{{lastupdated_at}} by {{lastupdated_by}}

Coding techniques

This pages summarizes some coding techniques that are used in GammaLib.

Interface classes

Interface classes are widely used throughout GammaLib. An interface class is an abstract virtual base class, i.e. all class that only has pure virtual methods. This forces the derives class to implement all the pure virtual methods.

Registries

Registries are widely used in GammaLib to collect the various derived classes that may exist for a base class. This allows automatic recognition of GammaLib of all derived classes that exist, and this even without recompilation of the code. One area where registries are widely used are model components. For example, by defining GModelSpectralRegistry class, all spectral models that are available through a registry, allowing for example automatic parsing of an XML file.

We illustrate the mechanism using the GModelSpectralRegistry class. Here an excerpt of the class definition:

```
class GModelSpectralRegistry {
public:
    GModelSpectralRegistry(const GModelSpectral* model);
    GModelSpectral* alloc(const std::string& type) const;
private:
    static int                    m_number;   //!< Number of models in registry
    static std::string*           m_names;    //!< Model names
    static const GModelSpectral** m_models;   //!< Pointer to seed models
};
```

All class members are static, so that every instance of the class gives access to the same data. The static variables are initialised in GModelSpectralRegistry.cpp:

```
int                    GModelSpectralRegistry::m_number(0);
std::string*           GModelSpectralRegistry::m_names(0);
const GModelSpectral** GModelSpectralRegistry::m_models(0);
```

m_number counts the number of derived classes that are registered, m_names points to a vector of string elements that contain the unique names of the derived classes, and m_models are pointers to instances of the different derived classes. These instances are also called the seeds.

The central method of the GModelSpectralRegistry class is the alloc() method. This method allocates a new instance of a derived class using the unique name of the class. For example, by specifying

```
GModelSpectralRegistry registry;
GModelSpectral*        spectral = registry("PowerLaw");
```

the pointer spectral will hold an instance of GModelSpectralPlaw. This is achieved by searching the array of m_names, and if a match is found, cloning the seed instance:

```
GModelSpectral* GModelSpectralRegistry::alloc(const std::string& type) const
{
    GModelSpectral* model = NULL;
    for (int i = 0; i < m_number; ++i) {
        if (m_names[i] == type) {
            model = m_models[i]->clone();
            break;
        }
    }
    return model;
}
```

If the requested model is not found, NULL is returned.

A spectral model is registered to the registry by adding two global variables at the top of the corresponding .cpp file. Here the example for GModelSpectralPlaw.cpp:

```
const GModelSpectralPlaw     g_spectral_plaw_seed;
const GModelSpectralRegistry g_spectral_plaw_registry(&g_spectral_plaw_seed);
```

The first line allocates one instance of the class, and the second line stores the address of this instance in the registry using a registry constructor.