Journal of Physics: Conference Series **396** (2012) 022002 doi:10.1088/1742-6596/396/2/022002

iSpy: a powerful and lightweight event display

G Alverson[1], G Eulisse[2], T McCauley[2], and L Taylor[2]

[1] Northeastern University, Boston MA 02115, USA
[2] Fermi National Accelerator Laboratory, Batavia IL 60510, USA

E-mail: george.alverson@cern.ch, giulio.eulisse@cern.ch, thomas.mccauley@cern.ch, lucas.taylor@cern.ch

Abstract. iSpy is a general-purpose event data and detector visualization program that was developed as an event display for the CMS experiment at the LHC and has seen use by the general public and teachers and students in the context of education and outreach. Central to the iSpy design philosophy is ease of installation, use, and extensibility.

The application itself uses the open-access packages Qt4 and Open Inventor and is distributed either as a fully-bound executable or a standard installer package: one can simply download and double-click to begin. Mac OSX, Linux, and Windows are supported. iSpy renders the standard 2D, 3D, and tabular views, and the architecture allows for a generic approach to production of new views and projections.

iSpy reads and displays data in the ig format: event information is written in compressed JSON format files designed for distribution over a network. This format is easily extensible and makes the iSpy client indifferent to the original input data source. The ig format is the one used for release of approved CMS data to the public.

1. Introduction

Event displays are invaluable tools that find many uses in high-energy physics experiments including geometry viewing, development of algorithms, and detector monitoring. They also are used to produce images for communication of physics results to the public. iSpy [1] was developed for the CMS experiment [2] at the LHC. The display and the file format it uses are still used in the traditional manner but have also found use in education and outreach. We describe in the following how the iSpy design philosophy was implemented and its implications for new and continued usage.

2. Release and distribution

Executables for iSpy on Linux (32 and 64-bit), Mac OSX, and Windows are available from the iSpy web page. Under Linux and Mac OSX, iSpy is built as a fully-bound executable. The user has only to download the executable and run. The Windows distribution is provided as a standard Windows Installer installation package, allowing for the easy installation, upgrade, or removal of iSpy.

3. Use-cases

In CMS the iSpy application has been used to create images of events online in the CMS control room [3] as well as offline for public use in the press. iSpy is also being developed for use in the CMS control room as part of the detector control systems (DCS) environment.

Data preservation and release of data to the public has become a major initiative of all LHC experiments, from the level of plots to source code and raw data[4]. On the CMS experiment ig files are the current implementation of "Level 2" data preservation, which is a description of event information in a simplified format. More information can be found in the CMS data preservation, re-use, and open-access policy [5].

The ease-of-use and extensibility of the iSpy application and the ig files have allowed for many different (and in some case unanticipated) use-cases. The CMS collaboration has agreed to release a small fraction of its data to the public for education and outreach, using the ig format. In the context of education and outreach the data in ig format is analyzed and used by students and teachers in masterclasses [6] and eLabs [7]. A browser-based event display has also been written which reads ig files and has a similar look-and-feel as iSpy. For more on the education programs and the display see [8]. The iSpy application has also been used in these educational settings.

An unanticipated use-case of the ig-formatted files has been in "Science Hack Days", where members of the general public meet over two days and collaborate on many different science and technology-oriented projects. Thus far, CMS data has been re-used to create new visualizations in events held in San Francisco and Nairobi [9].

With the addition of Ruby plugins, ig files can be read into the SketchUp[10] 3D modeling application, allowing one to easily create images and export files in various 3D formats. One can even create what is equivalently an event display using SketchUp.

4. Data format

The input format for iSpy is the ig format. An ig file is simply a zip archive containing event files in JSON format. There is a directory for each run and in this directory one finds the events for that particular run. Physics event and graphics information, such as positions in global coordinates, are contained in the JSON files.

The ig event file format is a valid JSON or python dictionary. There are three main keys in the dictionary: Types, Collections, and Associations. Types maps a type name (the name of a collection) with type attributes. For example, for an event just containing event information and tracks the Types are:

```
"Types": {"Event_V2": [["run", "int"],["event", "int"],["ls", "int"],
                       ["orbit", "int"],["bx", "int"],["time", "string"],
                       ["localtime", "string"]],
          "Tracks_V2": [["pos", "v3d"],["dir", "v3d"],["pt", "double"],
                        ["phi", "double"],["eta", "double"],
                        ["charge", "int"],["chi2", "double"],
                        ["ndof", "double"]]
          }
```

Collections contain the specific information for each instance (for example, two `Tracks_V2`) of the objects specified in Types:

```
"Collections": {"Tracks_V2": [[[0.00100636, 5.76383e-05, 0.074605],
                               [-0.858741, -0.391305, -0.330824],
                               0.849145, -2.71403, -0.343753, -1, 4.64884, 12],
                              [[0.000916733, 0.000224635, 0.074523],
                               [-0.796949, -0.587943, 0.138546],
                               0.879058, -2.50598, 0.139443, -1, 25.8956, 14]]
                }
```

Finally, Associations (as the name implies) associates two specific instances of different collections to each other. An association is identified by two pairs of numbers. The first number

in each pair specifies the index of the collection in Types. The second number in each pair specifies the index of the object in Collections. For example, one can have two `Tracks_V1` objects in an event file and have a collection of hits called `Hits_V1`. It is the association set (called for example `TrackHits_V1`) that specifies the relationship between specific tracks and hits. An example can be seen below:

```
"Associations": {"TrackHits_V1":[[[1,0],[2,0]],
                                 [[1,0],[2,1]],
                                 [[1,0],[2,2]],
                                 [[1,0],[2,3]],...[[1,0],[2,9]],
                                 [[1,1],[2,10]],
 [[1,1],[2,11]],...[[1,1],[2,15]]}
```

Here, the tracks have a collection index of 1 and the hits a collection index of 2. Associations are specified by the writer of the ig file.

The ig format has several beneficial features. For one, the files are self-documenting, containing a schema describing the contents. The schema evolution is handled simply by incrementing the version underscore in the collection name. By adding the new collection to the configuration described in the next section, iSpy is backwards-compatible. The JSON format is easily parsed and written using C++, python, ruby, and JavaScript; the files are of course human-readable. A C++ API [11] and a python [12] are available. The ig files themselves are created using the software framework of the CMS experiment (CMSSW), converting CMSSW objects into ig format. Users therefore require no special knowledge of CMS software and are also insulated from possible differences in versions of CMS software and event formats. This feature, along with its flexibility and extensibility, allows the ig format to be in principle experiment independent.

5. Features

The graphical user interface for iSpy is written in Qt (version 4) [13] and 2D and 3D rendering is done using Open Inventor (Coin3D) [14].

5.1. Menus and controls

Menu controls for iSpy are simple and minimal: File and Help menus are the only menus for the general user. A Tools menu is for experts and displays the settings. The File menu contains Open, Print, and Save options. Files may be opened from disk, from a specified url, or from the web. In the last case, example event display files are provided from a linked website.

5.2. Views

The iSpy display is partitioned into three different zones. A tree view displays the contents of the event files in a two-level hierarchy. Each selectable object corresponds to a Collection in the ig file. Whether or not a Collection is displayed is determined from a checkbox in the tree view. Selection of the Collection displays its contents in the second major area of the display, the table view. In the table view one can examine the full content of the Collection and the values of each individual element. Each column is sortable.

The last major area in the display is the graphical view, rendered using Open Inventor. In this view, standard 2D views such as R-Phi and R-Z are available as well as 3D and Lego. Views can be easily added or modified as required. To add a new view one simply has to specify it and its attributes (such as label and projection) and the collections it contains in the view configuration file, which described in more detail in the next section.

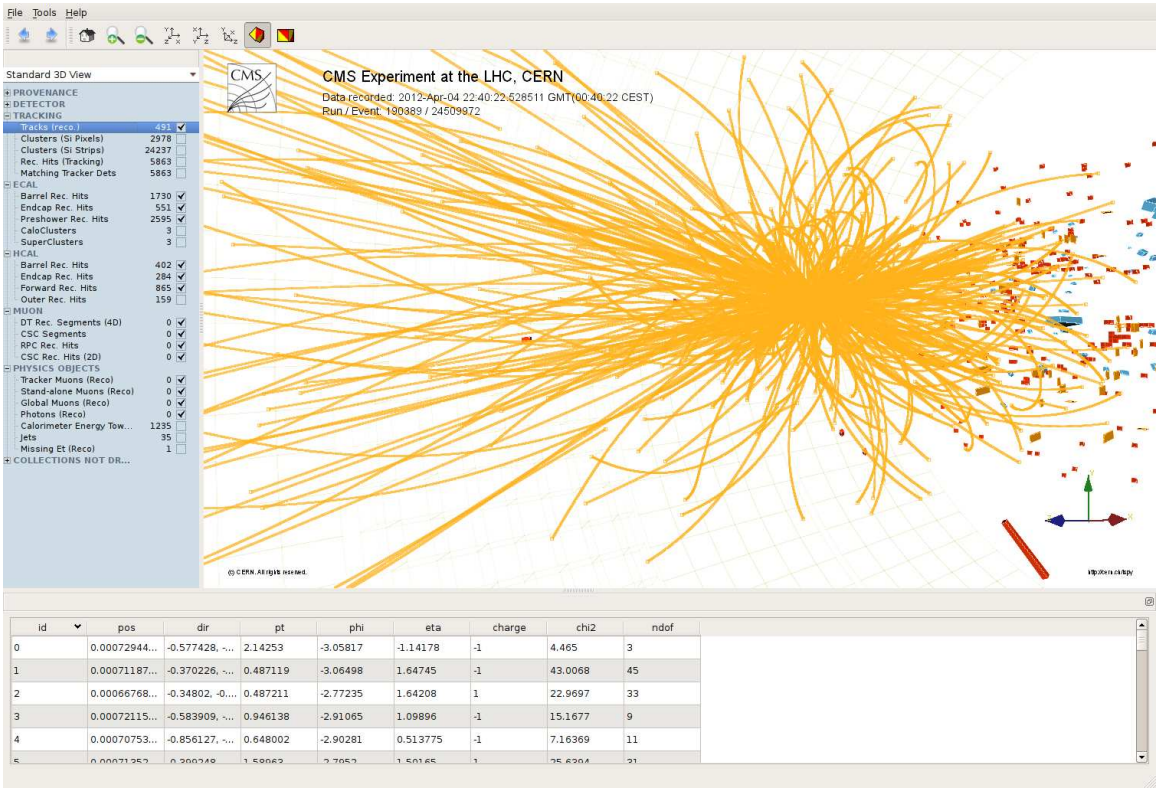

Figure 1. Screenshot of iSpy application showing controls, and tree, table, and 3D views.

6. Configuration

Configuration of graphical properties such as colors and physical properties such as the minimum energy of calorimetry objects is controlled from a iss file, the syntax of which mimics CSS syntax:

```
rule {
  property-name: property-value;
  ...
}
```

As of this writing the default * rule is:

```
* {
  diffuse-color: rgb(0.7, 0.7, 0.7);
  transparency: 0.0;
  line-width: 1.0;
  line-pattern: 0xffff;
  font-size: 12;
  font-family: Arial;
  draw-style: solid;
  marker-shape: square;
  marker-size: normal;
  marker-style: filled;
  text-align: left;
  min-energy: 0.2;
  max-energy: 5.0;
```

```
  energy-scale: 1.0;
  left: 0.;
  top: 0.;
}
```

View layout is controlled via an iml file, which is an xml file with the following example structure:

```
<?xml version="1.0" encoding='UTF-8'?>
<layout>
  <camera position="-18.1, 8.6, 14.0" pointAt="0, 0, 0"
          scale="10.6" orthographic="true" rotating="true">
    <visibilityGroup>
      <view label="Standard 3D View">
         <collection label="Provenance/Event information"
                     spec="Event_V1:time:run:event:ls:orbit:bx"
                     draw="make3DEvent"/>
      </view>
    </visibilityGroup>
  </camera>
</layout>
```

The `<layout>` tag is the top-level container and `<visibilityGroup>` specifies where the default camera should be for the contained views. All of the attributes for the `<camera>` element are shown in the above example. Within each view, which includes 3D, R-Phi, R-Z, and Lego, there are any number of collections specified. The **label** attribute specifies the label of the particular collection in the tree view. **spec** specifies the name of the collection in the ig file as well as the required data in the collection needed for rendering. **draw** specifies the method in the code used for drawing the collection. Two other attributes, **association** and **other**, specify, if needed, another collection and how it is associated with the collection in **spec**. Another attribute **projection** exists as well. Each drawing method takes as an argument an instance of a Projection class. Within the Projection class itself there are several projections specified, such as the common RZ projection.

Default settings for style and view are compiled into the released executable. After release, users can modify their own iml and iss files and use them by specifying them on the command line.

Another nice feature of iSpy's configuration management is that new objects, provided they contain the required data for a particular draw method, can be added to the display by appropriately modifying the iml and iss files. Say for example (this describes an actual use-case) that one wants to display a previously unsupported object and draw it as a line in the 3D view. In this case, there is already a supported method in iSpy called **make3DPointSetShapes**, which needs one position as input. One then writes analysis code in CMSSW (templates are provided) to make sure that a collection (*e.g.* called CSCLCTDigis_V1) is written out in the ig file and contains the collection type **pos**. Adding the following bit of xml to the view file in the `<view>` tag (as above)

```
<collection label="Muon/CSC LCT Digis"
    spec="CSCLCTDigis_V1:pos"
            draw="make3DPointSetShapes"/>
```

will add the object to the display. One can modify colors and other attributes by adding the new object to the style file. Otherwise default values are used. No new C++ code or recompilation is needed.

7. Conclusions and future plans

The iSpy design philosophy of ease of use, installation, and extensibility has been implemented and demonstrated in the iSpy application and ig format.

The style and view configuration system for iSpy, along with distribution of a fully-bound executable, make it such that frequent releases of the application are not necessary. Maintenance of the converters from CMSSW format to ig to keep up with releases and changes in CMSSW is where most work is needed.

Another benefit of the extensibility of iSpy is that users can add objects to the ig files and to the display via the style and view configuration. Collecting these changes (that have been used and tested for some time) into a release will allow all users to benefit. The most recent release (currently in beta) is version 1.5.0 (available from `http://cern.ch/ispy`).

We plan to maintain the ig format converters and APIs as well as add new features (such as correlated picking).

Acknowledgments

We wish to thank former developers Shahzad Muzaffar, Ianna Osborne, and Lassi Tuura as well as support from Luis Lopera Gonzalez and Zhang Jinzhong. We also acknowledge the support of the US DOE and NSF.

8. References

[1] `http://cern.ch/ispy`
[2] `http://cern.ch/cms`
[3] G Alverson *et al.* 2010 J. Phys.: Conf. Ser. 219 032054
[4] `http://www.dphep.org`
[5] `https://cms-docdb.cern.ch/cgi-bin/PublicDocDB/ShowDocument?docid=6032`
[6] `http://www.physicsmasterclasses.org`
[7] `http://www.i2u2.org/elab/cms`
[8] M Hategan *et al. A browser-based event display for the CMS experiment at the LHC*, these proceedings
[9] `http://www.sciencehackday.com`
[10] `http://sketchup.google.com`
[11] `https://github.com/tpmccauley/igfiles`
[12] `https://github.com/tpmccauley/pyig`
[13] `http://qt.nokia.com`
[14] `http://www.coin3d.org`