

# Analysis with Geant4 and AIDA

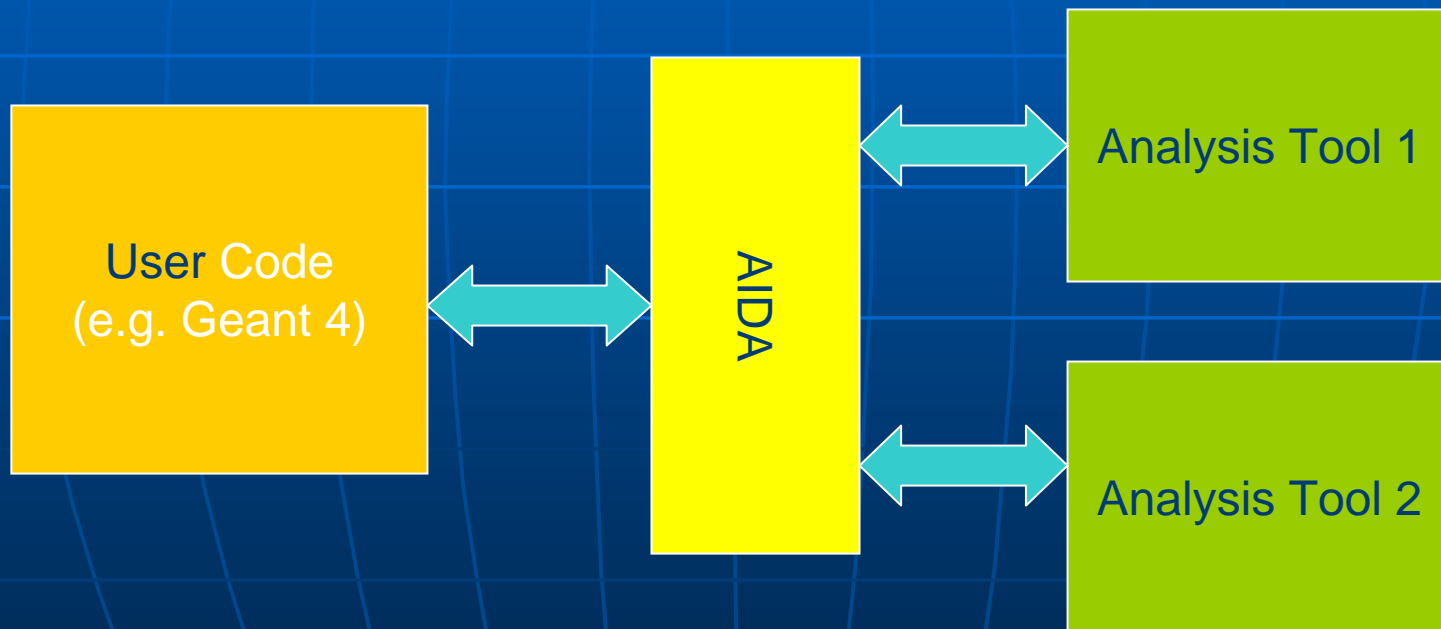
Tony Johnson

Fermilab Geant4 Workshop

October 2003

# What is AIDA?

- AIDA = Abstract Interfaces for Data Analysis



# Why AIDA?

- Goal
  - Provide the user with a powerful set of interfaces which can be used regardless of which underlying analysis tool they are using.
- Advantages
  - The user will only need to learn one set of interfaces even if they use more than one tool.
  - By pooling the experiences of several different development teams we will be able to develop a more complete solution to the data analysis problem.
  - Different analysis tools built using AIDA will be able to interoperate, for example by exchanging objects in a common storage format (initially XML).
- Currently two (.5) versions of the AIDA interfaces exist,
  - one for Java
  - one for C++
    - The two interfaces are as identical as the underlying languages will permit.
  - Also “unofficial” python interface

# Who is AIDA?

- Past and present contributors:
  - Guy Barrand, Pavel Binko, Grzegorz Chwajol, Mark Donszelmann, Wolfgang Hoschek, Tony Johnson, Emmanuel Medernach, Dino Ferrero Merlino, Lorenzo Moneta, Jakub Moscicki, Ioannis Papadopoulos, Andreas Pfeiffer, Max Sang, Victor Serbo, Max Turri
  - CERN, SLAC, LAL, others...

# Aida Interfaces: IHistogram

- Histograms

- fixed and variable width binning
- filling
- access to overall statistics (entries, mean, rms)
- bin information (center, entries, height, error)
- arithmetic (add, multiply, divide)
- projections
- slices

# Aida Interfaces: ICloud

- Histograms
- Clouds

- unbinned histograms
- scatter plots
- auto-conversion to histograms

# Aida Interfaces: IProfile

- Histograms
- Clouds
- Profiles

- fill 1D and 2D profile plots
- access to bin statistics (height, mean, rms)
- access to overall statistics (entries, mean rms)
- scaling

# Aida Interfaces: IDataPointSet

- Histograms
- Clouds
- Profiles
- Data Point Sets
  - sets of n-dimensional points with errors
  - add, remove, get points
  - scaling of values and errors
  - full mathematical arithmetic (+-\*/)
  - simple error propagation



# Aida Interfaces: ITuple

- Histograms
- Clouds
- Profiles
- Data Point Sets
- N-tuples
  - Evaluators
  - Filters

- fill and retrieve data
- support folder-like structure
- projections to histograms, clouds and profiles of "evaluated" quantities with "filtering"
- chaining, merging

# Aida Interfaces: IPlotter

- Histograms
- Clouds
- Profiles
- Data Point Sets
- N-tuples
  - Evaluators
  - Filters
- Plotting

- create plot area, regions
- control styles (title, text, markers, lines etc.)

# Aida Interfaces: IFitter

- Histograms
- Clouds
- Profiles
- Data Point Sets
- N-tuples
  - Evaluators
  - Filters
- Plotting
- Fitter

- fitting to all the data storage types  
`IFitResult result = fitter.fit(data, function)`
- change fit method ( $\chi^2$ , max. Likelihood, etc)
- change optimizer (Minuit, etc)
- control parameters (bounds, fix, step, etc.)
- set constraints
- create scans and contours
- use function's analytical gradient

# Aida Interfaces: IFunction

- Histograms
- Clouds
- Profiles
- Data Point Sets
- N-tuples
  - Evaluators
  - Filters
- Plotting
- Fitter
- Functions

- create scripted or built-in functions
- access/change parameter's values
- evaluate function and its gradient
- support of PDFs (normalized functions over a range)

# Aida Interfaces: ITree

- Histograms
- Clouds
- Profiles
- Data Point Sets
- N-tuples
  - Evaluators
  - Filters
- Plotting
- Fitter
- Functions
- IO

- storage for analysis objects
- XML interchange format standard (.aida files)
- no APIs; storage is application specific
- unix-like : ls, cp, mv, cd ....

# Example AIDA program (Java)

```
import hep.aida.*;
import java.util.Random;

public class Histogram
{
    public static void main(String[] argv)
    {
        IAnalysisFactory af = IAnalysisFactory.create();
        IHistogramFactory hf = af.createHistogramFactory(af.createTreeFactory().create());

        IHistogram1D h1d = hf.create1D("test 1d", 50, -3, 3);
        IHistogram2D h2d = hf.create2D("test 2d", 50, -3, 3, 50, -3, 3);

        Random r = new Random();
        for (int i=0; i<10000; i++)
        {
            h1d.fill(r.nextGaussian());
            h2d.fill(r.nextGaussian(), r.nextGaussian());
        }

        IPlotter plotter = af.createPlotterFactory().create("Plot");
        plotter.createRegions(1, 2, 0);
        plotter.plot(h1d);
        plotter.next();
        plotter.plot(h2d);
        plotter.show();
    }
}
```

# What AIDA Compliant Tools exist?

- Current Version of AIDA is 3.2.1

- Tools

- C++

- Open Scientist 11.0

<http://www.lal.in2p3.fr/OpenScientist/>

- Anaphe (5.0) <http://anaphe.web.cern.ch/anaphe/>

- Now moved to LCG-PI project

- Includes AIDA->Root

- AIDA-JNI (3.0.4) <http://java.freehep.org/aidajni>

- Java

- JAIDA (3.2.0) <http://java.freehep.org/jaida>

- JAS3 (0.7.3) <http://jas.freehep.org/jas3>

On  
The  
CD

- Python

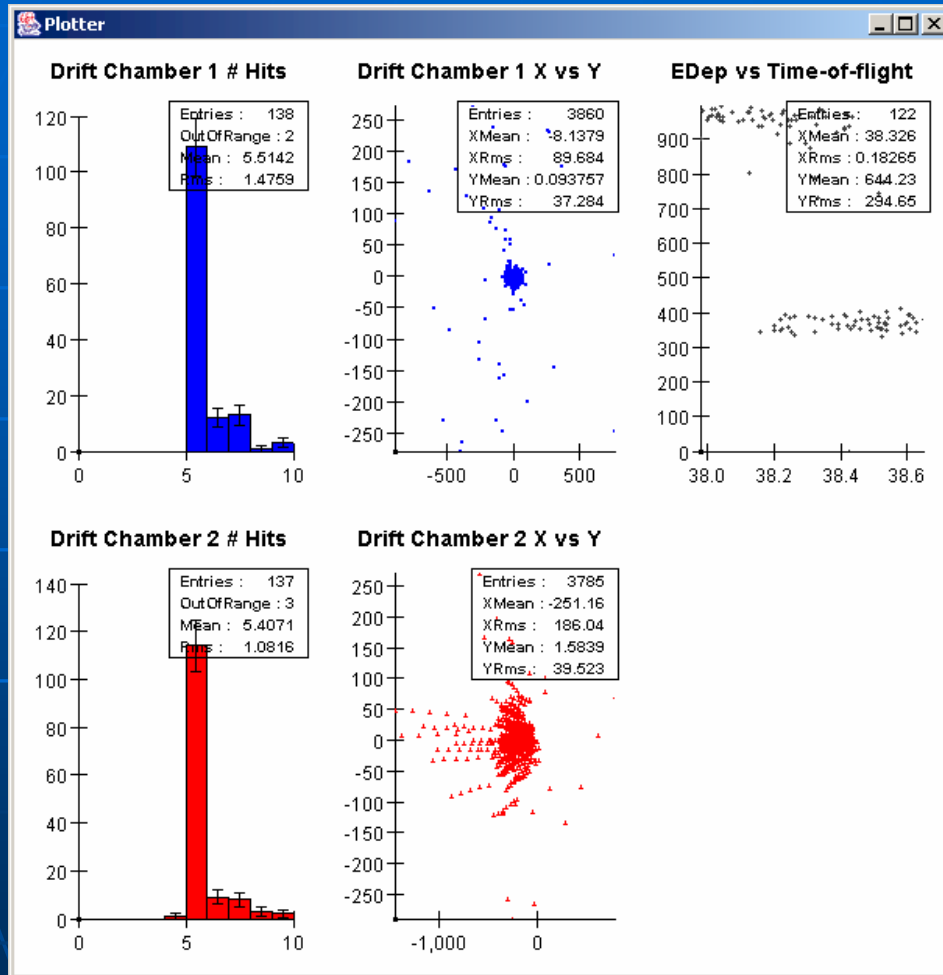
- PAIDA (3.2.1b1) <http://paida.sourceforge.net>

# AIDA with Geant4

- AIDA allows Geant4 examples to illustrate analysis techniques without choosing a single analysis tool
- The T01 example (and several others) has been set up to use AIDA for analysis
- By default links without AIDA implementation
- Switching to a real AIDA implementation is easy, just set environment variables and relink
  - `source setup-analysis # (or source setup-analysis.csh)`
  - `make`
  - `../../../../bin/$G4SYSTEM/A01app novis.mac`



# Simple GUI build into JAIDA



Plots update in real time.

A01 also creates a A01.aida file which can be analyzed offline.

# Adding to A01 Analysis

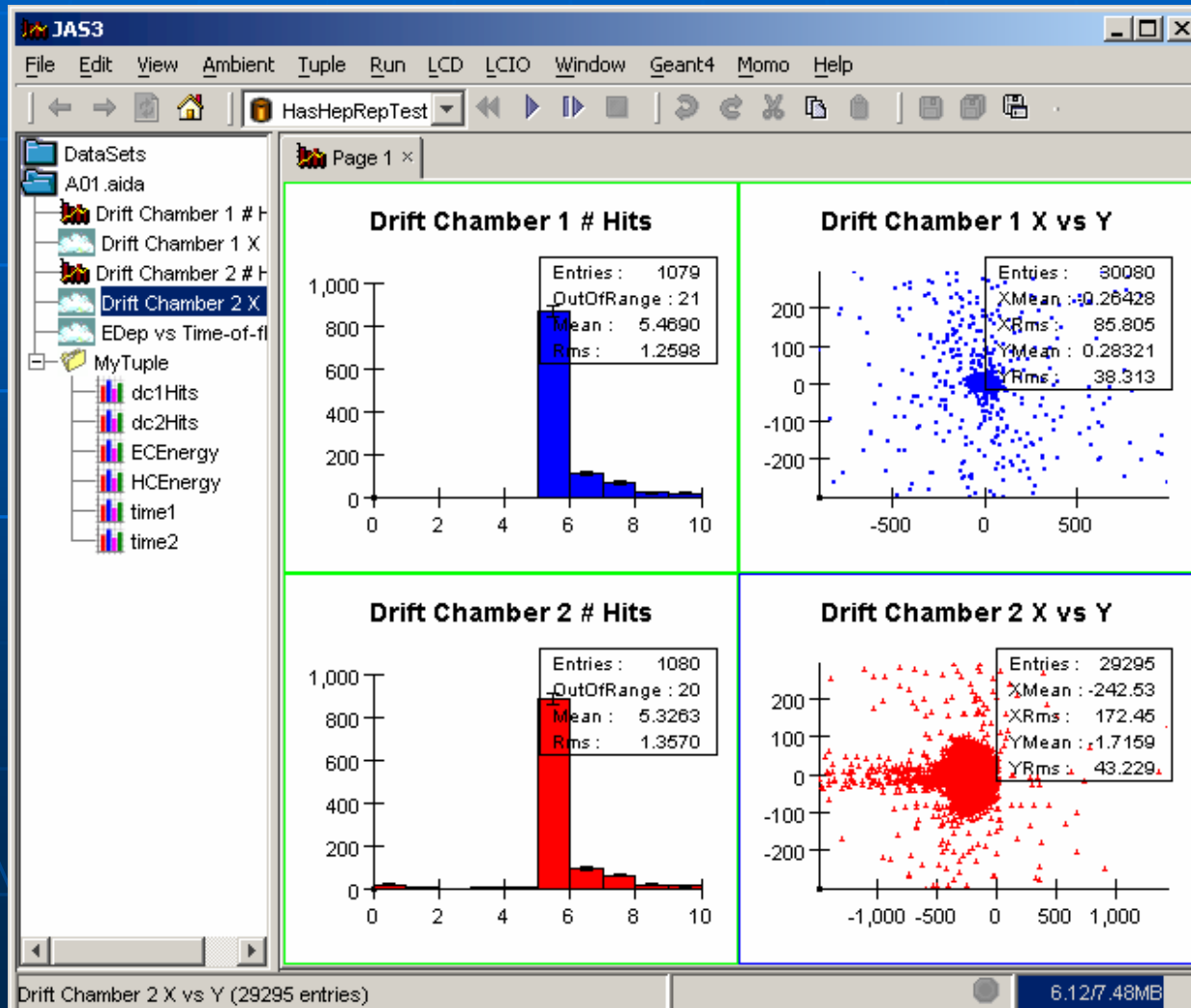
- **T01AnalysisManager** - This class is a simple wrapper around the AIDA factories used to create Histograms, Plotters and Tuples. You should not need to modify this class, but you may want to take a look at it to understand what it is doing. The useful methods it defines are:
  - *static T01AnalysisManager\* getInstance()* - Get the (singleton) instance of T01AnalysisManager
  - *IHistogramFactory\* getHistogramFactory()* - Used for creating histograms, clouds
  - *ITupleFactory\* getTupleFactory()* - Used for create n-tuples
  - *IPlotter\* createPlotter()* - Create a Plotter (used for displaying histograms and clouds)
- **T01EventAction** - This is where the user defined analysis is performed. After reading the [AIDA Users Guide](#) the code here should be fairly self explanatory. A good exercise would be to add you own histograms and display them using the plotter, or add your own columns to the Tuple.

# Offline Analysis with AIDA

- When run with analysis turned on the A01 example also generates a .aida file.
  - This is a (compressed) XML file containing all the histograms and tuples created by the analysis.
  - The .aida XML format is part of the AIDA standard and can be analyzed by any AIDA compliant tool.
  - One such tool is JAS3 – included on CD.

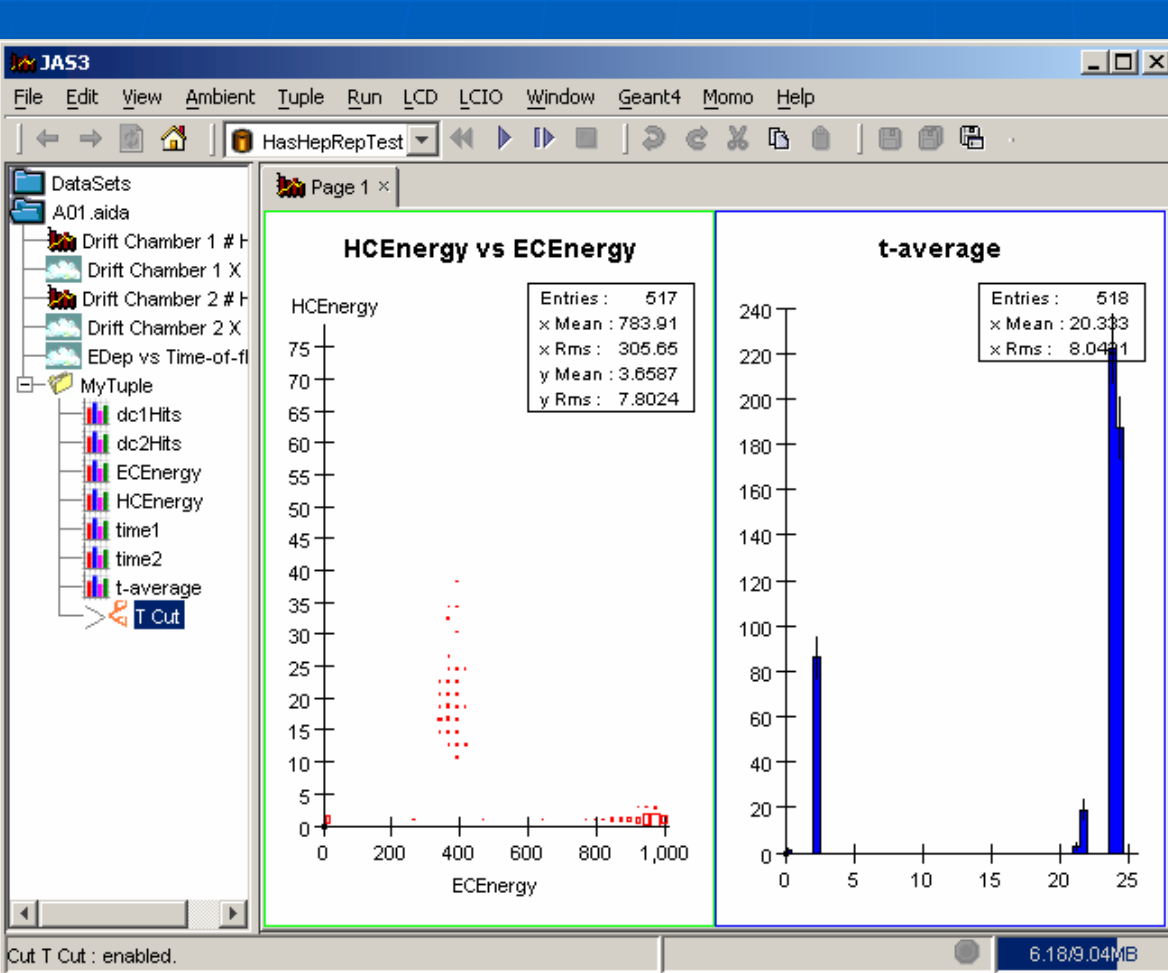
# Using JAS3 to Analyze AIDA files

## ■ Histogram Plotting



# Using JAS3 to Analyze AIDA files

- Tuple Explorer



The 'Define New Column' dialog box shows the following fields:

- Column Name: t-average
- Expression:  $(\text{time1} + \text{time2}) / 2$

Buttons: OK, Cancel, Help.

The 'Add Cut...' dialog box shows the following fields:

- Tab: Numeric 1D Cut
- Cut Name: T Cut
- Cut Type: Cut1 < x < Cut2
- Column: t-average

Buttons: OK, Cancel, Help.

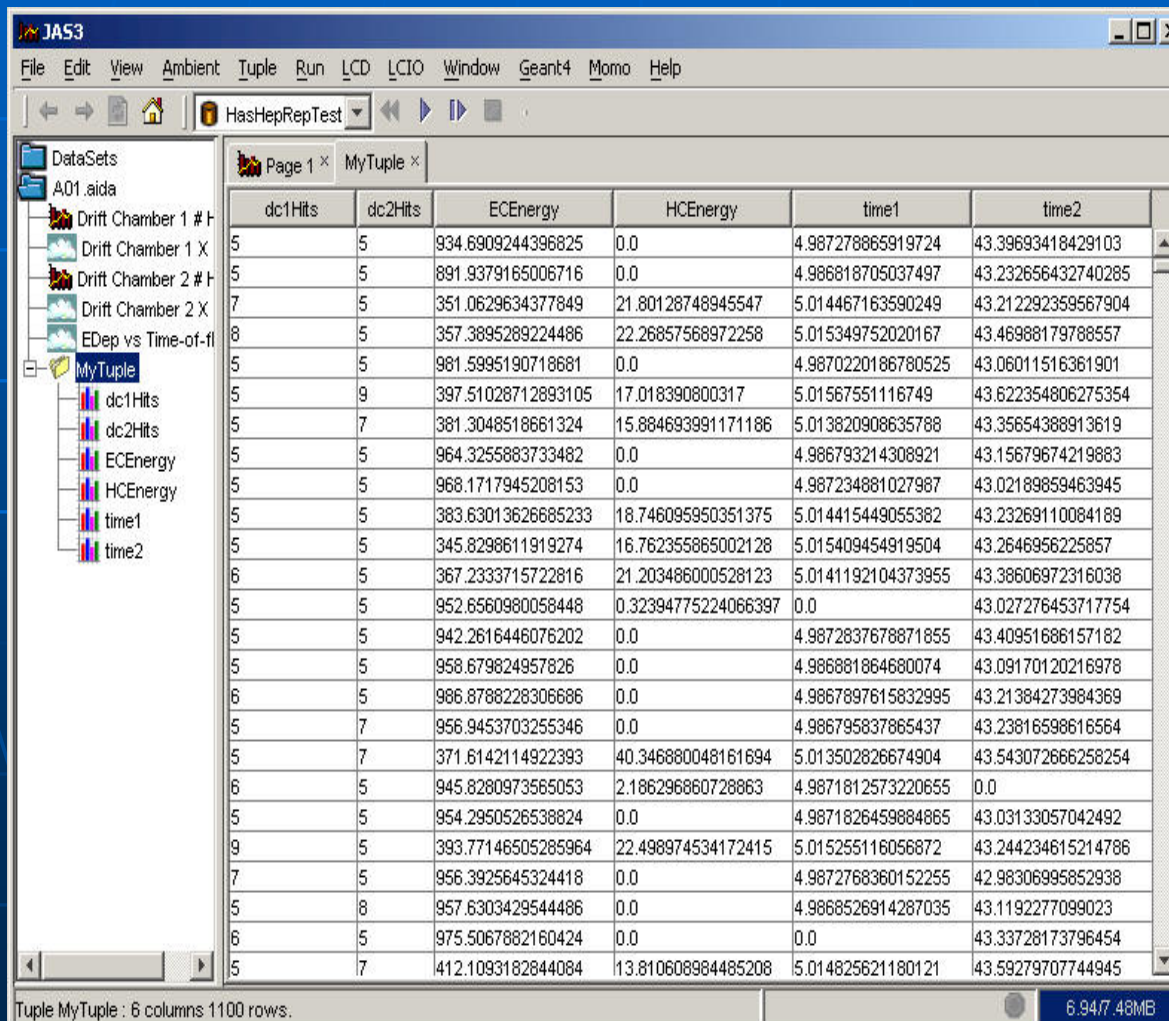
The 'Cut: T Cut' dialog box shows the following fields:

- Value 1: 24.131
- Value 2: 24.393

Buttons: Disabled, Inverted, Sync.

# Using JAS3 to Analyze AIDA files

- Tuple Explorer (cont) - Tabulation



The screenshot shows the JAS3 Tuple Explorer window. The main area displays a table with 6 columns and 1100 rows. The columns are labeled: dc1Hits, dc2Hits, ECEnergy, HCEnergy, time1, and time2. The data is organized into a tree view on the left, with 'MyTuple' selected. The status bar at the bottom indicates 'Tuple MyTuple : 6 columns 1100 rows.' and the file size is '6.947.48MB'.

dc1Hits	dc2Hits	ECEnergy	HCEnergy	time1	time2
5	5	934.6909244396825	0.0	4.987278865919724	43.39693418429103
5	5	891.9379165006716	0.0	4.986818705037497	43.232656432740285
7	5	351.0629634377849	21.80128748945547	5.014467163590249	43.212292359567904
8	5	357.3895289224486	22.26857568972258	5.015349752020167	43.46988179788557
5	5	981.5995190718681	0.0	4.9870220186780525	43.06011516361901
5	9	397.51028712893105	17.018390800317	5.01567551116749	43.622354806275354
5	7	381.3048518661324	15.884693991171186	5.013820908635788	43.35654388913619
5	5	964.3255883733482	0.0	4.986793214308921	43.15679674219883
5	5	968.1717945208153	0.0	4.987234881027987	43.02189859463945
5	5	383.63013626685233	18.746095950351375	5.014415449055382	43.23269110084189
5	5	345.8298611919274	16.762355865002128	5.015409454919504	43.2646956225857
6	5	367.2333715722816	21.203486000528123	5.0141192104373955	43.38606972316038
5	5	952.6560980058448	0.32394775224066397	0.0	43.027276453717754
5	5	942.2616446076202	0.0	4.9872837678871855	43.40951686157182
5	5	958.679824957826	0.0	4.986881864680074	43.09170120216978
6	5	986.8788228306686	0.0	4.9867897615832995	43.21384273984369
5	7	956.9453703255346	0.0	4.986795837865437	43.23816598616564
5	7	371.6142114922393	40.346880048161694	5.013502826674904	43.543072666258254
6	5	945.8280973565053	2.186296860728863	4.9871812573220655	0.0
5	5	954.2950526538824	0.0	4.9871826459884865	43.03133057042492
9	5	393.77146505285964	22.498974534172415	5.015255116056872	43.244234615214786
7	5	956.3925645324418	0.0	4.9872768360152255	42.98306995852938
5	8	957.6303429544486	0.0	4.9868526914287035	43.1192277099023
6	5	975.5067882160424	0.0	0.0	43.33728173796454
5	7	412.1093182844084	13.810608984485208	5.014825621180121	43.59279707744945

# Using JAS3 to Analyze AIDA files

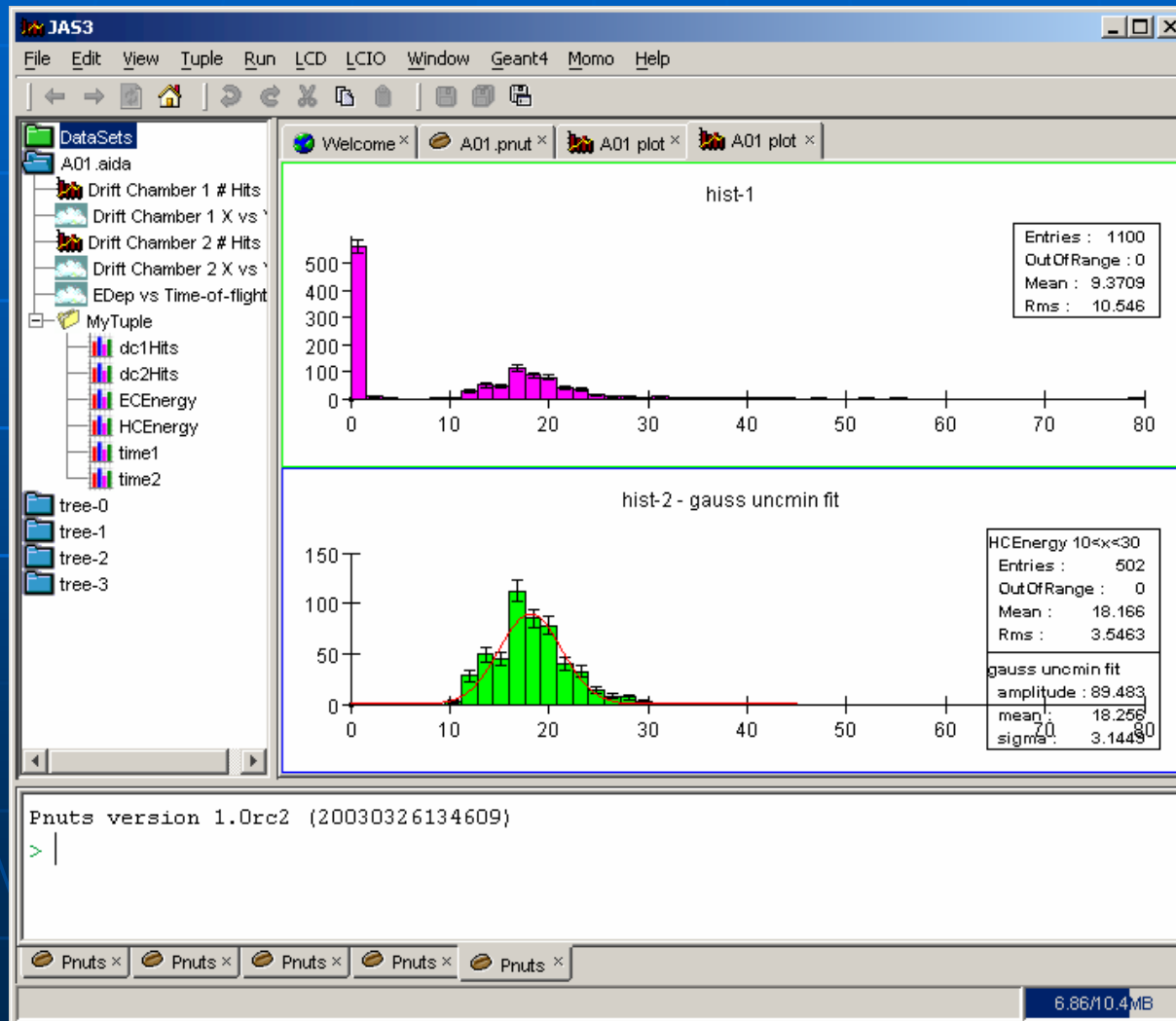
## ■ Scripting

```
IAnalysisFactory = class hep.aida.IAnalysisFactory.  
af = IAnalysisFactory::create().  
tree = af.createTreeFactory().create().  
tf = af.createTupleFactory(tree).  
hf = af.createHistogramFactory(tree).  
.   
// Locate the tuple.  
.   
tuple = aidaMasterTree.find("A01.aida/MyTuple");  
.   
// make a plot of the HCEnergy.  
.   
energy = tf.createEvaluator("HCEnergy").  
h1 = hf.createHistogramD("hist-1", "HCEnergy", 50, 0, 80).  
tuple.project(h1, energy).  
.   
// Make a plot of HCEnergy with a cut.  
.   
h2 = hf.createHistogramD("hist-2", "HCEnergy 10<x<30", 50, 0, 80).  
cut = tf.createFilter("HCEnergy>10 && HCEnergy<30");  
tuple.project(h2, energy, cut).
```

```
// Perform a simple fit.  
.   
ff = af.createFunctionFactory(tree).  
gauss = ff.createFunctionByName("gauss","g").  
gauss.setParameter("amplitude",h2.maxBinHeight()).  
gauss.setParameter("mean",h2.mean()).  
gauss.setParameter("sigma",h2.rms()).  
.   
fitter = af.createFitFactory().createFitter("Chi2").  
result = fitter.fit(h2,gauss).  
.   
// Plot the results.  
.   
plotter = af.createPlotterFactory().create("A01 plot");  
plotter.createRegions(1,2,0);  
plotter.region(0).plot(h1);  
plotter.region(1).plot(h2);  
plotter.region(1).plot(result.fittedFunction());  
plotter.show();
```

# Using JAS3 to Analyze AIDA files

## ■ Scripting (cont)





# Other JAS3 Features

- Reads Paw, Root files (extensible)
- Scripting in Python, Java, Pnuts
  - Built-in editor and compiler
- Built in spreadsheet
- Analysis of complex events
  - (not just n-tuples)
- Highly extensible
- High quality vector-graphics output
  - (pdf, ps, swf, svg etc).
- Grid Enabled

# Analysis Exercise

- Install JAIDA, AIDAJNI
- Recompile A01 example with analysis enabled
- Run example to produce .aida file and view plots
- Look at A01 source code
  - A01AnalysisManager
  - A01EventAction
- Modify to add extra histograms etc
  - AIDA Users Guide
    - On CD, or at: <http://aida.freehep.org/lib/doc/UsersGuide/index.shtml>
    - AIDA Web Site
    - <http://aida.freehep.org/>
- Use JAS3 to view/manipulate .aida file
- Read JAS3 tutorial (on CD) to learn more about JAS3