GammaLib - Change request #558

Gammalib CDC: Coding and Design Conventions

10/12/2012 01:09 PM - Deil Christoph

Status:	Closed	**Start date:**	10/12/2012
Priority:	Normal	**Due date:**	
Assigned To:	Knödlseder Jürgen	**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:	00-08-00		

Description

Feedback on the document

Here's my feedback on the "Gammalib CDC: Coding and Design Conventions" draft from 2012-09-21, located at doc/dev/coding/gammalib_coding.tex in the gammalib repo.

1. This is definitely a very useful document for a new gammalib developer like myself. Would it be possible to create html versions of this document and the other latex documents (like inst/gammalib_inst.tex and maths/gammalib_maths.tex) from the top-level "make doxygen"? This would be nice to always have easily accessible with the doxygen API class documentation while coding.
2. I think a high-level explanation of how the registries (GModelRadialRegistry, GModelRegistry, GModelSpatialRegistry, GModelSpectralRegistry, GModelTemporalRegistry, GObservationRegistry, GWcsRegistry) work should be added to this document or the Gammalib Software Design Description.
3. I think a rule "use C++ (std::string) instead of C-style (char*) strings" should be explicitly mentioned.
   This seems to be followed followed everywhere already, except why does GLog need a C-string interface in addition to the C++-string interface?
4. I'll try to make an Eclipse code format (one XML file) that matches the gammalib rules. Then developers don't have to think about code formatting, but can just press CMD+Shift+F before saving and checking in.
5. Why require a return statement in functions / methods returning void? This seems completely superfluous (i.e. a waste of screen real estate) to me.
6. The statements "Do not use C++ namespaces" and "Do not use C++ templates" should be qualified to gammalib, obviously std::vector is used a lot., which is a template in a namespace. I don't know if this is up for discussion, but I'd prefer if gammalib did use namespaces and templates. I'll try to make my case a section below.
7. Is it really necessary to include the license in each file? I don't like that whenever I open a file to read, all I see at first is the license and I have to scroll down to start reading. Many other projects just have one line per file "GPL 3 license, see LICENSE.txt", but I'm not even sure that is necessary.
8. If you look at the example hpp and cpp and python interface files for GClass, you'll see the description "My nice class" appear 8 times (In each file also in short form on top of the license). This is bad because it's easy to forget to edit some of the comments when copying and modifying old classes to make new ones. I'd prefer exactly one description, right in front of class GClass in the header, where doxygen will pick it up. Every C++ programmer knows to look at the corresponding header file for a given cpp or swig interface file. Also please mention that this class template is available at src/template/GClass.hpp and src/template/GClass.cpp and should be copied and edited when writing a new class instead of starting from scratch.
9. The GEbounds example violates the rule "Use std::vector instead of new / delete where possible", I think it should use std::vector.
10. For gammalib classes there are these recommendations "each C++ class should have the following private or protected methods for memory management: init_members(), copy_members(), free_members() and if new/delete is used alloc_members()" and "Every class should have at least a void constructor, a copy constructor, a destructor and an assignment operator" and then example implementations of these eight methods are given. Would it be worth to use an abstract base class for all gammalib classes (maybe called GClass or something else)? The advantage would be that the compiler helps the developer by enforcing these recommendations and maybe even save typing by giving the default implementations for the void constructor, copy constructor, destructor and assignment operator (I haven't thought about derived classes if there also default implementations would work). One disadvantage would be that if I want to take a gammalib class and use it in another project, I also have to copy GClass. I realize this would be a big change to gammalib, and Jürgen, if you consider it I think definitely a few C++ gurus should comment first.
11. At the end of 4.2.2 you show an example assignment operator for a derived class, then at the end of section 4.2.3 you show an example void constructor for a derived class. You should also show example copy constructor and assignment operator implementations for a derived class (and a multiple inheritance example if that exists) and put all four derived class examples in the same section (4.2.3).
12. Can you mention if gammalib ever use multiple inheritance or if it is allowed?
13. I generally like the balance between short and explicit class names in gammalib. There's a few cases where I think more explicit names would be better, because then I can guess from the name what they are and remember them better while coding: "Ptsrc" -> 'PointSource", "Plaw" -> "PowerLaw", "Ran" -> "Random", "Inst" -> "Instrument", "XmlPI" -> "XMLProcessingInstruction"
14. I did not understand (as a mediocre C++ programmer) the last two paragraphs in Section 4.2.4. Maybe giving an example would

help?

15. There's a few things in Section 4.2.5 I don't understand:
    1. Why does roi() and events() take and return pointers, but the other methods use references? You say the rule is "Pointers should only be used if NULL is a possible value", but why is NULL a possible value for an roi, but not a gti?
    2. Can you give an example that demonstrates the rule "never return base class objects by reference, as this will lead to code slicing if the method is used for object assignment"?
    3. I think in the examples of how method return values the comment should say "const class member reference" and not "const base class member reference" (otherwise, doesn't this example violate the rule just mentioned)?
    4. In the last paragraph you say "The ... methods do not return class members, hence they return an object by value". Why "hence", if that is a rule I would state it more prominently. Don't you have e.g. factory methods that create something and return a pointer and the user is responsible for deleting it?
16. For the container classes you have a "list of mandatory methods for container classes". Again, wouldn't it be better to have the compiler help the developer by enforcing this interface via n abstract base class? Naively I would think again (as for GClass, see above) that the advantages of using a GContainer base class would outweigh the disadvantages.
17. Why do you use 00-06-02 for the version number format? Almost all projects / package managers I know use 0.6.2.
18. "The definition of pure virtual methods should be done in a section that is separate from the methods that are implemented." -> where? At the beginning or end? Best include this in the example.
19. Write the python code example like this:

    ```
    >>> GClass c
    >>> print c
    ```

20. At the end of section 4.2.1 you write "Do always check ...", "Do never allocate ...", "Do always initialise ...", I would simply write "Always check ...", "Never allocate ...", "Always initialise ...".
21. At the beginning you mention "use std::vector or std::array containers". Actually std::array is never used in gammalib, so I would either just say "use std::vector as a container" or "use std::vector as a container and pre-allocate it's size when constructing it where possible for speed". I've never used std::array, so only using std::vector would (at least for me) simpler.
22. Functions (meaning standalone, not class member functions) are never mentioned. Do they exist? Are they discouraged for some reason?
23. Is this the right document to describe testing? I.e. that gammalib should be developed using test-driven development, i.e. each class should have C++ and python unit tests, how the test runner works locally and automatically on Jenkins.

In addition to this feedback on the CDC document I'd like to mention two important points where I would have preferred gammalib to be different.
Note that there is also a document "Gammalib SDD: Software Design Description" draft from 23 May 2010, located at doc/dev/sdd/gammalib_sdd.tex in the gammalib repo which is relevant to this discussion, but it doesn't seem to be maintained. It would be very good to have such a high-level document for new developers or users trying to learn the gammalib API.

General remarks on gammalib design decisions

Use namespaces in gammalib to be more structured / modular?

Currently gammalib is composed of these 14 modules that are at the end linked together into libgamma (alphabetical order): app, cta, fits, lat, linalg, model, mwl, numerics, obs, opt, sky, support, test, xml
These correspond to the folders src/app, inst/cta, src/fits, inst/lat, src/linalg, src/model, inst/mwl, src/numerics, src/obs, src/opt, src/sky, src/support, src/test, src/xml
However the header files (except for the inst = instrument classes) are all in one include folder, currently containing 127 header files.

As a new gammalib developer, I find it very difficult to get an overview of which of the 127 classes (will become more in the future) depend on each other. Having modules and a module diagram dependency graph ("depends on" meaning "has to link against") would help quite a bit to make the gammalib API more accessible, I think. Having such well defined module interfaces would probably also result in a simpler API where the different parts are less coupled (or maybe not, I can't tell what depends on what at the moment, I wouldn't be surprised if gammalib is already very well designed).

I also think it might be nice to use a global Gamma namespace instead of prepending every file and class with G. I think namespaces are one of the great advantages of C++ over C, because they make it explicit in library client code where a given symbol comes from.

Is there a technical reason (e.g. swig python interface or linker problems) why namespaces are not used?

Use templates in gammalib/fits to avoid code duplication?

For the FITS image and table classes we currently have 10x code duplication:

GFitsImage.hpp GFitsImageLong.hpp GFitsImageULong.hpp GFitsTableBoolCol.hpp GFitsTableCol.hpp GFitsTableLongLongCol.hpp GFitsTableUShortCol.hpp
GFitsImageByte.hpp GFitsImageLongLong.hpp GFitsImageUShort.hpp GFitsTableByteCol.hpp GFitsTableDoubleCol.hpp GFitsTableShortCol.hpp
GFitsImageDouble.hpp GFitsImageSByte.hpp GFitsTable.hpp GFitsTableCDoubleCol.hpp GFitsTableFloatCol.hpp GFitsTableStringCol.hpp
GFitsImageFloat.hpp GFitsImageShort.hpp GFitsTableBitCol.hpp GFitsTableCFloatCol.hpp GFitsTableLongCol.hpp GFitsTableULongCol.hpp

Wouldn't it make sense to use one C++ template class GFitsImage and GFitsTable instead?
Would that work?
What are the disadvantages?

There's a few other places where templates would help avoid code duplication.

Btw., how about add a rule: "Code duplication should be avoided by using composition, inheritance and templates."
(in case you hadn't noticed, I hate boilerplate and duplicated documentation / code.)

Subtasks:

Feature # 559: Create Eclipse C++ code formatting style for gammalib	**In Progress**

Related issues:

Related to GammaLib - Change request # 1061: Change isin, islong, etc. method...	**Closed**	**01/07/2014**

History

#1 - 10/12/2012 01:26 PM - Deil Christoph

- Description updated

#2 - 10/12/2012 04:22 PM - Knödlseder Jürgen

Christoph Deil wrote:

> # Feedback on the document
>
> Here's my feedback on the "Gammalib CDC: Coding and Design Conventions" draft from 2012-09-21, located at doc/dev/coding/gammalib_coding.tex in the gammalib repo.
>
> 1. This is definitely a very useful document for a new gammalib developer like myself. Would it be possible to create html versions of this document and the other latex documents (like inst/gammalib_inst.tex and maths/gammalib_maths.tex) from the top-level "make doxygen"? This would be nice to always have easily accessible with the doxygen API class documentation while coding.

Indeed, I have to see how I can automatically make the translation. I have already experimented with latex2html, and it seems to produce reasonable results.

> 1. I think a high-level explanation of how the registries (GModelRadialRegistry, GModelRegistry, GModelSpatialRegistry, GModelSpectralRegistry, GModelTemporalRegistry, GObservationRegistry, GWcsRegistry) work should be added to this document or the Gammalib Software Design Description.

It should probably not be in the coding and design conventions document, but somewhere else. I must admit that many of the other documents are not really up to date, and I should find time to bring them all up to the latest state. Explaining how registries work is indeed a fundamental issue, as they are now more and more used in GammaLib for extendability.

> 1. I think a rule "use C++ (std::string) instead of C-style (char*) strings" should be explicitly mentioned.

Agree.

> This seems to be followed followed everywhere already, except why does GLog need a C-string interface in addition to the C++-string interface?

This is more for historic reasons, as the C formatting is so much better than the C++ formatting. So I wanted to keep to possibility to use C formatting in GammaLib.

1. I'll try to make an Eclipse code format (one XML file) that matches the gammalib rules. Then developers don't have to think about code formatting, but can just press CMD+Shift+F before saving and checking in.

Great. I'm not really an Eclipse user, so I'm happy if someone more experienced is providing this format.

1. Why require a return statement in functions / methods returning void? This seems completely superfluous (i.e. a waste of screen real estate) to me.

The C++ documentation says "The return statement stops execution and returns to the calling function.", so this has nothing to do with having or having not something to return. I agree it's optional for void functions, but personally I preferred to always have a return statement for clarity. Probably I decided to use this rule also for compatibility assurance, but this is maybe not an issue.

1. The statements "Do not use C++ namespaces" and "Do not use C++ templates" should be qualified to gammalib, obviously std::vector is used a lot., which is a template in a namespace. I don't know if this is up for discussion, but I'd prefer if gammalib did use namespaces and templates. I'll try to make my case a section below.

I comment on this below. This is due to portability issues that existed when the GammaLib project was started.

1. Is it really necessary to include the license in each file? I don't like that whenever I open a file to read, all I see at first is the license and I have to scroll down to start reading. Many other projects just have one line per file "GPL 3 license, see LICENSE.txt", but I'm not even sure that is necessary.

This assures that the license information is also available when a file is separated from the rest of the package. I think there is no need to do so, but there is also nothing that speaks against is.

1. If you look at the example hpp and cpp and python interface files for GClass, you'll see the description "My nice class" appear 8 times (In each file also in short form on top of the license). This is bad because it's easy to forget to edit some of the comments when copying and modifying old classes to make new ones. I'd prefer exactly one description, right in front of class GClass in the header, where doxygen will pick it up. Every C++ programmer knows to look at the corresponding header file for a given cpp or swig interface file. Also please mention that this class template is available at src/template/GClass.hpp and src/template/GClass.cpp and should be copied and edited when writing a new class instead of starting from scratch.

I agree that you can always look up things, but I feel it's always better to have all relevant information in a single place. I should probably not have used "My nice class" so often in the example, as in practice, it's not the exactly the same information that is written in the files. It's just that Doxygen wants a file description and a class description. Furthermore, I like the idea that the first line of a file says what is in the file.

1. The GEbounds example violates the rule "Use std::vector instead of new / delete where possible", I think it should use std::vector.

Agree. There are a couple of cases where rules are violated. Some of them are justified, others are just old code pieces that need to be adjusted. I tried to write up the goal in the document, knowing that some code needs to be modified.

1. For gammalib classes there are these recommendations "each C++ class should have the following private or protected methods for memory management: init_members(), copy_members(), free_members() and if new/delete is used alloc_members()" and "Every class should have at least a void constructor, a copy constructor, a destructor and an assignment operator" and then example implementations of these eight methods are given. Would it be worth to use an abstract base class for all gammalib classes (maybe called GClass or something else)? The advantage would be that the compiler helps the developer by enforcing these recommendations and maybe even save typing by giving the default implementations for the void constructor, copy constructor, destructor and assignment operator (I haven't thought about derived classes if there also default implementations would work). One disadvantage would be that if I want to take a gammalib class and use it in another project, I also have to copy GClass. I realize this would be a big change to gammalib, and Jürgen, if you consider it I think definitely a few C++ gurus should comment first.

Indeed, this would be quite a change.

1. At the end of 4.2.2 you show an example assignment operator for a derived class, then at the end of section 4.2.3 you show an example void constructor for a derived class. You should also show example copy constructor and assignment operator implementations for a derived class (and a multiple inheritance example if that exists) and put all four derived class examples in the same section (4.2.3).

Agree.

1. Can you mention if gammalib ever use multiple inheritance or if it is allowed?

So far it is never used, but I see no reason to not allow this.

1. I generally like the balance between short and explicit class names in gammalib. There's a few cases where I think more explicit names would be better, because then I can guess from the name what they are and remember them better while coding: "Ptsrc" -> 'PointSource", "Plaw" -> "PowerLaw", "Ran" -> "Random", "Inst" -> "Instrument", "XmlPI" -> "XMLProcessingInstruction"

One of the reasons to have this short names is that code can fir more easily into a 80 characters window. Some of the names (such as Ptsrc and Plaw) are heritage from Fermi. Let's see what others think about the class names.

1. I did not understand (as a mediocre C++ programmer) the last two paragraphs in Section 4.2.4. Maybe giving an example would help?

Okay, I will add examples.

1. There's a few things in Section 4.2.5 I don't understand:
    1. Why does roi() and events() take and return pointers, but the other methods use references? You say the rule is "Pointers should only be used if NULL is a possible value", but why is NULL a possible value for an roi, but not a gti?

This is a tricky question and related to the problem of C++ code slicing: GGti is a simple C++ class, while GRoi and GEvent are abstract base classes, and here derived class objects are returned. This is what is ment by the bold statement **do never return base class objects by reference**.

1. Can you give an example that demonstrates the rule "never return base class objects by reference, as this will lead to code slicing if the method is used for object assignment"?

Okay, I will add such an example.

1. I think in the examples of how method return values the comment should say "const class member reference" and not "const base class member reference" (otherwise, doesn't this example violate the rule just mentioned)?

No, it does not violate the rule. The rule say that a base class **object** should never be returned by reference, but in the example, a reference to a base class **member** is returned.

1. In the last paragraph you say "The ... methods do not return class members, hence they return an object by value". Why "hence", if that is a rule I would state it more prominently. Don't you have e.g. factory methods that create something and return a pointer and the user is responsible for deleting it?

The "hence" means that they cannot return anything by reference. Only class members can be returned by reference, since their memory still exists after calling the method. Values that are computed by a method "on the fly" need be returned by value.

1. For the container classes you have a "list of mandatory methods for container classes". Again, wouldn't it be better to have the compiler

help the developer by enforcing this interface via n abstract base class? Naively I would think again (as for GClass, see above) that the advantages of using a GContainer base class would outweigh the disadvantages.

This is indeed an interesting suggestion. But the enforcing would then only work when you make sure that the container class is indeed derived from GContainer. So is there really some added value by having a common base class for a container? Are there cases were different types of containers can be used in an abstract way?

1. Why do you use 00-06-02 for the version number format? Almost all projects / package managers I know use 0.6.2.

I guess this is my Fermi history :-) But I have no strong opinion about this.

1. "The definition of pure virtual methods should be done in a section that is separate from the methods that are implemented." -> where? At the beginning or end? Best include this in the example.

I'll add an example.

1. Write the python code example like this:
   [...]

```
>>> GClass c
>>> print c
```

Okay

1. At the end of section 4.2.1 you write "Do always check ...", "Do never allocate ...", "Do always initialise ...", I would simply write "Always check ...", "Never allocate ...", "Always initialise ...".

Okay

1. At the beginning you mention "use std::vector or std::array containers". Actually std::array is never used in gammalib, so I would either just say "use std::vector as a container" or "use std::vector as a container and pre-allocate it's size when constructing it where possible for speed". I've never used std::array, so only using std::vector would (at least for me) simpler.

Agree

1. Functions (meaning standalone, not class member functions) are never mentioned. Do they exist? Are they discouraged for some reason?

They exist indeed, in GTools.cpp and now in the latest HESS branch also in GNumerics.cpp. In addition there are the friend functions used for printing and logging.

I have to think what kind of conventions should be mentioned for functions.

1. Is this the right document to describe testing? I.e. that gammalib should be developed using test-driven development, i.e. each class should have C++ and python unit tests, how the test runner works locally and automatically on Jenkins.

I guess a specific document should describe this, something like "GammaLib code development philosophy".

I'm wondering by the way to what extent all these things should go in specific documents, and to what extent we just should start filling up the Wiki

pages. One could also start with the Wiki pages, and when converged, turn this into a document.

> In addition to this feedback on the CDC document I'd like to mention two important points where I would have preferred gammalib to be different. Note that there is also a document "Gammalib SDD: Software Design Description" draft from 23 May 2010, located at doc/dev/sdd/gammalib_sdd.tex in the gammalib repo which is relevant to this discussion, but it doesn't seem to be maintained. It would be very good to have such a high-level document for new developers or users trying to learn the gammalib API.

General remarks on gammalib design decisions

Use namespaces in gammalib to be more structured / modular?

> Currently gammalib is composed of these 14 modules that are at the end linked together into libgamma (alphabetical order): app, cta, fits, lat, linalg, model, mwl, numerics, obs, opt, sky, support, test, xml
> These correspond to the folders src/app, inst/cta, src/fits, inst/lat, src/linalg, src/model, inst/mwl, src/numerics, src/obs, src/opt, src/sky, src/support, src/test, src/xml
> However the header files (except for the inst = instrument classes) are all in one include folder, currently containing 127 header files.
>
> As a new gammalib developer, I find it very difficult to get an overview of which of the 127 classes (will become more in the future) depend on each other. Having modules and a module diagram dependency graph ("depends on" meaning "has to link against") would help quite a bit to make the gammalib API more accessible, I think. Having such well defined module interfaces would probably also result in a simpler API where the different parts are less coupled (or maybe not, I can't tell what depends on what at the moment, I wouldn't be surprised if gammalib is already very well designed).
>
> I also think it might be nice to use a global Gamma namespace instead of prepending every file and class with G. I think namespaces are one of the great advantages of C++ over C, because they make it explicit in library client code where a given symbol comes from.
>
> Is there a technical reason (e.g. swig python interface or linker problems) why namespaces are not used?

The main reason is portability. I'm not sure that this is still an issue, but I think it was when I started the project (6 years ago). At least that is what I recall from studying various coding rules at the beginning of the development. Here a weblink:

- http://www.literateprogramming.com/portablecpp.pdf (rule 5)

I also could not really find the advantage, but see a major disadvantage for a programmer: you always have to find out what the appropriate namespace is. I preferred a flat system.

Use templates in gammalib/fits to avoid code duplication?

> For the FITS image and table classes we currently have 10x code duplication:
> [...]
>
> Wouldn't it make sense to use one C++ template class GFitsImage and GFitsTable instead?
> Would that work?
> What are the disadvantages?
>
> There's a few other places where templates would help avoid code duplication.
>
> Btw., how about add a rule: "Code duplication should be avoided by using composition, inheritance and templates."
> (in case you hadn't noticed, I hate boilerplate and duplicated documentation / code.)

This is the same argument as above: when I started to look into coding rules for GammaLib, I have read several warnings about using templates, and people arguing against their usage. Here a few sites that I found from googling:

- http://www.literateprogramming.com/portablecpp.pdf (see rule 1)
- http://people.cs.uchicago.edu/~jacobm/pubs/templates.html

I agree, it's a question of taste, but as I wanted to have maximum portability, I decided not to use templates in GammaLib.

#3 - 10/12/2012 05:36 PM - Deil Christoph

Jürgen Knödlseder wrote:

> Christoph Deil wrote:
>
> 1. Can you mention if gammalib ever use multiple inheritance or if it is allowed?
>
> So far it is never used, but I see no reason to not allow this.

Multiple inheritance can make things very complicated.
I'd also expect problems because there might be differences how multiple inheritance works in C++ and python.

I like that it's not used at the moment, so how about simply adding this statement:
"Multiple inheritance is not used at the moment in gammalib. Because of the added complexity of multiple inheritance in C++ in python there would have to be very good reasons to use it in gammalib." and then also remind the reader of this point in Section 4.2.3 (there I was thinking: how would this work for multiple inheritance and my head started to hurt).

> 1. There's a few things in Section 4.2.5 I don't understand:
>   1. Why does roi() and events() take and return pointers, but the other methods use references? You say the rule is "Pointers should only be used if NULL is a possible value", but why is NULL a possible value for an roi, but not a gti?
>
> This is a tricky question and related to the problem of C++ code slicing: GGti is a simple C++ class, while GRoi and GEvent are abstract base classes, and here derived class objects are returned. This is what is ment by the bold statement **do never return base class objects by reference**.

Maybe mention this fact which classes are abstract base classes and which are concrete earlier in Section 4.2.5. I believe the first time it is mentioned is in the code comments for the return value examples, but I was already wondering why GGti and GRoi are passed as pointers in the method parameter examples.

> 1. For the container classes you have a "list of mandatory methods for container classes". Again, wouldn't it be better to have the compiler help the developer by enforcing this interface via n abstract base class? Naively I would think again (as for GClass, see above) that the advantages of using a GContainer base class would outweigh the disadvantages.
>
> This is indeed an interesting suggestion. But the enforcing would then only work when you make sure that the container class is indeed derived from GContainer. So is there really some added value by having a common base class for a container? Are there cases were different types of containers can be used in an abstract way?

Or maybe some of the methods could have default implementations in the base class?
I haven't looked, so I don't know if the savings are significant enough to make the change.

> 1. Functions (meaning standalone, not class member functions) are never mentioned. Do they exist? Are they discouraged for some reason?
>
> They exist indeed, in GTools.cpp and now in the latest HESS branch also in GNumerics.cpp. In addition there are the friend functions used for printing and logging.
>
> I have to think what kind of conventions should be mentioned for functions.

I think there will be cases where we have groups of related functions. In the HESS software this is sometimes grouped in a namespace (which I'd prefer), sometimes a class with only static methods is used:

```
namespace Utilities {

 class Statistics {
 public:
   virtual ~Statistics() {}
   static double Alpha( double exposure_on, double exposure_off);
   static double LiMa(int non, int noff, double alpha = 1.0);
   static double Significance(int non, int noff, double alpha = 1.0);
   static double SafeSignificance(int non, int noff, double alpha = 1.0);
   static double LiMa_LogLikelihood(int non, int noff, double alpha,double signal);
   static double LiMa_dExcess_Up(int non, int noff, double alpha = 1.0,double nsig = 1.0);
   static double LiMa_dExcess_Down(int non, int noff, double alpha = 1.0,double nsig = 1.0);
   static double LiMa_dExcess(int non, int noff, double alpha = 1.0,double nsig = 1.0);
   static double LiMa_requiredExcess(double significance, int noff, double alpha = 1.0);
...
```

1. Is this the right document to describe testing? I.e. that gammalib should be developed using test-driven development, i.e. each class should have C++ and python unit tests, how the test runner works locally and automatically on Jenkins.

> I guess a specific document should describe this, something like "GammaLib code development philosophy".

Yes, but the most important point for new developers are the practical things:

- How do I run all tests, how do I run subsets of tests that cover what I'm working on?
- When do I have to type "./autogen.sh", when "make install", when "reload gammalib" or start a new python interpreter in my edit - compile - test cycle?
- Do I have to type "make clean" or "./autogen.sh" after switching my git branch?
- How to I debug and profile C++ and python (this is probably too extensive, but links to good references would be useful).

You have figured this all out, but for developers that are new to git, auto tools, swig, or maybe even C++ or python, having some documentation on these things would be really helpful to get started.

> I'm wondering by the way to what extent all these things should go in specific documents, and to what extent we just should start filling up the Wiki pages. One could also start with the Wiki pages, and when converged, turn this into a document.

I think the best place would be with the code, so that it is always up to date and available. Maybe restructured text or markdown (instead of latex) converted to html would be convenient? doxygen is the best tool for C++, swig for python, for a mixed project like gammalib I'm not sure. In the end it's not so important as there are converter scripts that work pretty well for all the various markup languages.

About the question on namespaces and templates:

I agree that this is a matter of taste and that being very conservative in using C++ features is extremely important to get a portable and maintainable library.

But I would use namespaces at least to group functions (see example with statistics functions above) and I would use templates in simple cases where you want to use the same function or class for different data types (i.e. as a template :-) as e.g. for the FITSImage and FITSTable classes.

I think you'd have a hard time nowadays finding a system that doesn't have a C++ compiler that doesn't support namespaces and templates. The references you gave were from 1998 and 2003.

#4 - 10/12/2012 06:31 PM - Knödlseder Jürgen

Christoph Deil wrote:

> Jürgen Knödlseder wrote:
>
> > Christoph Deil wrote:
> >
> > 1. Can you mention if gammalib ever use multiple inheritance or if it is allowed?
> >
> > So far it is never used, but I see no reason to not allow this.
>
> Multiple inheritance can make things very complicated.
> I'd also expect problems because there might be differences how multiple inheritance works in C++ and python.
>
> I like that it's not used at the moment, so how about simply adding this statement:
> "Multiple inheritance is not used at the moment in gammalib. Because of the added complexity of multiple inheritance in C++ in python there would have to be very good reasons to use it in gammalib." and then also remind the reader of this point in Section 4.2.3 (there I was thinking: how would this work for multiple inheritance and my head started to hurt).

Agree.

> 1. There's a few things in Section 4.2.5 I don't understand:
>   1. Why does roi() and events() take and return pointers, but the other methods use references? You say the rule is "Pointers should only be used if NULL is a possible value", but why is NULL a possible value for an roi, but not a gti?
>
> This is a tricky question and related to the problem of C++ code slicing: GGti is a simple C++ class, while GRoi and GEvent are abstract base classes, and here derived class objects are returned. This is what is ment by the bold statement **do never return base class objects by reference**.
>
> Maybe mention this fact which classes are abstract base classes and which are concrete earlier in Section 4.2.5. I believe the first time it is mentioned is in the code comments for the return value examples, but I was already wondering why GGti and GRoi are passed as pointers in the method parameter examples.

I added an example to illustrate what I mean.

> 1. For the container classes you have a "list of mandatory methods for container classes". Again, wouldn't it be better to have the compiler help the developer by enforcing this interface via n abstract base class? Naively I would think again (as for GClass, see above) that the advantages of using a GContainer base class would outweigh the disadvantages.
>
> This is indeed an interesting suggestion. But the enforcing would then only work when you make sure that the container class is indeed derived from GContainer. So is there really some added value by having a common base class for a container? Are there cases were different types of containers can be used in an abstract way?
>
> Or maybe some of the methods could have default implementations in the base class?
> I haven't looked, so I don't know if the savings are significant enough to make the change.

I think savings could only be obtained here when template classes were used, as the containers contain different types of objects.

1. Functions (meaning standalone, not class member functions) are never mentioned. Do they exist? Are they discouraged for some reason?

> They exist indeed, in GTools.cpp and now in the latest HESS branch also in GNumerics.cpp. In addition there are the friend functions used for printing and logging.
>
> I have to think what kind of conventions should be mentioned for functions.

> I think there will be cases where we have groups of related functions. In the HESS software this is sometimes grouped in a namespace (which I'd prefer), sometimes a class with only static methods is used:
>
> [...]

I agree that having a namespace for the functions would be good, so that we can be sure that no conflict arises with functions. Defining them in a gammalib namespace is probably a good suggestion.

Gathering functions in a class seems quite weird to me. One can of course do this, but somehow it's not really what you would use a class for.

I'm still reluctant to introduce various namespaces within GammaLib (except of a gammalib namespace for functions), because at the end it just will produce a hurdle for the developer to figure out in which namespace a class lives. And if one includes all namespaces by default in a development, I can't see the point of using them.

Is there any other argument for namespaces as the conflict with existing names? I think the GammaLib classes, with the standard naming scheme starting with GXxxx should largely protect for this. But as said before, for the functions I can see the usefulness.

1. Is this the right document to describe testing? I.e. that gammalib should be developed using test-driven development, i.e. each class should have C++ and python unit tests, how the test runner works locally and automatically on Jenkins.

> I guess a specific document should describe this, something like "GammaLib code development philosophy".

> Yes, but the most important point for new developers are the practical things:
>
> - How do I run all tests, how do I run subsets of tests that cover what I'm working on?
> - When do I have to type "./autogen.sh", when "make install", when "reload gammalib" or start a new python interpreter in my edit - compile - test cycle?
> - Do I have to type "make clean" or "./autogen.sh" after switching my git branch?
> - How to I debug and profile C++ and python (this is probably too extensive, but links to good references would be useful).
> You have figured this all out, but for developers that are new to git, auto tools, swig, or maybe even C++ or python, having some documentation on these things would be really helpful to get started.

Maybe we can start gathering these things in the Wiki ... and then turn those later in a document. I started to modify [[Contributing to GammaLib]] to make this a central place for developer tips. Please check out this page and comment on information that is missing. Also, don't hesitate to add information to this page.

> I'm wondering by the way to what extent all these things should go in specific documents, and to what extent we just should start filling up the Wiki pages. One could also start with the Wiki pages, and when converged, turn this into a document.

> I think the best place would be with the code, so that it is always up to date and available. Maybe restructured text or markdown (instead of latex) converted to html would be convenient? doxygen is the best tool for C++, swig for python, for a mixed project like gammalib I'm not sure. In the end it's not so important as there are converter scripts that work pretty well for all the various markup languages.

Agree. I have not yet looked how documentation can be directly included in Doxygen. This is certainly a good way to deal with this. But I also like the "standalone" .tex files, as they can live independently of a large documentation system.

About the question on namespaces and templates:

I agree that this is a matter of taste and that being very conservative in using C++ features is extremely important to get a portable and maintainable library.

But I would use namespaces at least to group functions (see example with statistics functions above) and I would use templates in simple cases where you want to use the same function or class for different data types (i.e. as a template :-) as e.g. for the FITSImage and FITSTable classes.

I think you'd have a hard time nowadays finding a system that doesn't have a C++ compiler that doesn't support namespaces and templates. The references you gave were from 1998 and 2003.

Agree, they were old. But the project also starts to get old :-)

By the way: there is one point in the FITS classes where templates can't be used, as the FITS data types are encoded by a numeric value, and this numeric value is needed in the code.

#5 - 10/12/2012 06:33 PM - Knödlseder Jürgen

- Status changed from New to In Progress

- Assigned To set to Knödlseder Jürgen

Updated the CDC document.

#6 - 10/13/2012 04:23 PM - Knödlseder Jürgen

Christoph Deil wrote:

1. For gammalib classes there are these recommendations "each C++ class should have the following private or protected methods for memory management: init_members(), copy_members(), free_members() and if new/delete is used alloc_members()" and "Every class should have at least a void constructor, a copy constructor, a destructor and an assignment operator" and then example implementations of these eight methods are given. Would it be worth to use an abstract base class for all gammalib classes (maybe called GClass or something else)? The advantage would be that the compiler helps the developer by enforcing these recommendations and maybe even save typing by giving the default implementations for the void constructor, copy constructor, destructor and assignment operator (I haven't thought about derived classes if there also default implementations would work). One disadvantage would be that if I want to take a gammalib class and use it in another project, I also have to copy GClass. I realize this would be a big change to gammalib, and Jürgen, if you consider it I think definitely a few C++ gurus should comment first.

I start to like this idea. I think it's generally called adding an interface class. I started to draft such an interface class (see [[GBase]]) and started also to play with it in the definition of the GammaLib classes. I'm not sure that it's such a big issue, because basically only the header files need to be updated. I still have to see if it has an impact on the Python interface. It will at least help to gather all classes that do not follow the interface.

I think I'll do the same thing for the container ([[GContainer]]) and the elements of a container ([[GElement]]).

#7 - 10/13/2012 08:32 PM - Deil Christoph

Jürgen Knödlseder wrote:

> By the way: there is one point in the FITS classes where templates can't be used, as the FITS data types are encoded by a numeric value, and this numeric value is needed in the code.

Do you mean this (for the example of GFitsImageFloat)?

```
// in GFitsCfitsio.hpp
#define __TFLOAT      TFLOAT
#define __TFLOAT      42

-----------------------------------------------
// in GFitsImageFloat.cpp
/*****************************************************************************//**
 * @brief Return image type
 *****************************************************************************/
int GFitsImageFloat::type(void) const
{
    // Return type
    return __TFLOAT;
}
```

I didn't look yet at all what this is used for, but as far as I can see CCfits handles this via an enum in CCfits.h and then avoids all the class / code duplication you have for the 10 data types by using templates.

Actually I think including CCfits in gammalib would have a lot of advantages :-)

- It's **the** C++ CFITSIO library (the only other I could find, AIPS++ FITS library, wasn't updated since 1997). Most developers will work on other C++ projects using FITS besides gammalib in their career, for them learning the CCfits API is more useful than learning the GFits... API.
- CCfits is battle-tested and written / maintained by NASA, allowing us to focus on gamma-ray data analysis
- CCfits should be easy to include / build as an external package in gammalib (to avoid users having to install it separately as a dependency)? Possible showstoppers:
- CCfits license compatible with gammalib?
- Swig python wrapping works easily?

I know we have discussed this in the past, and you said you'd rather have gammalib be very independent, but I thought I'd give it another shot.
Feel free to ignore me here, this is not really the right place for this discussion.
Note that if you consider using CCfits here I will most likely make similar arguments for WCS and XML in the future. :-)

#8 - 10/14/2012 12:07 AM - Knödlseder Jürgen

Christoph Deil wrote:

> Jürgen Knödlseder wrote:
>
> > By the way: there is one point in the FITS classes where templates can't be used, as the FITS data types are encoded by a numeric value, and this numeric value is needed in the code.
>
> Do you mean this (for the example of GFitsImageFloat)?
> [...]
>
> I didn't look yet at all what this is used for, but as far as I can see CCfits handles this via an enum in CCfits.h and then avoids all the class / code duplication you have for the 10 data types by using templates.

This is only part of the problem. Another point is the conversion of FITS types into C++ data types, which is not a 1:1 conversion. For example, how to handle bit columns? I'm not sure how CCfits is doing this, but a template is always reductive. A template is useful if you deal with different C++ data types, but here we have storage classes, and they don't map 100%.

> Actually I think including CCfits in gammalib would have a lot of advantages :-)
>
> - It's **the** C++ CFITSIO library (the only other I could find, AIPS++ FITS library, wasn't updated since 1997). Most developers will work on other C++ projects using FITS besides gammalib in their career, for them learning the CCfits API is more useful than learning the GFits... API.
> - CCfits is battle-tested and written / maintained by NASA, allowing us to focus on gamma-ray data analysis
> - CCfits should be easy to include / build as an external package in gammalib (to avoid users having to install it separately as a dependency)?
>   Possible showstoppers:
> - CCfits license compatible with gammalib?
> - Swig python wrapping works easily?
>
> I know we have discussed this in the past, and you said you'd rather have gammalib be very independent, but I thought I'd give it another shot. Feel free to ignore me here, this is not really the right place for this discussion.
> Note that if you consider using CCfits here I will most likely make similar arguments for WCS and XML in the future. :-)

I'm not sure how battle tested all these things indeed are. For example, I discovered a bug in WCS that was fixed later in wcslib. Generally, these are all libraries that do many things that are not needed by GammaLib, and GammaLib focuses on doing things it needs well. For example, the FITS classes only read data when they are requested. I'm not sure the CCfits has such a capability. The WCS library does not provide support for HealPix maps, GammaLib does. Most XML libraries are extremely complex, the GammaLib is very lightweight and perfectly does the job.

Experience just tells that it's always complicated if you depend on other systems, where you have no lever on maintenance, etc. The whole point for writing GammaLib was to have a system that is not just the sum of N external libraries, but to have a system where we have 100% control over maintenance and evolution.

#9 - 01/04/2014 12:46 PM - Deil Christoph

I just re-read the coding conventions at http://gammalib.sourceforge.net/coding/index.html and will make a pull request with (hopefully) some improvements shortly.

I think it would be best to close this issue soon and to create separate issues for individual changes to the CDC.

One more suggestion though:

- I would find is_long, is_in and has_edisp, separating words by underscores, more readable than the current islong, isin and hasedisp style.
  I can make the change globally quickly if you agree, though I'm also happy keeping it the way it is ... even the C++ language is inconsistent on this (static_cast, but nullptr, see http://en.cppreference.com/w/cpp/keyword).

One question:

- "De-allocate all memory that is not de-allocated by the destructor before throwing an exception." -> Does this case exist? Shouldn't the destructor always de-allocate all memory?

#10 - 01/04/2014 03:44 PM - Deil Christoph

- Status changed from In Progress to Pull request

Here are some updates to the CDC (and an old commit of minor updates to the Getting started I accidentally had in this branch):
https://github.com/cdeil/gammalib/compare/sphinx_docs_cdc_update

I'm afraid the diff is completely useless because the files contain Windows line-end characters which are not recognised by default by git-diff or github and thus the files are seen to only have a single line.
I'll make a separate issue for that ... this should help to get a meaningful diff locally:
http://stackoverflow.com/questions/1889559/git-diff-to-ignore-m

The only substantial change that I think needs review is this commit:

```
commit db7c08094cea0f7754c17e3662aa591a6402290b
Author: Christoph Deil <Deil.Christoph@gmail.com>
Date:   Sat Jan 4 15:16:00 2014 +0100

    Extend C++ and Python coding rules.

    - Add more information
    - State that GammaLib uses C++98, not C++11
    - Change statement on supported Python versions
      (was 2.4 upwards, now 2.6 upwards)
```

https://github.com/cdeil/gammalib/blob/db7c08094cea0f7754c17e3662aa591a6402290b/doc/source/coding/rules.rst

#11 - 01/04/2014 08:11 PM - Deil Christoph

OpenMP should at least be briefly mentioned in the CDC.

#12 - 01/07/2014 03:06 PM - Knödlseder Jürgen

Deil Christoph wrote:

> I just re-read the coding conventions at http://gammalib.sourceforge.net/coding/index.html and will make a pull request with (hopefully) some improvements shortly.
>
> I think it would be best to close this issue soon and to create separate issues for individual changes to the CDC.
>
> One more suggestion though:
>
> - I would find is_long, is_in and has_edisp, separating words by underscores, more readable than the current islong, isin and hasedisp style. I can make the change globally quickly if you agree, though I'm also happy keeping it the way it is ... even the C++ language is inconsistent on this (static_cast, but nullptr, see http://en.cppreference.com/w/cpp/keyword).

I'm hesitating. isin is in fact not terrible, as it would be mixed up with some complex sine function. I won't be opposed to change the methods as you propose (i.e. adding an underscore). If you want to do this, just create a branch and make the changes. I just created an issue #1061 to track this change.

> One question:
>
> - "De-allocate all memory that is not de-allocated by the destructor before throwing an exception." -> Does this case exist? Shouldn't the destructor always de-allocate all memory?

This refers in fact to temporary memory. There are methods that allocate some temporary memory, and one has to make sure that this temporary memory is de-allocated before throwing an exception.

#13 - 01/07/2014 03:07 PM - Knödlseder Jürgen

- Status changed from Pull request to Closed

- Target version set to 00-08-00

#14 - 01/07/2014 07:56 PM - Deil Christoph

Knödlseder Jürgen wrote:

> Deil Christoph wrote:
>
> > One question:
> >
> > - "De-allocate all memory that is not de-allocated by the destructor before throwing an exception." -> Does this case exist? Shouldn't the destructor always de-allocate all memory?
>
> This refers in fact to temporary memory. There are methods that allocate some temporary memory, and one has to make sure that this temporary memory is de-allocated before throwing an exception.

Can you point to an example?

Are those uses of new really necessary or should they just be replaced with the use of a std::vector, making the code simpler and this rule superfluous?

#15 - 01/07/2014 08:42 PM - Knödlseder Jürgen

GSparesNumeric and GSparseSymbolic are full of temporary memory allocations. You'll also find in GMatrixSparse. Also GObservation::likelihood_poisson_unbinned etc. use temporary memory.

I agree that std::vector could be used in many cases, and the code may eventually be changed in the future in that respect. Yet there are cases where std::vector is not optimal:
http://assoc.tumblr.com/post/411601680/performance-of-stl-vector-vs-plain-c-arrays
http://stackoverflow.com/questions/3664272/stdvector-is-so-much-slower-than-plain-arrays

I see no harm to have such a rule, it just recalls that when dynamic memory is used, it has to be de-allocated properly in case of exceptions.