The FRED Event Display

an Extensible HepRep Client for GLAST

R.Giannitrapani
M.Frailis

CHEP'03 (San Diego), 24 March 2003

Dipartimento di Fisica
Università degli Studi di Udine

Contents

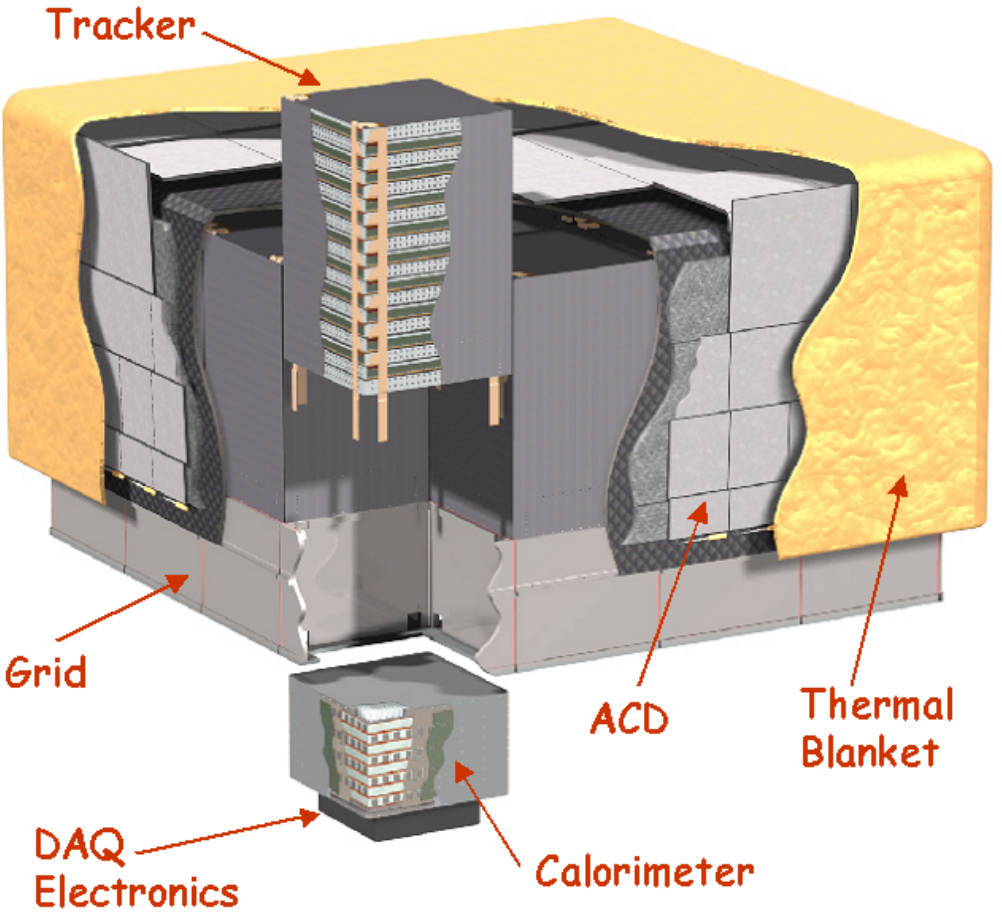
This talk reviews the present situation of **FRED**, an event display **proposal** for **GLAST**. FRED is part of a bigger framework we are working on for GLAST; details on such a framework are discussed in a companion talk that will be presented later during the week.

We wish to thank all the GLAST people for helpful discussions on event display and related issues. A big thank also to J.Perl and M.Donszelmann; without them and their HepRep and WIRED, we never started FRED.

Slides made in ConTEXt, edited in emacs, rendered in PDF.

Introduction

- GLAST measures the direction, energy and arrival time of celestial gamma rays
  - ▷ **LAT** measures gamma-rays in the energy range from ~ 20 MeV to more than 300 GeV. **There is no telescope now covering this range!!**
    - Array of 16 identical Tower Modules, each with a **tracker** (Si strips) and a **calorimeter** (CsI with PIN diode readout) and DAQ module.
    - Surrounded by finely **segmented ACD** (plastic scintillator with PMT readout).
  - ▷ **GBM** provides correlative observations of transient events in the energy range from ~ 20 keV to 20 MeV.
- **Launch**: September 2006, Florida
- **Orbit**: 550 km, 28.5° inclination
- **Lifetime**: 5 years (minimum)

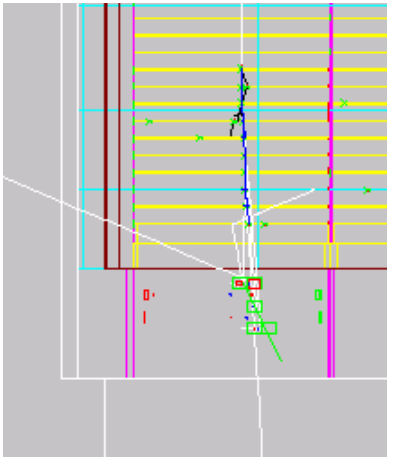


Requirements

GLAST has "normal" user requirements for its event display

- No revolutions in the collaboration software
- No tight dependencies from the collaboration software
- Must be flexible (and rigid)
- It should be easy
- It should be performant
- It must be fancy ...
- It must provide enought interactivity (e.g. click and inspect)
- Open user requirements (people will not tell you what they need and sometime they don't know what they need till the last minute)

Some of them are **structural**, some of them are related to **a good GUI**



- Our actual **GUI service**, althought fast, easy and fully integrated in our software, miss some of the structural requirements
- Start to use a new external program is **risky** when you are in an andvanced state with the infrastructure software (framework, montecarlo, datastores, geometry repository etc etc)
- Solution (partial): **commit to a protocol**, not to an application

Structure

So we started an experiment to use the **HepRep** protocol paradigm for GLAST event display, i.e. a client-server framework (Thursday talk will expand on this)

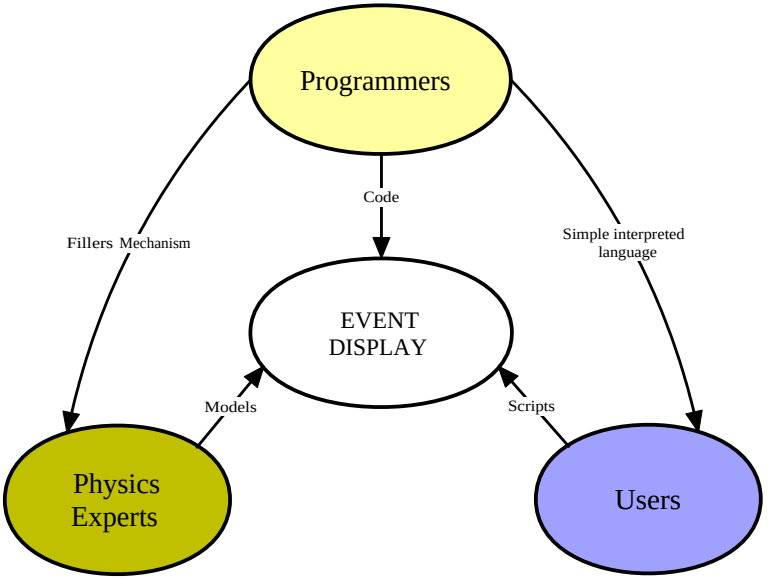
- The **server** side contains all the relevant **physics algorithms** specific of the experiment and all the data relative to the **event structure** and **geometric information**; in our case it lives inside **GAUDI**, the framework adopted by GLAST
- The **client** deals with all the graphics issues and the user interfaces

FRED is a new HepRep compatible graphics client.

There is a ready, nice and complete HepRep graphics client in Java, i.e. **WIRED**; so why another client?

- A new client for HepRep can help in **spreading the protocol**
- Programming our own client means we can have **full control** on functionalities to implement
- We want to promote a slightly **different approach** from WIRED

- We identify three main **actors** (roles) in an event display applications:
  - ⋆ **Programmers** that provide the main programs and all the infrastructure code
  - ⋆ **Physics experts** that decide what physical information augment the graphical representation of an event
  - ⋆ **End users** that need a fast and easy way to customize the application to their need



- GLAST HepRep server side provide a nice separation/integration between physics and graphics, enforcing in some way the first two roles.
- For the third one it is up to the graphics client to provide the end user with enought powerful tools (plugins mechanism, API etc etc)
- So we want FRED to be a graphics client that can be "programmed" with an **easy scripting language**

Our choices (see references for links)

⋆ **C++** is our choice language

⋆ **RUBY** is our scripting language

⋆ **FOX-Toolkit** is our GUI widgets library

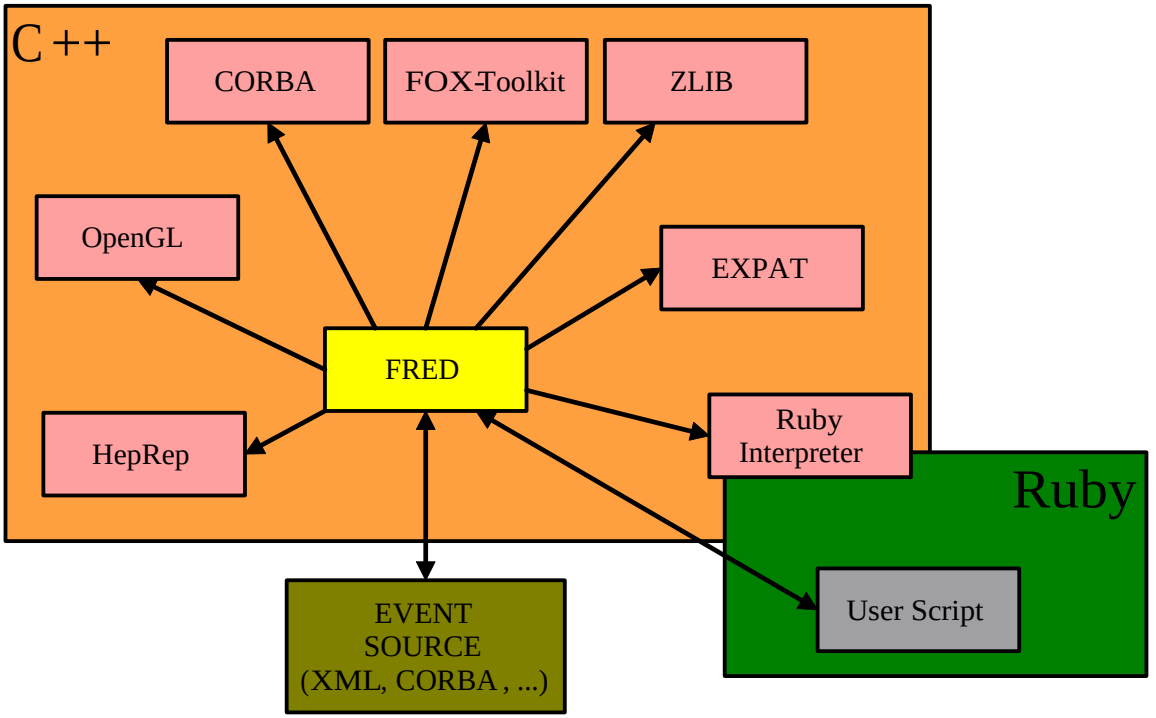
⋆ **XML** is our main persistency format (also compressed)

⋆ **CORBA** is our main middleware for server-client operations

⋆ **OpenGL** is our 2D/3D graphics library

XML and CORBA have been choosen for compatibility with WIRED (many more formats are possible)

The first version (September 2002) of FRED was a C++ application with a RUBY interpeter embedded.



This first design had some **limitations**

- No easy WAY to open the GUI API to users (it was possible, but very hard to do and not really stable)
- To add new features to the exposed users API we needed lots of wrapping work of legacy code

And it suffered also of some structural problems:

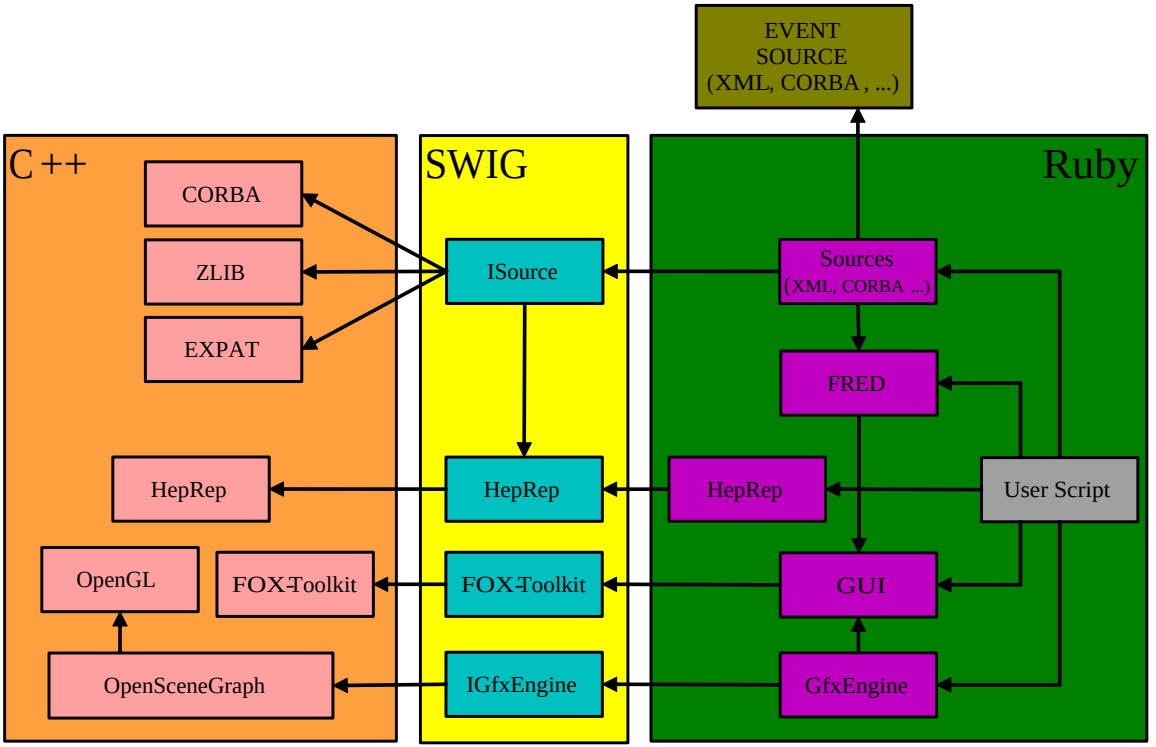
- Developing of new features was slow (most of the time waiting for compilation/linking just to find that a new smart idea was crap)
- The classes were not well structured; too much tight to libraries
  - For example the main FRED class was also the Main Window class; big mistake ...

But we were happy of some basic choices

- Compiled C++ code is fast
- Interpreted RUBY code is easy and helpful
- The FOX toolkit provides nice GUI
- CORBA can be a nightmare, but is really useful and fast (at least in our experience)
- OpenGL is fun (and fast); for now we are using the plain z-buffer approach, but more elaborate ones are possible

The idea is than to do all the heavy computational tasks with C++ code, and all the rest, comprising the **GUI**, with RUBY code.

So FRED is today a RUBY application that load some C++ compiled code (shareble libraries) and wrap interfaces to it



- Faster developing of core functionalities for us
  - I have an idea, I test it almost in real time
  - I add a new feature in one place, but I can easily move it in another place later
  - Easier to distribute small parts (ideas) between us
- Easier extendibility from the users side for all aspects, including GUI changes
- Same performances of the full C++ application
  - All input-output are in the C++ side
  - We don't wrap graphics operation: on the RUBY side we just open a canvas, all the OpenGL operations are performed in the C++ side.
  - The GUI is not a performance bottleneck (for normal users)

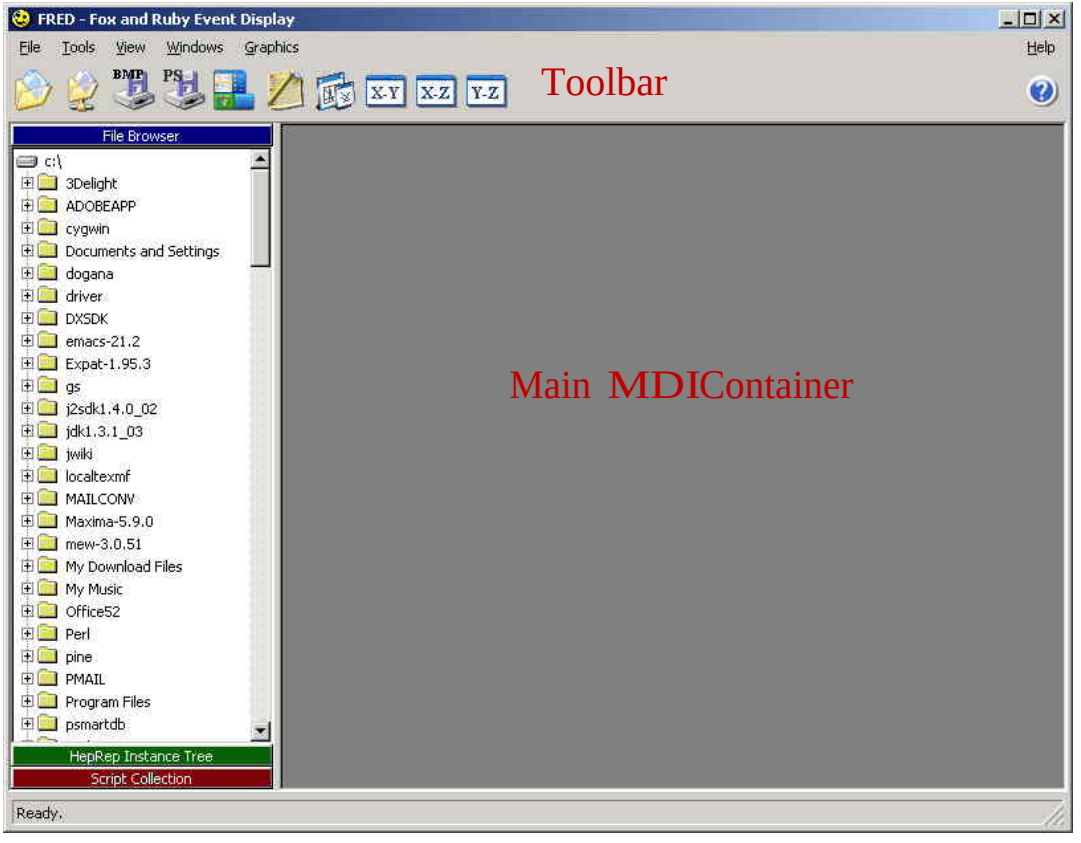
Features

- **Multiplatform**: tested and supported on Windows 2000/XP and Linux RH, but should work on many Unix flavours
- **HepRep Sources**: XML file (also compressed) by default, a CORBA connection with an optional plugin
- **Output**: both bitmap (for now BMP, soon PNG and JPG) and vectorial (PostScript).
- **Interaction**: usual zoom, pan, rotate operations via mouse and keys, plus selection via mouse of graphics objects and inspection of their attributes. Possible interaction with GAUDI in the future
- **Graphics**: using OpenGL, FRED automatically uses any available hardware acceleration; for simple events (like GLAST ones) and in wireframe mode performances are very good also in software mode.
- **Scripting**: script API gives access to all the main functionalities of FRED, comprising the possibility to add new widgets to the GUI. FRED has a simple internal editor for scripts and also a command line interpreter.

Examples

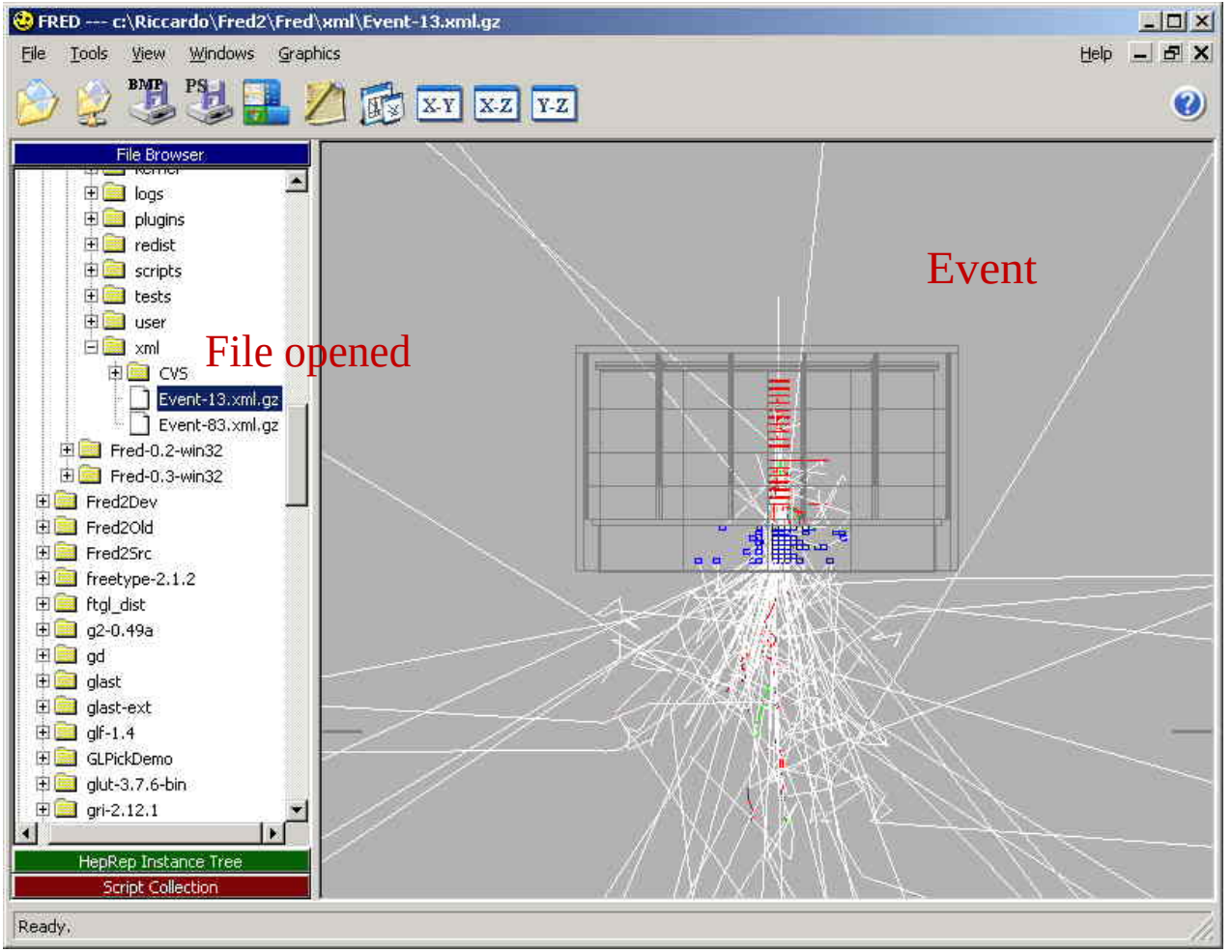
Menu

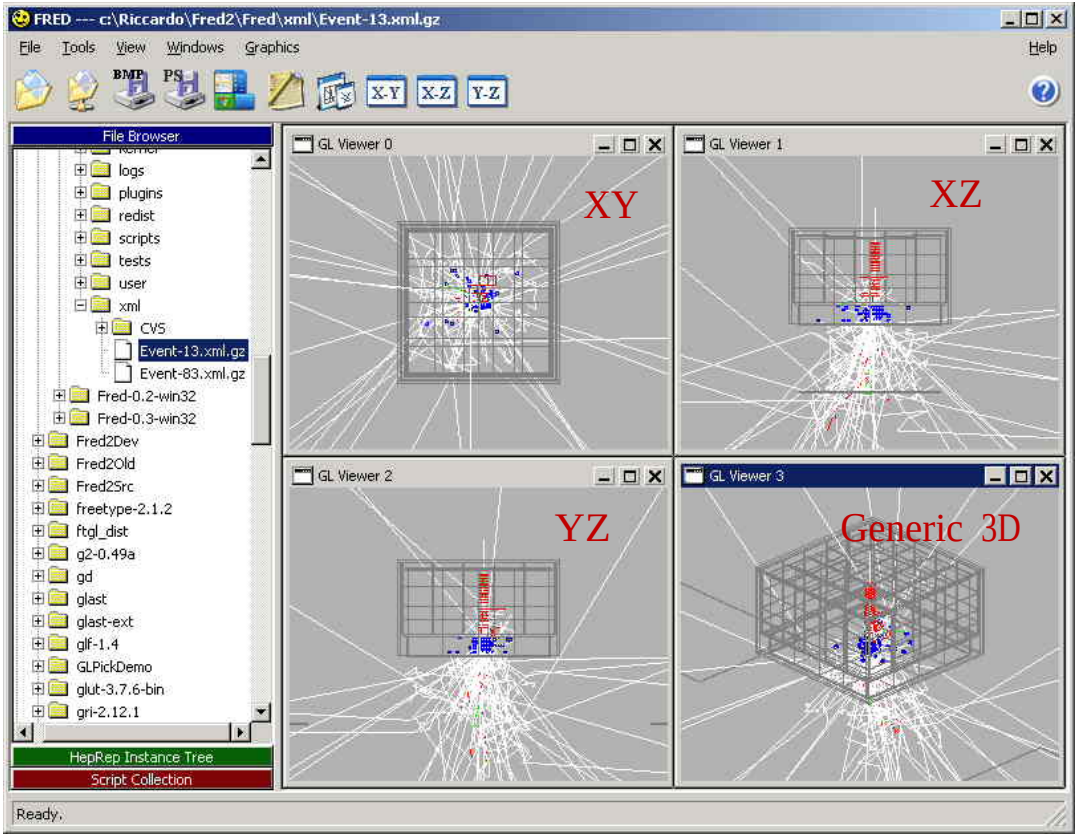
Left Slider



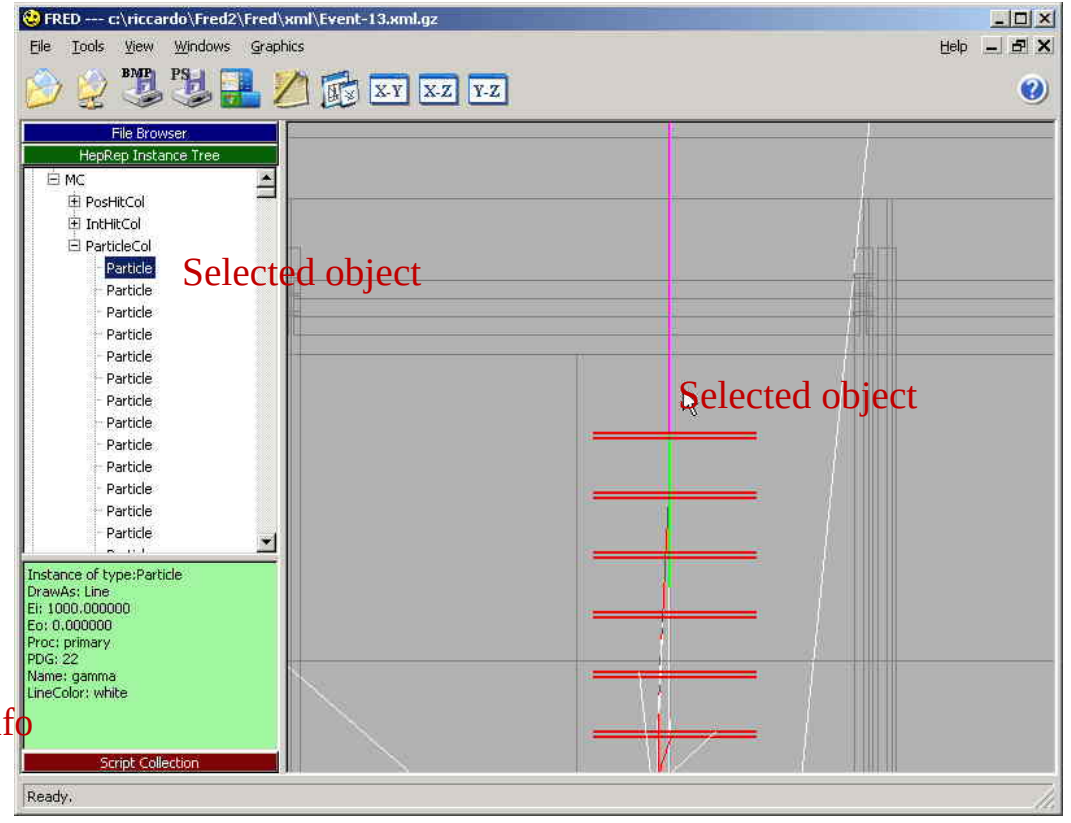
Toolbar

Status Bar

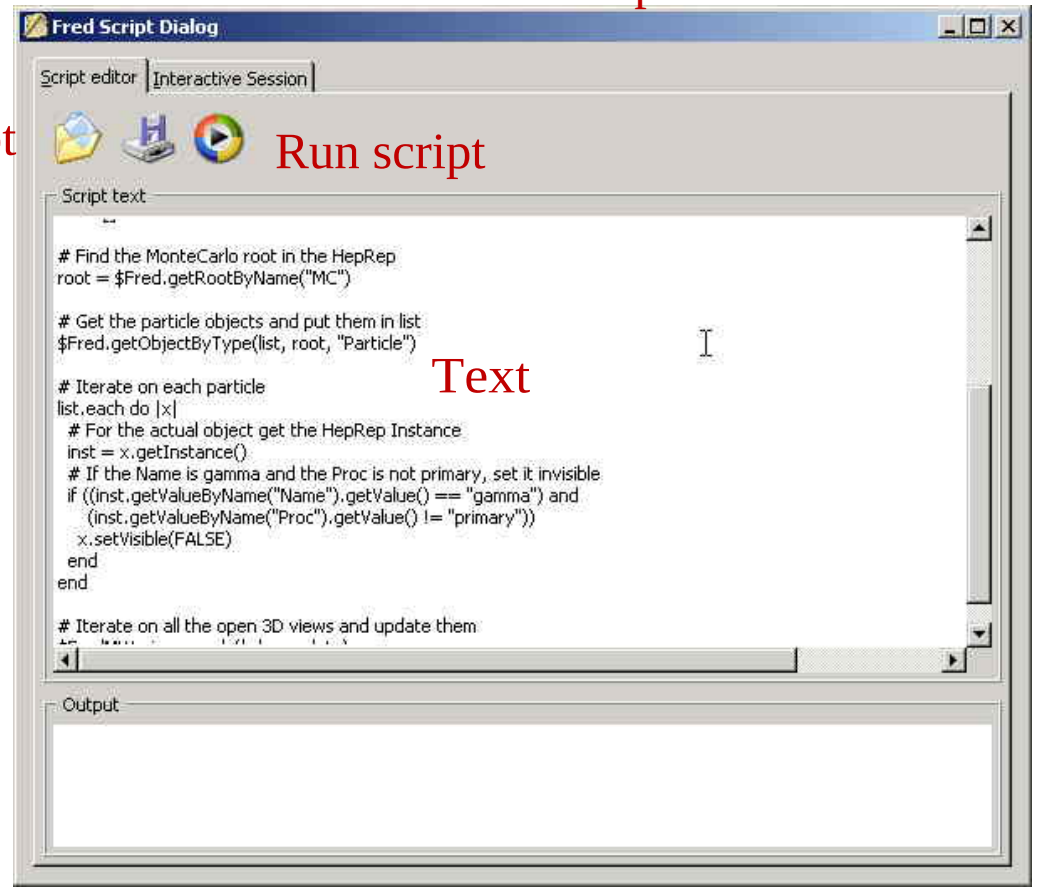




Multiple Windows



The Script Editor



Examples of script: hide gammas

```
# Define a list to fill with paticle objects
list = []
# Find the MonteCarlo root in the HepRep
root = $Fred.getRootByName("MC")
# Get the particle objects and put them in list
$Fred.getObjectByType(list, root, "Particle")
# Iterate on each particle
list.each do |x|
  # For the actual object get the HepRep Instance
  inst = x.getInstance()
  # If the Name is gamma and the Proc is not primary, hide
  if ((inst.getValueByName("Name").getValue() == "gamma") and
      (inst.getValueByName("Proc").getValue() != "primary"))
    x.setVisible(FALSE)
  end
end
# Iterate on all the open 3D views and update them
$FredMW.views.each{|x| x.update}
```

Examples of script: add menu voice

```
hide = proc{load "scripts/hidegamma.rb"}
$FredMW.addItemMenuProc("&Graphics", "Hide gammas", hide, TRUE)
```

Examples of script: change the background color of the first view

```
$FredMW.views[0].renderer.setBackground(0,0,0)
```

Outlooks

Some new features that are "coming soon"

- New HepRep sources, expecially an **HTTP** one for browsing events accessible from internet
- Use of the Ruby RMI mechanism, **druby**, for remote control of FRED from other applications
- Export various format for photorealistic rendering of the event (we are working to a **POV** one and to a **RenderMan** one)
- More than just wireframe graphics (filled volumes, semitransparent, layers, outlines etc etc).
- A "batch" mode to produce event images (vectorial or bitmap) without starting the GUI
- Options panel, with possibility to save preferred configuration from one session to the other
- More GLAST specific features (that will be collected in a single plugin)

Conclusions

So, is FRED a **complete ready-to-use product**? **Not yet** ...

- We need to extend the supported drawables shapes; we have just implemented the GLAST relevant ones (boxes, polylines, points)
- We need also to improve the HepRep compliancy
- Need some documentation
- Lots of fine tunings and tweaks

... **but**

- An early beta version for GLAST people has been released
- Already usable and quite stable and all the main functionalities are in place
- It is quite easy and fast for us to add new features, soon also for users
- We are ready for feedback

References

- **FRED** http://www.fisica.uniud.it/~riccardo/research/fred/
- **GLAST Software** http://www-glast.slac.stanford.edu/software
- **RUBY** http://www.ruby-lang.org/en
- **FOX-Toolkit** http://www.fox-toolkit.org
- **ACE-TAO (Corba)** http://www.cs.wustl.edu/~schmidt/TAO.html
- **HEPREP** http://www.heprep.freehep.org
- **WIRED** http://www.slac.stanford.edu/BFROOT/www/Computing/Graphics/Wired