Accessing the IRFs in C++ and Python

Jim Chiang
GSSC/UMBC/SLAC

irfInterface

- Recall

$$R_i(E', \hat{p}'; E, \hat{p}) = A_i(E, \hat{p}) P_i(\hat{p}'; E, \hat{p}) D_i(E'; E, \hat{p}) \tag{1}$$

- Abstract interface classes, `IAeff, IPsf, IEdisp`
  Concrete implementations accessed through `IrfsFactory`.

- Each has a basic interface,

```
IAeff::operator()(double energy,
                  const astro::SkyDir &srcDir,
                  const astro::SkyDir &scZAxis,
                  const astro::SkyDir &scXAxis) const;
IPsf::operator()(const astro::SkyDir &appDir,
                 double energy, const astro::SkyDir &srcDir,
                 const astro::SkyDir &scZAxis,
                 const astro::SkyDir &scXAxis) const;
IEdisp::operator()(const astro::SkyDir &appEnergy,
                   double energy, const astro::SkyDir &srcDir,
                   const astro::SkyDir &scZAxis,
                   const astro::SkyDir &scXAxis) const;
```

Accessing the IRFs

- Add this line to your `requirements` file:

```
use irfLoader v0* irfs
```

- In C++, do

```
#include "irfInterface/IrfsFactory.h"
#include "irfLoader/Loader.h"
<...>
irfLoader::Loader::go();
irfInterface::IrfsFactory * myFactory = irfInterface::IrfsFactory::instance();
irfInterface::Irfs * irfs = myFactory->create("DC1::Front");
irfInterface::IPsf & psf = *irfs->psf();
irfInterface::IAeff & aeff = *irfs->aeff();
irfInterface::IEdisp & edisp = *irfs->edisp();
astro::SkyDir srcDir;
<...>
// Coordinate system-independent interface
psf(appDir, energy, srcDir, scZAxis, scXAxis);
// In instrument coordinates
double separation = appDir.difference(srcDir)*180./M_PI;
double theta = scZAxis.difference(srcDir)*180./M_PI;
```

```
psf.value(separation, energy, theta, phi);
```

- In Python, do

```
import math
import irf_loader
irf_loader.Loader_go()
myFactory = irf_loader.IrfsFactory_instance()
irfs = myFactory.create("DC1::Front")
<...>
srcDir = irf_loader.SkyDir(0, 0)
<...>
psf = irfs.psf()
aeff = irfs.aeff()
edisp = irfs.edisp()
psf(appDir, energy, srcDir, scZAxis, scXAxis)
sep = appDir.difference(srcDir)*180./math.pi
<...>
psf.value(sep, energy, theta, phi)
```

Summary and random thoughts

- If CALDB is set up properly, the IRFs you get from running `irfLoader::Loader::go()` are retrieved from there, otherwise the data in `irfs/caldb` package are used.
- Implementation and event type combinations are identified entirely by the string passed to the `IrfsFactory::create` method, e.g., `myFactory->create(D̈C1::Front)̈`. Is this interface flexible enough if we have many event types? Dunno.
- IRF Visualization tool. Python `irf_loader` interface + a plotting module (pyROOT, HippoDraw, matplotlib, etc.) probably can provide the necessary bits.