GammaLib - Bug #367

Registries seem not to work when compiled statically on CentOS

07/24/2012 03:13 PM - Knödlseder Jürgen

Status:	In Progress	**Start date:**	07/24/2012
Priority:	Normal	**Due date:**	
Assigned To:		**% Done:**	10%
Category:		**Estimated time:**	0.00 hour
Target version:			

Description

When compiling GammaLib using the following configuration on CentOS

```
./configure CXXFLAGS="-static -fprofile-arcs -ftest-coverage -fPIC"
```

some of the unit tests fail as the registries do not contain all required components. Only the first component that had been registered seems to be visible.

History

#1 - 08/20/2012 12:19 AM - Knödlseder Jürgen

- Status changed from New to In Progress

- % Done changed from 0 to 10

Here some interesting links:

http://stackoverflow.com/questions/1804606/static-initialization-and-destruction-of-a-static-librarys-globals-not-happenin
.a static libraries contain several .o but they are not linked in unless you reference them from the main app.
.o files standalone link always.

So .o files in the linker always go inside, referenced or not, but from .a files only referenced .o object files are linked.

As a note, static global objects are not required to be initialized till you actually reference anything in the compilation unit, most compilers will initialize all of them before main, but the only requirement is that they get initialized before any function of the compilation unit gets executed.

Thanks. It seems that linking with all .o files contained in a .a can be forced by using the linker option -Wl,--whole-archive (or -Wl,-all_load on MacOSX)...

See also -force_load for MacOSX

http://www.gamedev.net/topic/622861-how-to-force-global-variable-which-define-in-a-static-library-to-initialize/
There's no official way to do this according to the standard -- it depends on implementation-defined behaviour (i.e. every compiler is allowed to do it differently).
This means that it's bad and wrong to rely on unreferenced globals in your static library to actually be constructed/linked...

To reliably force your global to be linked on any C++ compiler (i.e to do it the standard way), you've actually got to use the global inside your program, e.g.

```
//static_library.cpp
Reflection g_myGlobal( "foo", "bar" );
Reflection** GetReflectionTable()
{
  static Reflection* table = { &g_myGlobal, NULL };
  return table;
}

//main.cpp
int main()
{
  // this forces g_myGlobal to be linked, because it's actually used by the program now.
  for( Reflection** staticLibReflection = GetReflectionTable(); *staticLibReflection, ++staticLibReflection )
  {
        const Reflection& r = **staticLibReflection;
        r.foo();
  }
}
```