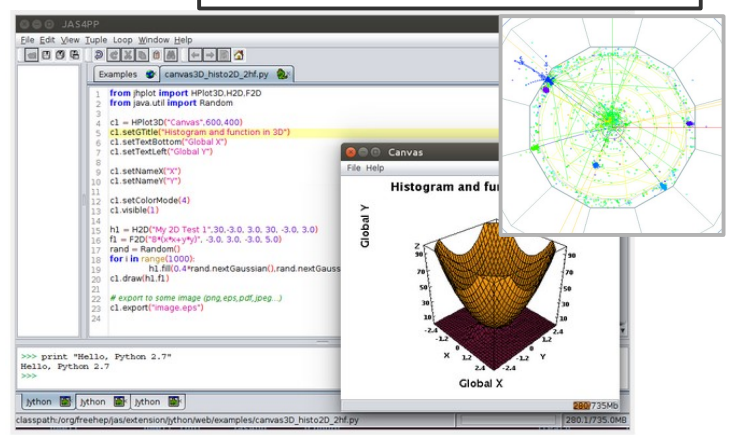
ongoing optimization

SLIC 5.1 (Geant4, version 10.3p1) → **LCSIM track reconstruction** → **SlicPandora and Pandora PFA** → **End-User Analysis: Jas4pp**

ProMC → SLCIO

Compact geometry description (XML)





http://atlaswww.hep.anl.gov/hepsim/

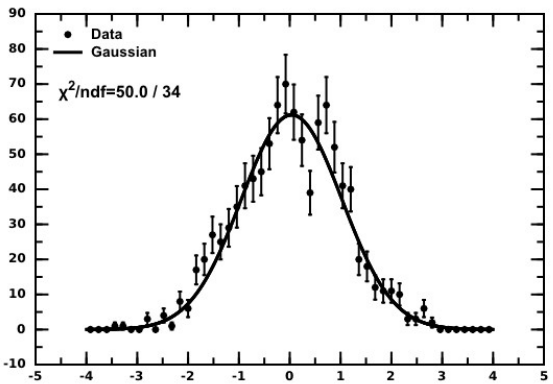
Main features of JAS4pp

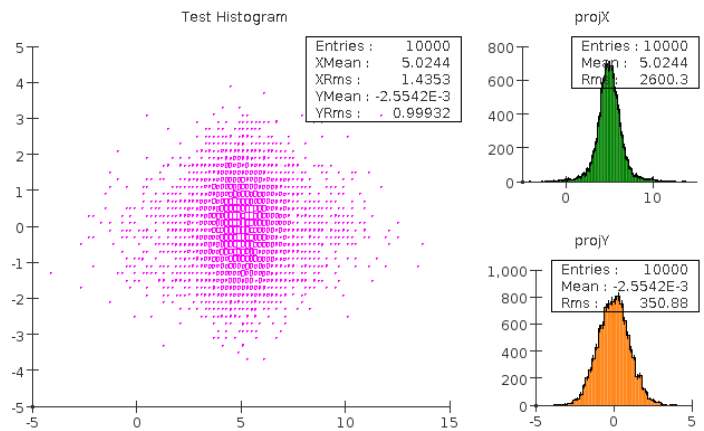
Platform-independent light-weight Java program (~90 MB) running on JVM :

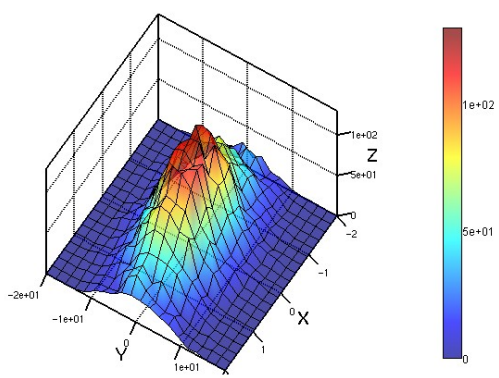
- Combined strengths of 2 most popular languages: Python and Java
  - Analysis code can be written in Python 2.7.2 (Jython) or Java (8+). Optional: Groovy, Ruby, etc.
- Includes extensive histogramming in 2D/3D, graphs, data-manipulation methods, I/O, mathematical packages from Apache Common Math
  - Example:~100 methods for 1D histograms → similar to TH1D in ROOT
- Support for ProMC, ProIO (Google protocol buffers) and LCIO files with reconstructed data with various object collections (hits, tracks, PFO)
- Much more than ROOT: It is software suite with HEP analysis tools:
  - Jet algorithms for e+e-, pp (including fast anti-KT), interactive fits, event shape algorithms, limit settings, tools for analysis of detector-level information (tracks, hits, clusters)
  - Data browser for reconstructed events and universal event display based on Wired4

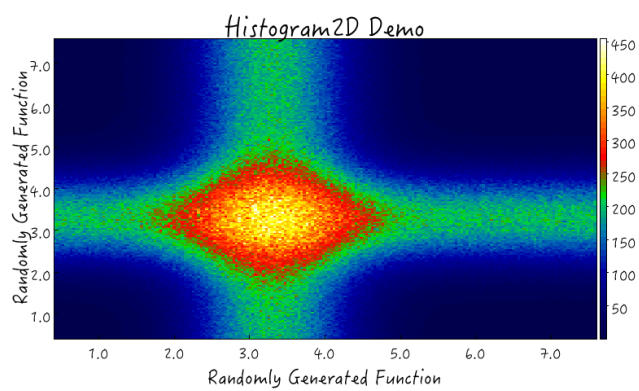
Graphics

- Similar to ROOT: Histograms in 2D/3D, all types of plots
- Supports Vector Graphics outputs (SVG, EPS, PDF)
- Includes packages from JAS3, JAIDA, LCSIM (SLAC), GROOT (JLab), DataMelt









Sustainability of JAS4pp & Java

- Java is the most popular enterprise language
- Stable, backward compatible, sustainable. Applications do not require maintenance for a long time. OpenJDK is open-source implementation
- *Example of backward compatibility:*
  - JAS 0.4 ftp://ftp.slac.stanford.edu/software/jas/JAS3/v0.4/ from 2003 runs without modification on modern computers with OpenJDK14 (compiles too!)
  - Big contrast to C/C++ (especially if graphics is involved)
- Java-based end-user environments are the optimal solution for experiments to be built in 20-30 years due to very low maintenance of software

Previous usage of JAS4pp

Used since 2017 for physics, future detector studies and TileCal/ATLAS work

- FCC-hh detector performance studies (~6 detector versions) for 100 TeV
- CEPC studies (Si-tracking version)
- EIC physics and detector studies (TOPSiDE detector concept)
- SiD detector is still supported
  - JAS3 used for SiD detector design (the USA concept of an ILC detector)
- Analysis support for HepSim repository (truth-level and LCIO files)

Results included in a dozen of publications, CDR reports and talks

One of the data-analysis frameworks created
in USA laboratories
(but with no financial support at present)

Some features and benchmarks

- Fast deployment on any platform (Linux/Windows/MacOS)
  - time needed to deploy the entire data-analysis environment is comparable with the time to download it! → takes ~1 min (90 MB)
- Scripts can be executed in a batch mode and in full-featured IDE
- Can be used with Java, Groovy, Ruby (required additional jar files)
- Execution speed using Jython is comparable with CPython / PyROOT
  - more memory usage, but better memory management

Built-in universal event display based on Wired:

- Supports 4 different detectors, SiFCC, SiD, CEPC-SiD, TOPSide (EIC) with about 10 different versions
- Detector geometry dynamically downloaded during opening LCIO files (no actions from the users required)
- Faster rendering compared to the original JAS3+LCSIM

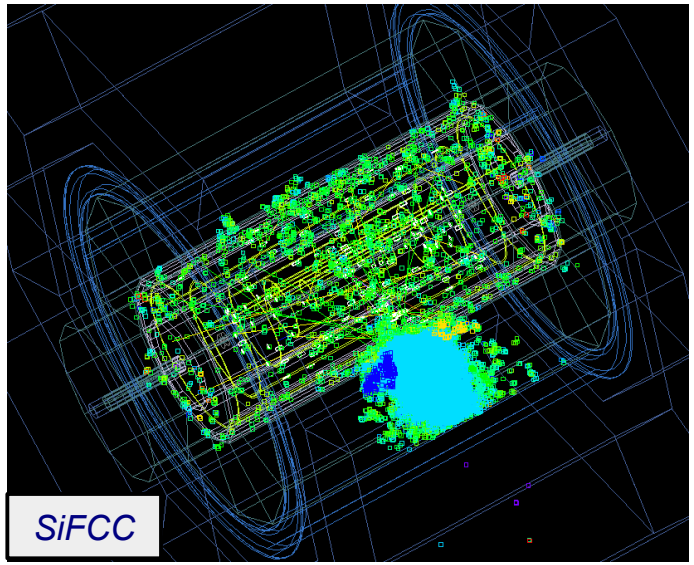
JAS4pp (Wired) event display:
Response to single particles: 8 TeV pions

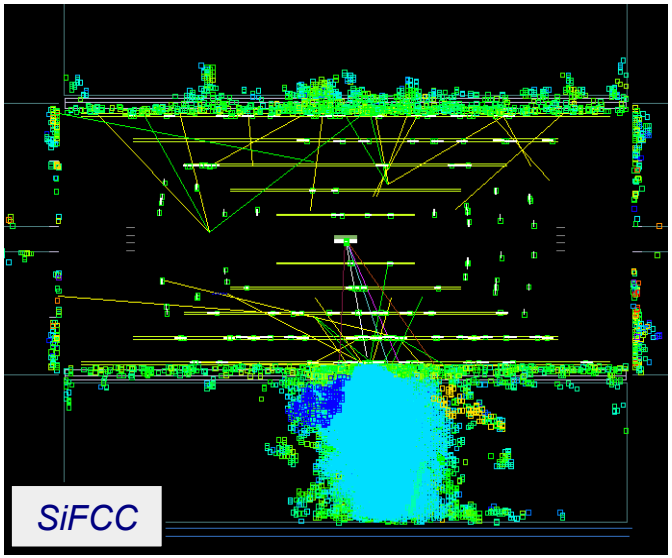
True momentum of π+ : 8.16 TeV

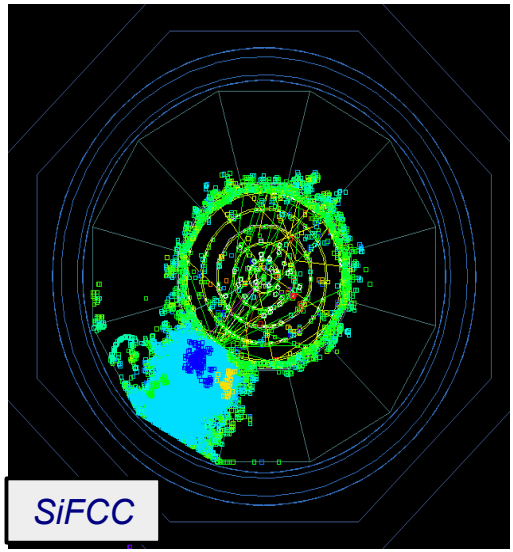
Presented at various FCC-hh meetings between 2016-2018 dedicated to calorimeter studies

After SiFCC reconstruction (>1.5 M HCAL cells):

- ~30000 calorimeter hits, ~500 SiTracker hits
- 1 reconstructed PFA (pi+) P=8.97 TeV
- 1 reconstructed CaloCluster at P=8.40 TeV







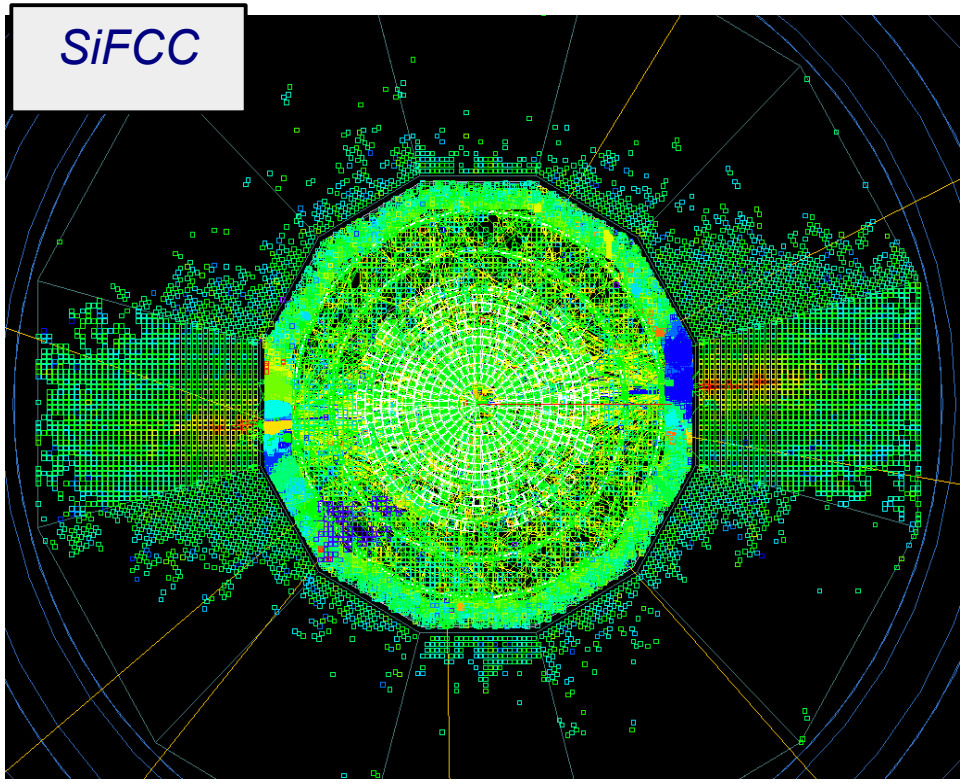
Based on **HepSim**: http://atlaswww.hep.anl.gov/hepsim/info.php?item=201

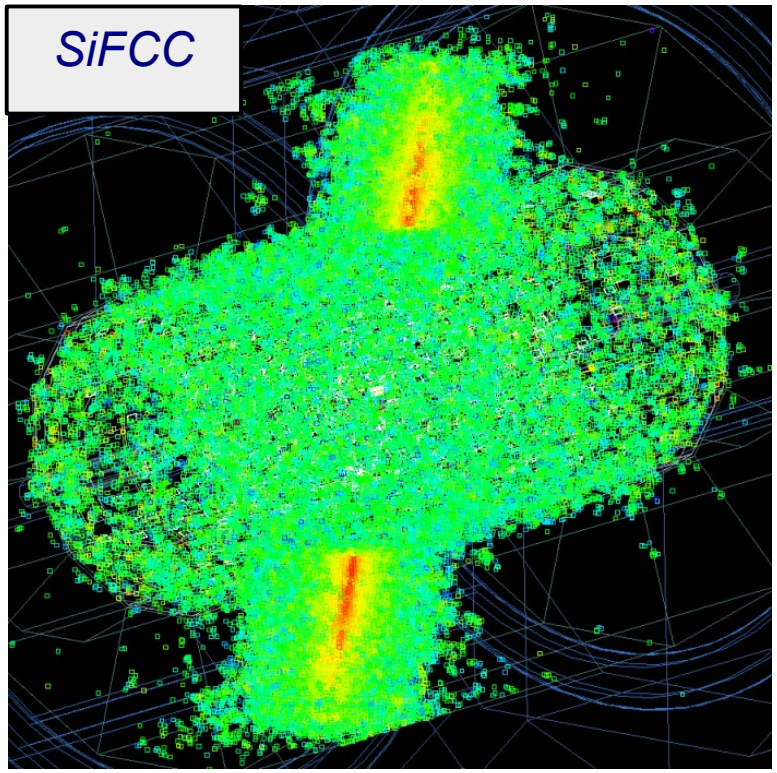
Event display of Z' (40 TeV) → $q\bar{q}$

First realistic Geant4 simulation of ~20 TeV jets (FCC week, Apr 2016)

High-granularity HCAL, 10k hits in ECAL, 46k hits in HCAL, 12k/1k hits in the outer/inner tracker

Images from FCC news featured article





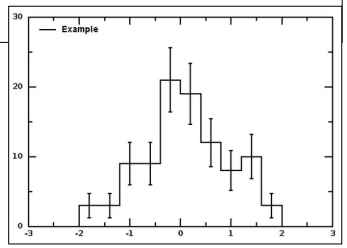
Advanced features for detector designs:

- Dynamic ruler to measure and debug detector volumes
- 3D rendering etc.

Basic example: A histogram with random numbers (Python syntax)

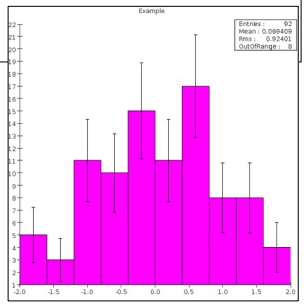
DataMelt style

```
from java.util import *
from jhplot  import *
c1 = HPlot()
c1.visible()
c1.setAutoRange()
h1 = H1D("Example",10,-2,2)
rand = Random()
for i in xrange(100):
   h1.fill(rand.nextGaussian())
c1.draw(h1)
c1.export("histo.eps")
```



JAIDA style using factories

```
from hep.aida import *
from java.util import *
fac = IAnalysisFactory.create()
hf = fac.createHistogramFactory(None)
h1 = hf.createHistogram1D("Example",10,-2,2)
r = Random()
for i in xrange(100):
    h1.fill(r.nextGaussian())
c1 = fac.createPlotterFactory().create("plot")
c1.show()
c1.createRegions(1,1)
c1.region(0).plot(h1)
```



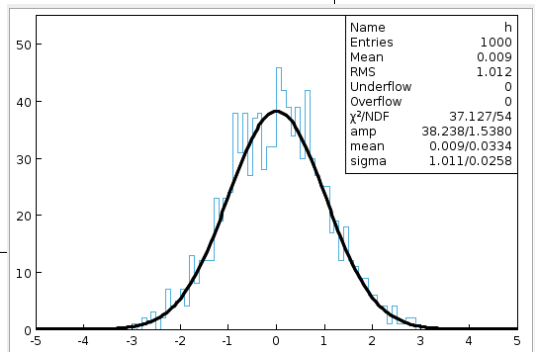
Jas4pp supports DataMelt and JAIDA styles
DataMelt API leads to shorter programs

JAS4pp supports GROOT syntax (similar to ROOT)

```
from org.jlab.groot.data import *
from org.jlab.groot.ui import *
from org.jlab.groot.math import *
from org.jlab.groot.fitter import *
from java.util import Random

c = TCanvas('c',500,500)
h = H1F('h',100,-5.0,5.0)
r = Random()
for i in xrange(1000):
    h.fill(r.nextGaussian())
f = F1D('f','[amp]*gaus(x,[mean],[sigma])',-5,5)
f.setParameter(0,1000)
f.setParameter(1,0.0)
f.setParameter(2,1.0)
DataFitter.fit(f,h,'Q')
h.setLineColor(4)
f.setLineWidth(3)
f.setLineStyle(4)
h.setOptStat('111111111111')
c.draw(h)
c.draw(f,'same')
```

Unlike ROOT, it imports Java packages

- GROOT (JLAB) syntax very similar to PyROOT
- Switching from the Python language to Groovy leads to **a factor 10 faster** execution than equivalent PyROOT codes (based on CPython)
- See Benchmark results for large loops



JAS4PP

File Edit View Tuple Loop Window Help

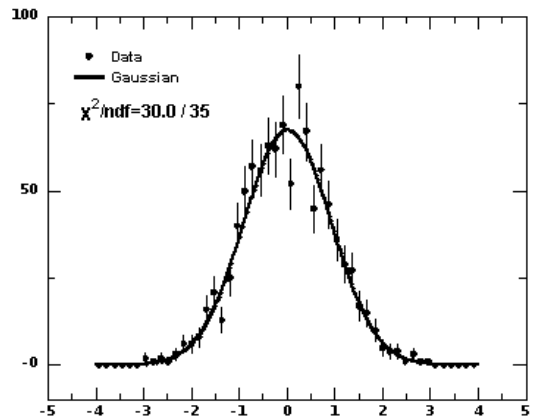
tree-0

Welcome | Untitled

```
from jhplot  import *
from java.util import Random

f=HFitter()
print f.getFuncCatalog()
f.setFunc('g')
f.setPar('amplitude',50)

h1 = H1D('Data',50, -4, 4)
h1.setPenWidthErr(2)
h1.setStyle("p")
h1.setSymbol(4)
h1.setDrawLine(0)

r = Random()
for i in range(1000):
    h1.fill(r.nextGaussian())

c1 = HPlot('Canvas')
c1.visible()
c1.setAutoRange()
c1.draw(h1)

f.setRange(-4,4)
f.fit(h1)
ff=f.getFittedFunc()

# get fitted results
r=f.getResult()
Pars   = r.fittedParameters()
Errors = r.errors()
Names  = r.fittedParameterNames()
print "Fit results:"
for i in range(ff.numberOfParameters()):
  print Names[i]+" : "+str(Pars[i])+" +- "+str(Errors[i])


mess='&chi;^{2}/ndf='+str(round(r.quality()*r.ndf()))
mess=mess+" / "+str(r.ndf())
lab=HLabel(mess, 0.12, 0.69, 'NDC')
c1.add(lab)
```

Canvas

File Help



131/258Mb

```
mean : 0.0162384984253 +- 0.0294495114907
sigma : 0.91560984693 +- 0.0215383653962
Quality= 0.858002125011  NDF= 35
>>>
```

131.8/258.5MB

Editor with syntax highlighting

interactive console

Where to find more examples

- The Welcome screen of JAS4pp navigates to ~20 examples executed directly in the editor
- ~140 example snippets using truth-level MC from HepSim
  - https://atlaswww.hep.anl.gov/hepsim/macrolist.php
  - https://atlaswww.hep.anl.gov/hepsim/macrolist_lcio.php
- About ~300 examples from DataMelt
  - https://datamelt.org/code/
- GROOT also has a repository with examples
  - https://github.com/gavalian/groot

Snowmass21 contribution

- Paper with Jas4pp description on Overleaf (~7 pages)
- Structure:
  - History
  - Usage
  - Structure
  - Examples implemented in the Python/Jython language:
    - Analysis of truth-level events
    - Analysis of reconstructed tracks (+histogram)
    - Analysis of calorimeter clusters and reconstruction of antiKT jets

- Will be finished by the end of 2020
- To be submitted to Comp.Phys.Comm.
- “Open” author list. Interested? Join!

Thanks!

Backup

1-page example code:

Reading Monte Carlo events over the network

Stream ProMC files (based on Google Protocol Buffers) with Monte Carlo (13 TeV pp collision events) from HepSim repository, reconstruct anti-KT jets and print Eta and Phi

No download of files to a local computer is needed

```
from proto import FileMC
from jhplot.utils import FileList
from hepsim import HepSim
from java.util import ArrayList
from hephysics.hepsim import PromcUtil
from hephysics.jet import JetN2

ktjet=JetN2(0.4,"antikt",20)
sites=HepSim.urlRedirector( "tev13pp_pythia8_gkk2radion2gg" )
url=sites[0]+"/"
flist=HepSim.getList(url)
if len(flist)==0: print "Error: No input file!"; sys.exit(0)

nev=0
Nfiles=len(flist)
Total=len(flist)
for m in range(Nfiles):              # loop over all files in this list
   file=FileMC(url+flist[m])         # get input file
   header = file.getHeader()
   un=float(header.getMomentumUnit()) # conversion units
   lunit=float(header.getLengthUnit())
   print "Read=",flist[m]
   for i in range(file.size()):
     nev=nev+1
     if (nev%100==0):
         print "Event=",nev
     eve = file.read(i)
     pa = eve.getParticles()    # particle information
     ve = eve.getEvent()        # event information
     particles=PromcUtil.getParticleDList(file.getHeader(), pa, 1, 0, 1000);
     ktjet.buildJets(particles);
     jets=ktjet.getJetsSorted()
     finaljets=[]
     for jet in jets:
           print jet.getPhi(),jet.getRapidity()
   stat = file.getStat()
   xcross=stat.getCrossSectionAccumulated()
   erro=stat.getCrossSectionErrorAccumulated();
   file.close()
```

1-page example code:

Read a local LCIO file with truth-level objects from LCIO

Reading Monte Carlo e+e- events and plot Pz of stable particles. Export 1D histogram to a PDF file

```
from hep.lcio.implementation.io import LCFactory
from jhplot import H1D,HPlot  # import graphics

files=["gev250ee_pythia6_zpole_ee.slcio"]
factory = LCFactory.getInstance()
nEvent=0
h1=H1D("Mass",70,50,120)
for f in files:
  print "Open file=",f
  reader = factory.createLCReader()
  reader.open(f)
  while(1):
    evt=reader.readNextEvent()
    if (evt == None): break
    nEvent=nEvent+1
    print " file event: ",evt.getEventNumber(), " run=",evt.getRunNumber()
    col = evt.getCollection("MCParticle")
    nMc=col.getNumberOfElements()
    for i in range(nMc): # loop over all particles
      par=col.getElementAt(i)
      if(par.getGeneratorStatus() == 1 and par.getCharge() !=0):
          vertex = par.getVertex();
          pdg=par.getPDG()
          momentum = par.getMomentum()
          ee=par.getEnergy()
          mass=par.getMass()
          px,py,pz=momentum[0],momentum[1], momentum[2]
          h1.fill(pz)
    del col,evt
  reader.close() # close the file
c1=HPlot()
c1.visible()
c1.setAutoRange()
c1.setMarginLeft(100)
c1.setNameX("Pz [GeV]")
c1.setNameY("Events")
c1.draw(h1)
c1.export("mc_truth.pdf")
```

1-page example code

Tracking from LCIO

Read all LCIO files from a local directory, extract tracking information and fill a LorentzVector object.

```
from hep.lcio.implementation.io import LCFactory
from org.lcsim.event.base import BaseTrackState
from hephysics.particle import LParticle
from math import sqrt
import glob
files=glob.glob("files/*.slcio")
factory = LCFactory.getInstance()
nEvent=0
for f in files:
  print "Open file=",f
  reader = factory.createLCReader()
  reader.open(f)
  while(1):
    evt=reader.readNextEvent()
    if (evt == None): break
    nEvent=nEvent+1
    # print " file event: ",evt.getEventNumber(), " run=",evt.getRunNumber()
    if (nEvent%100==0): print "# Event: ",nEvent
    strVec = evt.getCollectionNames()
    if nEvent == 1:
          for col in  strVec:  print col
    col = evt.getCollection("Tracks")
    ntracks = col.getNumberOfElements()
    Bfield=5.0 # B-field of sidloi3 detector
    particles=[]
    for i in range(ntracks):
        track=col.getElementAt(i)
        trk=BaseTrackState()
        trk.setZ0(track.getZ0())
        trk.setPhi(track.getPhi())
        trk.setOmega(track.getOmega())
        trk.setD0(track.getD0())
        trk.setTanLambda(track.getTanLambda())
        charge=1
        if (track.getOmega()<0): charge=-1
        mom=trk.computeMomentum(Bfield) # B filed in Z
        px,py,pz=mom[0],mom[1],mom[2]
        ee=sqrt(px*px+py*py+pz*pz)
        p=LParticle("track",px,py,pz,ee,0)
        print(p)
    del col,evt
  reader.close() # close the file
```

1-page example code

Create antiKT jets from PFO objects

Read all LCIO files from a local directory, extract PFO information and reconstruct antiKT jets using the fast implementation.

```
from java.util import ArrayList
from hep.lcio.implementation.io import LCFactory
from jhplot import H1D
from hephysics.jet import ParticleD
from hephysics.jet import JetN2
import glob # make list of files..
files=glob.glob("files/*.slcio")
factory = LCFactory.getInstance()
h1=H1D("jet pt",100,0,20)
ktjet=JetN2(0.5,"antikt",20)    # antiKT with R=0.5, E-mode, anti-KT,min pT=20
print ktjet.info()              # print its settings
nEvent=0
for f in files:
  print "Open file=",f
  reader = factory.createLCReader()
  reader.open(f)
  while(1):
    evt=reader.readNextEvent()
    if (evt == None): break
    nEvent=nEvent+1
    if (nEvent%50==0): print "# Event: ",nEvent
    strVec = evt.getCollectionNames()
    if nEvent == 1:
          for col in  strVec:
                      print col
    col = evt.getCollection("PandoraPFOCollection")
    nPFA = col.getNumberOfElements()
    alljets=[] # make a new list with jets
    particles=ArrayList() # list of particles
    for i in range(nPFA):
        pa=col.getElementAt(i)
        p4=pa.getMomentum()
        ee=pa.getEnergy()
        p=ParticleD(p4[0],p4[1],p4[2],ee);
        particles.add(p) # add particle to the list
    ktjet.buildJets(particles)
    jets=ktjet.getJetsSorted()      # get a list with sorted jets
    if (len(jets)>0):
              print "pT of a leading jet =",jets[0].perp()," GeV”
    del col,evt
  reader.close() # close the file
```