

Accessing SimTrackerHit information

A simple example of how to access the SimTrackerHits in an event.

Download the attachment for a compilable version of the following annotated code.

Explanation of the **TrackerHitAccessDriver** Driver Code

```
import java.util.List;
import org.lcsim.event.EventHeader;
import org.lcsim.event.EventHeader.LCMetaData;
import org.lcsim.event.SimTrackerHit;
import org.lcsim.geometry.IDDecoder;
import org.lcsim.util.Driver;
import org.lcsim.util.aida.AIDA;
import hep.aida.ITree;

/*
 * TrackerHitAccessDriver.java
 *
 * Created on July 31, 2005, 3:03 PM
 */

/**
 *
 * @author Norman A. Graf
 */
public class TrackerHitAccessDriver extends Driver
{
    private AIDA aida = AIDA.defaultInstance();
    private ITree _tree;

    public TrackerHitAccessDriver()
    {
        _tree = aida.tree();
    }
    protected void process(EventHeader event)
    {
        List<List<SimTrackerHit>> simTrackerHitCollections = event.get(SimTrackerHit.class);
        for ( List<SimTrackerHit> simTrackerHits : simTrackerHitCollections )
        {
            LCMetaData meta = event.getMetaData(simTrackerHits);
            IDDecoder decoder = meta.getIDDecoder();
            for (SimTrackerHit trackerHit : simTrackerHits)
            {
                decoder.setID(trackerHit.getCellID() );
                int layer = decoder.getLayer();
                double[] pos = trackerHit.getPoint();
                aida.cloud2D(meta.getName()+" layer "+layer+" x vs y").fill(pos[0], pos[1]);
            }
        }
    }
}
```

First, as in all Java programs, there are the import statements:

```
import java.util.List;
import org.lcsim.event.EventHeader;
import org.lcsim.event.EventHeader.LCMetaData;
import org.lcsim.event.SimTrackerHit;
import org.lcsim.geometry.TrackerIDDecoder;
import org.lcsim.util.Driver;
import org.lcsim.util.aida.AIDA;
import hep.aida.ITree;
```

[EventHeader](#) provides access to the event data.

[SimTrackerHit](#) is the tracker detector hit.

[LCMetaData](#) and [TrackerIDDecoder](#) provide access to tracker detector information encoded in the hit IDs. Note that we cast from the default IDDecoder which the meta data returns.

[Driver](#) is a steering routine which can deal with event processing.

The aida classes are used for histogramming.

Next is the declaration of the Driver class.

```
public class TrackerHitAccessDriver extends Driver
```

To be run from within JAS3, one needs a no-argument constructor.

```
public TrackerHitAccessDriver()
{
    _tree = aida.tree();
}
```

In our case we use this to instantiate the aida tree.

We choose here only to override the process() method of Driver. This is called once per event during the event loop. Its single argument is the EventHeader of the current LCIO event.

```
protected void process(EventHeader event)
```

All collections in this event are accessible through the EventHeader interface.

We are interested in accessing the SimTrackerHits for the tracking detectors. We could use the method [getSimTrackerHits\(String name\)](#), but we choose not to in this example, since we do not, a priori, know the names of the collections in an arbitrary LCIO file. Instead, we fetch all the collections which contain objects of type SimTrackerHit, viz.

```
List<List<SimTrackerHit>> simTrackerHitCollections = event.get(SimTrackerHit.class);
```

Here we are using the [event.get\(java.lang.Class\)](#) method, which returns a list of lists of tracker hits. That is, each tracker subdetector provides a list of SimTrackerHits.

Next, we loop over this list of SimTracker collections using the Java 1.5 loop syntax

```
for ( List<SimTrackerHit> simTrackerHits : simTrackerHitCollections )
{
```

Each collection contains metadata which provides more information about the content. We fetch this via

```
LCMetaData meta = event.getMetaData(simTrackerHits);
```

and obtain a hit ID decoder, which will be used later to access additional information about the tracker hits. We cast this to a TrackerIDDecoder since we know that we are dealing with tracker hits.

```
TrackerIDDecoder decoder = (TrackerIDDecoder) meta.getIDDecoder();
```

Finally we loop over the SimTrackerHits in the collection:

```
for (SimTrackerHit trackerHit : simTrackerHits)
{
```

Each hit contains a packed ID, which we can interpret using the IDDecoder obtained from the collection metadata.

```
decoder.setID(trackerHit.getCellID() );
int layer = decoder.getLayer();
```

For illustration, we plot the x vs. y position of the tracker hits.
First, we fetch the position

```
double[] pos = trackerHit.getPoint();
```

We currently return an array of doubles, which we interpret as the (x,y,z) position. This should, of course, be encapsulated into a space point class to remove this ambiguity of interpretation. Look for this in a future release.

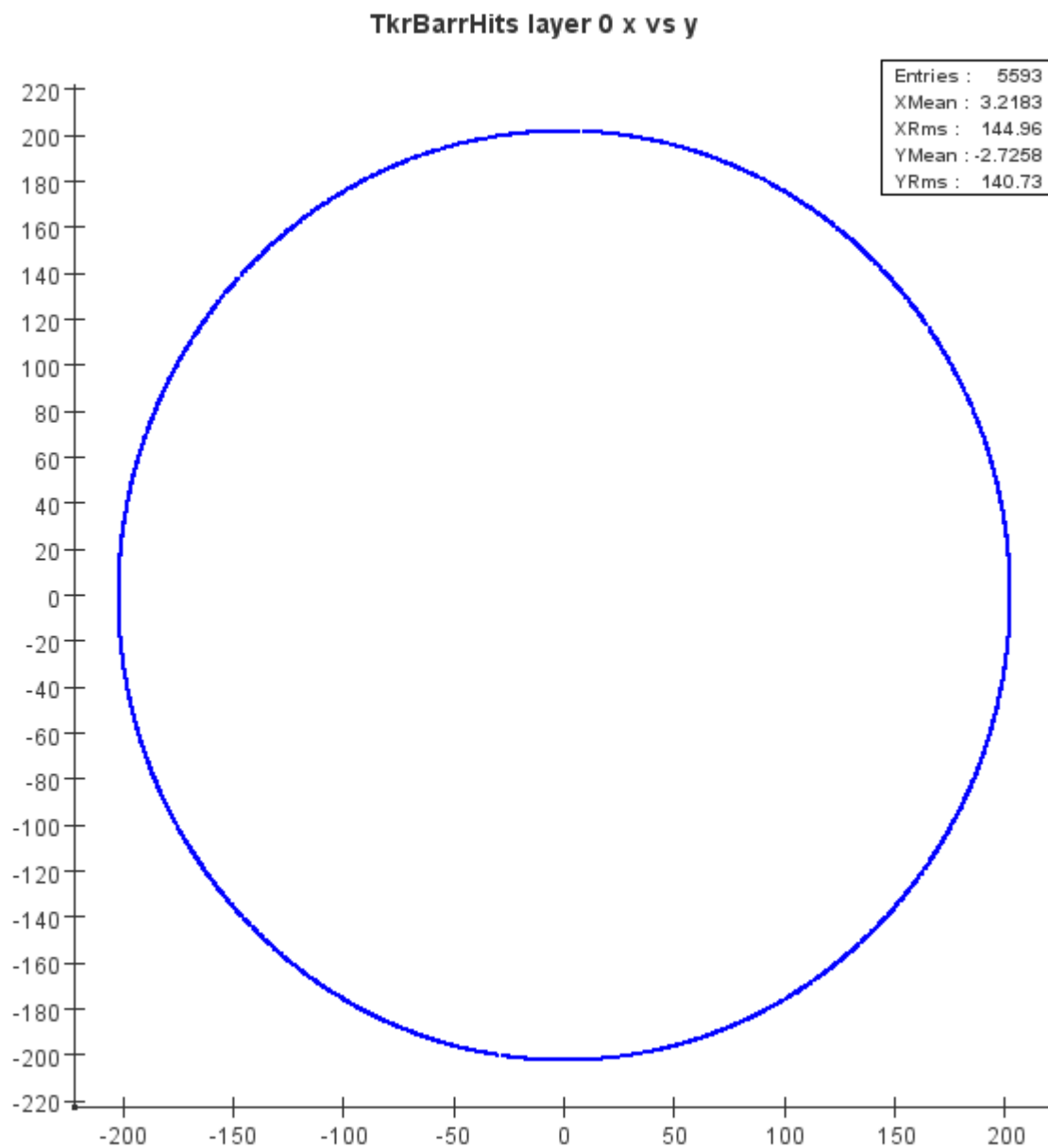
Finally we fill the histogram:

```
aida.cloud2D(meta.getName()+" layer "+layer+" x vs y").fill(pos[0], pos[1]);
```

This one line contains a lot of functionality, so merits some extra attention.

First, a cloud2D is a two-dimensional tuple. The cloud2D method of the AIDA object takes a string which is the title. We construct the title from the name of the tracker collection (obtained from the metadata via its getName() method) and add the layer number as well. For example, we might end up with a tuple titled "**TkrBarrHits layer 0 x vs y**" for the x,y hit position of SimTrackerHits on layer 0 of the collection TkrBarrHits. If this cloud does not already exist, it will be created, else the instance is returned. We then fill the tuple with the cloud2D fill() method, where pos[0] and pos[1] are understood to be the x and y position, respectively.

Here is an example plot showing the x vs y hit position for layer 0 of a silicon strip barrel detector:



Not too exciting, but now you know how to access the tracker hit information in an event.