Introduction to AIDA

• What is AIDA
• History/Collaboration/Documentation
• Examples
  – Examples here are based on AIDA 3.0 and Java implementation of AIDA (JAIDA).
• Some Details
• Conclusion
What is AIDA

- Abstract Interfaces for Data Analysis (AIDA)

- “The goals of the AIDA project are to define abstract interfaces for common physics analysis objects, such as histograms, ntuples, fitters, IO etc. The adoption of these interfaces should make it easier for developers and users to select to use different tools without having to learn new interfaces or change their code. In addition it should be possible to exchange data (objects) between AIDA compliant applications.”
Abstract Interfaces

• Start with “I”, like IHistogram, ITuple, IFitter
• Only define “protocol” that analysis objects have to understand, no internal details
• Can use object only through Abstract Interface
• Define pure virtual methods only
  – AIDA defines behavior, analysis tool provides implementation
• Use Factories to be able to switch implementations easily:
  ✓ Use: IHistogram1D h1 = histogramFactory.createHistogram1D();
  ✓ Not: Histogram1D h1 = new Histogram1D();
Example Program (Java)

- Create, fill, and view 1D and 2D histograms
- Factories:
  - Can switch implementations without recompiling
  - Framework can hide factories

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

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

        IHistogram1D h1d = hf.createHistogram1D("test 1d",50,-3,3);
        IHistogram2D h2d = hf.createHistogram2D("test 2d",50,-30,30,50,-3,3);

        Random r = new Random();
        for (int i=0; i<10000; i++)
            { h1d.fill(r.nextDouble());
              h2d.fill(10*r.nextDouble(),r.nextDouble());
            }

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

• Advantages
  – Write code using interfaces only – IHistogram1D, ITuple, NOT Implementation-specific classes.
  – The user needs to learn only one set of interfaces.
  – Same user code can be used with different AIDA-compliant analysis applications.
  – Different analysis tools can exchange analysis objects
    • same storage format, use functionality from other tools
  – Pool experience of different developer teams
    • LHC++, OpenScientist, JAS

• Currently two versions of AIDA interfaces are generated:
  – One for C++
  – One for Java
AIDA Example

• Use same code with any AIDA-compliant analysis tool.

User code (e.g. GEANT4)  ↔  AIDA  ↔  Analysis tool 1
                           ↔
                           ↔
                           ↔  Analysis tool 2
History

- Initial idea formed during discussion at HepVis-99 workshop at Orsay
  - Informal AIDA discussions at CERN in 2000
  - AIDA workshops:
    - January 2001 - Paris/Orsay
    - April 2001 - Boston (preceding HepVis 2001)
    - Informal meetings (e.g during Geant4 meetings and video conferences)
    - June 2002 – CERN
  - Interfaces have been designed by discussion and (eventual) consensus
    - Takes some time, but result is well thought out and robust
Organization – Code, Documentation

• AIDA – open source project
  – CVS repository: cvs.freehep.org
    • “anonymous” download available
  – Web page: http://aida.freehep.org
    • General information, relevant links
    • Tutorial, users’ guide, examples
    • Downloads and web-browsable source code
AIDA Version 3.0 released (October 2002)
- Many changes from version 2.2
- Three implementations of AIDA exist
  - Anaphe/Lizard (C++)
    - http://anaphe.web.cern.ch/anaphe
  - Open Scientist (C++)
    - http://www.lal.in2p3.fr/OpenScientist
  - JAIDA/JAS (Java) + AIDA-JNI 1.0 (C++)
- First version of proxies to the AIDA interfaces – Root implementation of AIDA histograms and profile in 1D – use of SEAL plugin manager to choose between Root and native AIDA implementations. Release notes are available at:
  - http://lcgapp.cern.ch/project/pi/relnotes.html
- GEANT4 adopted AIDA for analysis
AIDA Interfaces Summary

- **AIDA Main Objects:**
  - AIDA Factories
  - ITree
  - ITuple
  - IHistogram
  - ICloud
  - IProfile
  - IDataPointSet
  - Plotting

- Also see Max Turri talk for more details on:
  - Functions
  - Fitting
  - FitResults
AIDA Factories – IAnalysisFactory

- Factory – “a building or set of buildings with facilities for manufacturing” (Webster Dictionary).
- Objects in AIDA are created by Factories or internally by other objects. User should newer utilize “new”.
  - static IAnalysisFactory create()
    - IHistogramFactory createHistogramFactory (ITree tree)
    - IFunctionFactory createFunctionFactory (ITree tree)
    - ITupleFactory createTupleFactory (ITree tree)
    - ITreeFactory createTreeFactory()
    - IPlotterFactory createPlotterFactory()
    - IDataSetFactory createDataSetFactory (ITree tree)
    - IFitFactory createFitFactory()
AIDA Factories – IHistogramFactory

- Creates 1D, 2D, 3D IHistograms, IClouds, IProfiles:
  - IHistogram1D createHistogram1D (String name, String title, int nBins, double lowerEdge, double upperEdge, String options)
  - IHistogram1D createHistogram1D (String name, String title, int nBins, double[] binEdges, String options)
  - Options: “type=efficiency” – create efficiency histogram.
  - Options: “autoconvert=true” – to enable auto conversion for Clouds.

- Does operations on those objects:
  - Copy and Arithmetic operations add/subtract/divide/multiply
    IHistogram1D add (String name, IHistogram1D h1, IHistogram1D h2)
  - Projections and Slices of 2D and 3D Objects: 2D -> 1D, 3D -> 2D
    IHistogram1D projectionX (String name, IHistogram2D h2D)
    IHistogram1D sliceX (String name, IHistogram2D h2D, int index)
AIDA Factories – IDataPointSetFactory

- **IDataPoint** is holding and managing a single set of "measurements". Say: (value, x, y, z).

- **IDataPointSet** is holding and managing a single set of **IDataPoints**

- **IDataPointSetFactory** creates **IDataPointSets**:
  - Empty **IDataPointSet**
  - **IDataPointSet** filled with data from **IHistogram** or **ICloud**.

- **IDataPointSetFactory** does operations on **IDataPointSets**:
  - Copy and Arithmetic operations add/subtract/divide/multiply
    **IDataPointSet add** (String name, **IDataPointSet** dps1, **IDataPointSet** dps2)
AIDA Factories – IFunctionFactory

- Creates function from a model name registered in the catalog:
  
  Model => ‘e’, ‘g’, ‘g2’, ‘p0’, ‘p1’, ‘p2’, …, ‘p9’
  
  IFunction createFunctionByName (String name, String model)

- Creates function from script (use JEL: Java Expressions Library)
  
  IFunction createFunctionFromScript (String name, int dim, String valueExpr, String parameters, String description)
  
  Example: valueExpr = "a*(1+c*cos(x[0]-d))", parameters = "a,c,d", description="My COS function"

- IFunction cloneFunction (String name, IFunction f)

- User can register its own functions in the IFunctionCatalog (advanced):
  
  IFunctionCatalog catalog ()
AIDA Factories - TupleFactory

- Creates ITuples:
  ITuple create (String name, String title, String[] columnNames, Class[] columnType, String options)

- Chains ITuples. Creates a logical chain of ITuples. All ITuples in the set must have the same structure. Chained ITuple can not be filled. Original NTuple data is not copied.
  ITuple createChained (String name, String title, ITupe[] set)

- Filters ITuples. Creates a new reduced NTuple (less rows) from an existing one by applying a filter. Data is explicitly copied to a new NTuple.
  ITuple createFiltered (String name, ITuple tuple, IFilter filter)
**AIDA Factories - TreeFactory**

- Most AIDA objects appear in an ITree (IManagedObjects that have a name and are displayed in an ITree by that name).

- TreeFactory creates ITrees:
  - Stand-alone ITree:
    ```
    ITree create()
    ```
  - ITree associated with a Store. Valid types are “xml”, “hbook”, “root”. ITree can read data from the Store and, possibly, write to the Store.
    ```
    ITree create(String name, String type)
    ITree create(String name, String type, boolean readOnly, boolean createNew)
    ```
ITree

• **ITree**
  – directory-like structure (Unix directory convention)
    • Methods like: cd, ls, mkdir, etc.
  – AIDA analysis objects (tuples, histograms, clouds, etc.) exist within ITree directories.
  – “save/restore” functionality, hides storage details from the user
    • Compatible with database or file storage
    • Can support multiple file formats (hbook, root, simple text file)
    • Mount/Unmount functionality (like unix) allows multiple stores to be seamlessly merged
    • AIDA XML format is defined for data interchange
ITree – Details

- ITree: Directory-like structure

```java
public interface ITree {
    public String storeName();
    public IManagedObject find(String path) throws IllegalArgument exception;
    public void cd(String path) throws IllegalArgument exception;
    public String pwd();
    public Iterator ls();
    public Iterator ls(boolean recursive);
    public Iterator ls(String path) throws IllegalArgument exception;
    public Iterator ls(String path, boolean recursive) throws IllegalArgument exception;
    public void mkdir(String path) throws IllegalArgument exception;
    public void rmdir(String path);
    public void rm(String path);
    public void mv(String oldPath, String newPath) throws IllegalArgument exception;
    public String findPath(IManagedObject object) throws IllegalArgument exception;
    public void commit() throws IOException;
    public void cp(String oldPath, String newPath) throws IllegalArgument exception;
    public void cp(String oldPath, String newPath, boolean recursive) throws IOException;
    public void symlink(String path, String alias) throws IllegalArgument exception;
    public void mount(String path, ITree tree, String treePath) throws IllegalArgument exception;
    public void unmount(String path) throws IllegalArgument exception;
    public void close() throws IOException;
}
```
1. Stand-alone ITree (no storage):

   IAnalysisFactory af = IAnalysisFactory.create();
   ITreeFactory tf = af.createTreeFactory();
   ITree tree = tf.create();
   IHistogramFactory hf = af.createHistogramFactory(tree);
   IHistogram1D h1 = hf.createHistogram1D("Hist-1", 50, -5., 5.);
   .......
   tree.close();

2. ITree with existing storage, to read data:

   ITree tree = tf.create("/tmp/data.hbook", "hbook", true);  //read-only = true
   IHistogram1D h1 = tree.find("/Hist-1");
   .......
   tree.close();

3. ITree with new storage file, to save your AIDA objects:

   ITree tree = tf.create("/tmp/aida_data.aida", "xml", false, true);
   IHistogramFactory hf = af.createHistogramFactory(tree);
   IHistogram1D h1 = hf.createHistogram1D("Hist-1", 50, -5., 5.);
   .......
   tree.commit();
   tree.close();
ITuple

- **ITuple** - interface to the Data
  - “get/set” methods for double, float, int, …
  - Information about columns: min, max, mean, rms
  - Navigating: start(), next(), skip(int nRows)
  - Project ITuple into 1D, 2D, 3D histogram
  - New features since AIDA 3.0:
    - Support for complex internal structures (subfolders)
    - Chaining of ITuples
ITuple - Details 1

public interface ITuple {
    public String label();

    public void fill(int column, double value) throws IllegalArgumentException;
    public void fill(int column, float value) throws IllegalArgumentException;
    public void fill(int column, int value) throws IllegalArgumentException;

    public void addRow() throws OutOfStorageException;
    public void resetRow();

    public int rows();
    public int columns();
    public void start();
    public boolean next();

    public double getDouble(int column) throws ClassCastException;
    public float getFloat(int column) throws ClassCastException;
    public int getInt(int column) throws ClassCastException;

    public double columnMin(int column);
    public double columnMax(int column);
    public double columnMean(int column);
    public double columnRms(int column);

    public boolean project(IHistogram1D histogram,
                           IEvaluator evaluatorX,
                           IFilter filter);
Example: Create, fill and write out ITuple

```java
IAnalysisFactory af = IAnalysisFactory.create();
ITreeFactory tf = af.createTreeFactory();
ITree tree = tf.create("TestTupleData.aida", "xml", false, true);

ITupleFactory tuf = af.createTupleFactory(tree);

String[] columnNames = new String[] { "Row_ID=-1", "Energy=0.", "Relative_Pz=0." };  
Class[] columnTypes = new Class[] { Integer.TYPE, Double.TYPE, Double.TYPE };  
ITuple t1 = tuf.create("Name: TestTuple", "Title: TestTuple", columnNames, columnTypes);  
Random r = new Random();  
for (int i=0; i<100; i++) {
    t1.fill(0, i);
    t1.fill(1, 100*r.nextGaussian());
    t1.fill(2, 2*r.nextGaussian()-1.);
    t1.addRow();
}

tree.commit();
tree.close();
```
IEvaluators and IFilters

• They are created by ITupleFactory:
  – `createEvaluator` (String expression);
  – `createFilter` (String expression);

• Evaluate expression based on current ITuple row:
  – double value = evaluator. `evaluateDouble`();
  – boolean value = filter. `accept`();

• Can evaluate different ITuples, so must be initialized before use:
  – filter. `initialize` (ITuple myTuple);
  – evaluator. `initialize` (ITuple myTuple);
IHistogram (1D-3D)

- Binned histogram: IHistogram1D, 2D, 3D
  - “fill” methods (with/without weight)
  - Histogram info: entries, mean, rms, axis
  - Bin info: centre, entries, height, error
  - Histogram arithmetic: add, multiply, divide
  - Convenience methods, like coordinate-to-index conversion
IHistogram – Details 1

IHistogram:
Common functionality for all histograms (like entries, label, dimension,)

IHistogram1D

IHistogram2D

IHistogram3D

public interface IHistogram1D extends IHistogram {
    public void fill(double x);
    public void fill(double x, double weight);
    public double binCentre(int index);
    public int binEntries(int index);
    public double binHeight(int index);
    public double binError(int index);
    public double mean();
    public double rms();
    public IAxis axis();
    public int coordToIndex(double coord);
}
ICloud

- Unbinned collection of points: ICloud1D, 2D, 3D
  - Can represent scatter plot, dynamically rebinnable histogram
  - Can be converted to a binned histogram
  - Standard “get/set” methods for entries
  - Collection info: lower, upper, mean, rms
ICloud – Details 1

- Unbinned collection of points: ICloud1D, 2D, 3D
  - Can represent scatter plot, dynamically rebinnable histogram
  - Can be automatically converted to a binned histogram

ICloud:
Common functionality for all histograms (like entries, label, dimension,)

ICloud1D

ICloud2D

ICloud3D

```
public interface ICloud2D extends ICloud {

    public boolean fill(double x, double y);
    public boolean fill(double x, double y, double weight);

    public double lowerEdgeX();
    public double lowerEdgeY();
    public double upperEdgeX();
    public double upperEdgeY();

    public double valueX(int index);
    public double valueY(int index);
    public double weight(int index);

    public double meanX();
    public double meanY();
    public double rmsX();
    public double rmsY();

    public void convert(int nBinsX, double lowerEdgeX, double upperEdgeX,
                        int nBinsY, double lowerEdgeY, double upperEdgeY);

    public IHistogram2D histogram() throws RuntimeException;
}
```
**IFunction and IModeFunction**

- **IFunction – Simple**
  - Has value, variables, parameters
  - May provide its gradient

- **IModeFunction – advanced**
  - Extends IFunction
  - Has validity range
  - Can be normalized on the validity range
  - May provide gradient with respect to parameter

- More on Functions and Fitting in Max Turri talk.
Future

• Remote Access to AIDA:
  – Access to remote AIDA objects (through CORBA, RMI, SOAP, …) from another AIDA implementation.
  – Access to AIDA objects through WEB Browser (with limited functionality)
Conclusion

• AIDA is becoming mature project
  – Functionally sound
  – Regular meetings
  – Release schedule
• Plan to have next AIDA workshop July 2003 at CERN.
• JAS 3 – AIDA Compliant Analysis Tool
  – Easy to use and extend
  – Uses Java implementation of AIDA for analysis