BornAgain - Testing #253

PythonAPI: learn how to create shared_ptr objects in python and deliver them into C++

03 Apr 2013 16:59 - pospelov

Status:	Archived	**Start date:**	03 Apr 2013
Priority:	Normal	**Due date:**	
Assignee:	pospelov	**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:	Sprint 12		

Description

History

#1 - 03 Apr 2013 17:00 - pospelov

- Status changed from Backlog to Sprint

#2 - 03 Apr 2013 17:00 - pospelov

- Target version set to Sprint 12

#3 - 03 Apr 2013 22:19 - pospelov

1) I was able to create toy example with shared pointer in python and pass it to C++.

How it works:
Suppose that you have in C++ class FormFactor and two Particles:

```
class Particle {
Particle(FormFactor *factor) : m_factor(factor){}
FormFactor *m_factor;
};
```

```
class ParticleWithShared {
ParticleWithShared(shared_ptr<FormFactor> factor) : m_factor(factor){}
shared_ptr<FormFactor> m_factor;
};
```

After exposing these objects you can do following in python:
Example1:

```
ff = FormFactor()
p1 = Particle(ff)
```

'ff' in this example is in fact some PyObject. By some boost::python magic it will be automatically converted into FormFactor* while passing in Particle(ff).
'ff' is watched by python garbage collector and will be deleted soon (together with underlying FormFactor *),
causing Particle(ff) crash. The problem is that the garbage collector doesn't consider Particle as another user
and doesn't increase usage counter. To avoid this one have to instruct boost about coupled time of life of Particle and FormFactor (which one would like to avoid),
or pass 'ff' into Particle by const reference to make copy. The last approach we use now.

Example2:

```
ff = FormFactor()
p1 = ParticleWithShared(ff)
```

'ff' is the same object as before, and by some boost::python magic it will be converted into shared_ptr which ParticleWithShared expects.

- 'ff' in python and ParticleWithShared::m_factor in C++ are still pointing to the same FormFactor* object.
  You can change it from both, python and C++, and see changes in both python and C++
- python garbage collector still doesn't know anything about extra user of 'ff' and 'ff' will die soon
- but when it dies, the underlying FormFactor* object remains unaffected and ParticleWithShared survives. How it works I really do not know.
  - does it mean that behavior of the destructor of 'ff' PyObject depends on the 'ff' usage history?
- shared pointer ParticleWithShared::m_factor doesn't know about extra user in python. Inside ParticleWithShared the counter of m_factor is 1.

One have to investigate possibility that FormFactor* will be deleted in C++ by shared pointer machinery while 'ff' object is still alive in python.
What will happen when 'ff' will go out of scope?

Example3:

```
ff = FormFactor()
p1 = Particle(ff)
p2 = ParticleWithShared(ff)
```

One could have both calls simultaneously. In this case Particle(ff) will not crash, when 'ff' goes out of scope, because 'ff' PyObject destructor knows somehow, that internals were passed also as shared pointer.

2) BornAgain with shared pointers still crashes "Fatal Python error: GC object already tracked"

#4 - 04 Apr 2013 16:42 - pospelov

I have changed comment above in the view of acquired knowledge.
Short conclusion: needs to investigate it further. Questions, what to check, are welcomed.

#5 - 06 Apr 2013 14:41 - pospelov

- Tracker changed from Feature to Testing

#6 - 06 Apr 2013 14:42 - pospelov

- Status changed from Sprint to Resolved

#7 - 11 Apr 2013 10:04 - herck

- Status changed from Resolved to Archived