Customizing the Event Display

The WIRED4 event display is set up to understand and display all standard LCIO objects such as tracks, clusters, reconstructed particles (including jets), calorimeter and tracker hits etc. Often during development of analysis algorithms it is desirable to display extra objects. This mini-tutorial shows how to do this within the org.lcsim framework.

The org.lcsim framework and the WIRED4 event display are not tied to each other, instead there is code in org.lcsim which converts the event into intermediate HepRep objects.

If You Don't Like To Read Long Docs

Let's go through the example of how to put a list of `Hep3Vectors` into the event and transform it to a list of HepRep object that can be displayed by WIRED. The full code is in the class `org.lcsim.util.heprep.Hep3VectorConverter`

```
class Hep3VectorConverter implements HepRepCollectionConverter {
   public boolean canHandle(Class k) {
      return Hep3Vector.class.isAssignableFrom(k);
   }
   public void convert(EventHeader event, List collection, HepRepFactory factory, HepRepTypeTree typeTree,
HepRepInstanceTree instanceTree) {
      LCMetaData meta = event.getMetaData(collection);
      String name = meta.getName();
      int flags = meta.getFlags();

      HepRepType typeX = factory.createHepRepType(typeTree, name);
      typeX.addAttValue("layer",LCSimHepRepConverter.HITS_LAYER);
      typeX.addAttValue("drawAs","Point");
      typeX.addAttValue("color",Color.YELLOW);
      typeX.addAttValue("fill",true);
      typeX.addAttValue("fillColor",Color.RED);
      typeX.addAttValue("MarkName","Box");

      for (Hep3Vector hit : (List<Hep3Vector>) collection) {
         HepRepInstance instanceX = factory.createHepRepInstance(instanceTree, typeX);
         HepRepPoint pp = factory.createHepRepPoint(instanceX,hit.x(),hit.y(),hit.z());
      }
   }
}
```

Now we need to tell org.lcsim that it can display Hep3Vectors: Let's edit the constructor of the class `package org.lcsim.util.heprep.LCSimHepRepConverter()`

```
public LCSimHepRepConverter()
   {
      try
      {
         factory = HepRepFactory.create();
         register(new CalorimeterHitConverter());
         register(new ClusterConverter());
         register(new MCParticleConverter());
         register(new SimTrackerHitConverter());
         register(new TrackerHitConverter());
         register(new ZvTubeConverter());
         register(new TrackConverter());
         register(new ReconstructedParticleConverter());
         register(new ZvVertexConverter());
         register(new Hep3VectorConverter());
      }
      catch (Exception x)
      {
         throw new RuntimeException("Could not create heprep factory",x);
      }
   }
```

We just have to make sure there is a line to register a new instance of the class we just wrote. As you can see, it's already there.
That's all there is to it. If you want to know more about the inner workings, read on, reader.

If You Have To Read Longer Docs

Let's go through the code step by step:
The basic skeleton looks like this:

```
class Hep3VectorConverter implements HepRepCollectionConverter {
   public boolean canHandle(Class k) {
      return Hep3Vector.class.isAssignableFrom(k);
   }
   public void convert(EventHeader event, List collection, HepRepFactory factory, HepRepTypeTree typeTree,
HepRepInstanceTree instanceTree) {
   }
}
```

The convention is to give the class the name of the class that you wish WIRED to be able to display, followed by "Converter". It has to implement the `HepRepCollectionConverter`, because we want to pick up a collection from the `EventHeader` cache and display its elements.

The first function

```
public boolean canHandle(Class k) {
      return Hep3Vector.class.isAssignableFrom(k);
   }
```

tells org.lcsim about the ability of this class. For every list in the EventHeader cache org.lcsim encounters, it tries to match the elements to the registered `HepRepCollectionConverter` instances. The order in which you register your instance therefore matters. If the class of the elements returns `true` for the `boolean canHandle()` function, then this converter will be used to display the elements of the list.

The second function

```
   public void convert(EventHeader event, List collection, HepRepFactory factory, HepRepTypeTree typeTree,
HepRepInstanceTree instanceTree) {
      LCMetaData meta = event.getMetaData(collection);
      String name = meta.getName();
      int flags = meta.getFlags();

      HepRepType typeX = factory.createHepRepType(typeTree, name);
      typeX.addAttValue("layer",LCSimHepRepConverter.HITS_LAYER);
      typeX.addAttValue("drawAs","Point");
      typeX.addAttValue("color",Color.YELLOW);
      typeX.addAttValue("fill",true);
      typeX.addAttValue("fillColor",Color.RED);
      typeX.addAttValue("MarkName","Box");

      for (Hep3Vector hit : (List<Hep3Vector>) collection) {
         HepRepInstance instanceX = factory.createHepRepInstance(instanceTree, typeX);
         HepRepPoint pp = factory.createHepRepPoint(instanceX,hit.x(),hit.y(),hit.z());
      }
   }
```

is used to communicate between org.lcsim objects and its HepRep representations. For each type of element you want to display, you will have to create a `HepRepType`. The name can be the name of the collection in the EventHeader (as is the case in the example) or any string you want. Please make sure it is **unique**.
We then add some properties to the representation. A complete list of available options and parameters is posted elsewhere.
Finally, for each element in the list of org.lcsim objects (`Hep3Vector` in this case) we create a `HepRepInstance` object. It inherits its properties from the `HepRepType` it belongs to, but you can overwrite the properties. You might, for example want to assign certain colors to different instances, grouping them by momentum, parent, distance, or whatever you can think of. Now that we have created a `HepRepInstance`, we want to draw it. In order to do that we need to create one (or more) `HepRepPoint` instances. In this example, where we draw the `Hep3Vector` as a point, one `HepRepPoint` is sufficient, but there may be cases where you want to create a more complex representation that requires more points to draw.
Please have a look at the classes in the package `org.lcsim.util.heprep` for further inspiration.

References

- DrawAs Values
- Attribute Defaults