# GammaLib - Bug #413

## Memory conflict when working with GLog clones

08/13/2012 01:10 AM - Knödlseder Jürgen

| | | | |
|---|---|---|---|
| **Status:** | New | **Start date:** | 08/13/2012 |
| **Priority:** | Normal | **Due date:** | |
| **Assigned To:** | | **% Done:** | 0% |
| **Category:** | | **Estimated time:** | 0.00 hour |
| **Target version:** | | | |

**Description**

The GLog class implements copying of objects, yet the class is not safe against memory corruption. If one creates a copy of a GLog object, and then one of the copies closes a file, the other copy will not be aware of the closing of the file.

One possible solution would be that the file is static, but then, any GLog copy that will become out of scope will close the file. This is not a desired property.

Better, any copy will not be allowed to close the file, and in addition, the file member is static. We thus make sure that only the master closes the file, and any copy can write to it as long as it is kept open. We can achieve this by adding a m_allow_close member that signals whether an object is allowed to close the file or not.

Note that this will make the GLog object memory save, but it should be questionned whether cloning of GLog for writing in the same file makes indeed sense. Maybe we should better not allow for cloning of GLog, or we should rethink of how this class should work. A clone could for example be just a buffer that is flushed into the original GLog object upon closing. In this case, the GLog object should have a member m_parent that points to a possible parent. If the pointer is NULL, the old workings will be obtained. Otherwise, instead of cloning the information is simply written into the parent. The problem here is that we don't really know whether the parent still exists at the moment of flushing.

Another approach would be the storing of the filename, and the opening of the file just at the time of writing. This would allow concurrent writing. Some tests need to be done to see whether this presents an important performance drawback.

### History

**#1 - 10/14/2012 03:05 PM - Deil Christoph**

Is it ever necessary to copy / clone a GLog object?

I think it is very common to have some objects that work with certain resources like files or databases that cannot be copied / cloned without inventing complicated mechanisms like the ones you describe above.

A standard solution is to simply not provide a copy constructor (by making it an empty private method) and not implement a clone method (or if a base class already has it, again make it an empty private method to effectively remove it).
The second standard C++ solution is boost::noncopyable, so in case you use GBase, maybe using an abstract base class GBaseNonCopyable is worth it if there are many non-copyable classes or to make it very explicit what the reason for this C++ "trick" with the private methods is, which is otherwise easily overlooked.