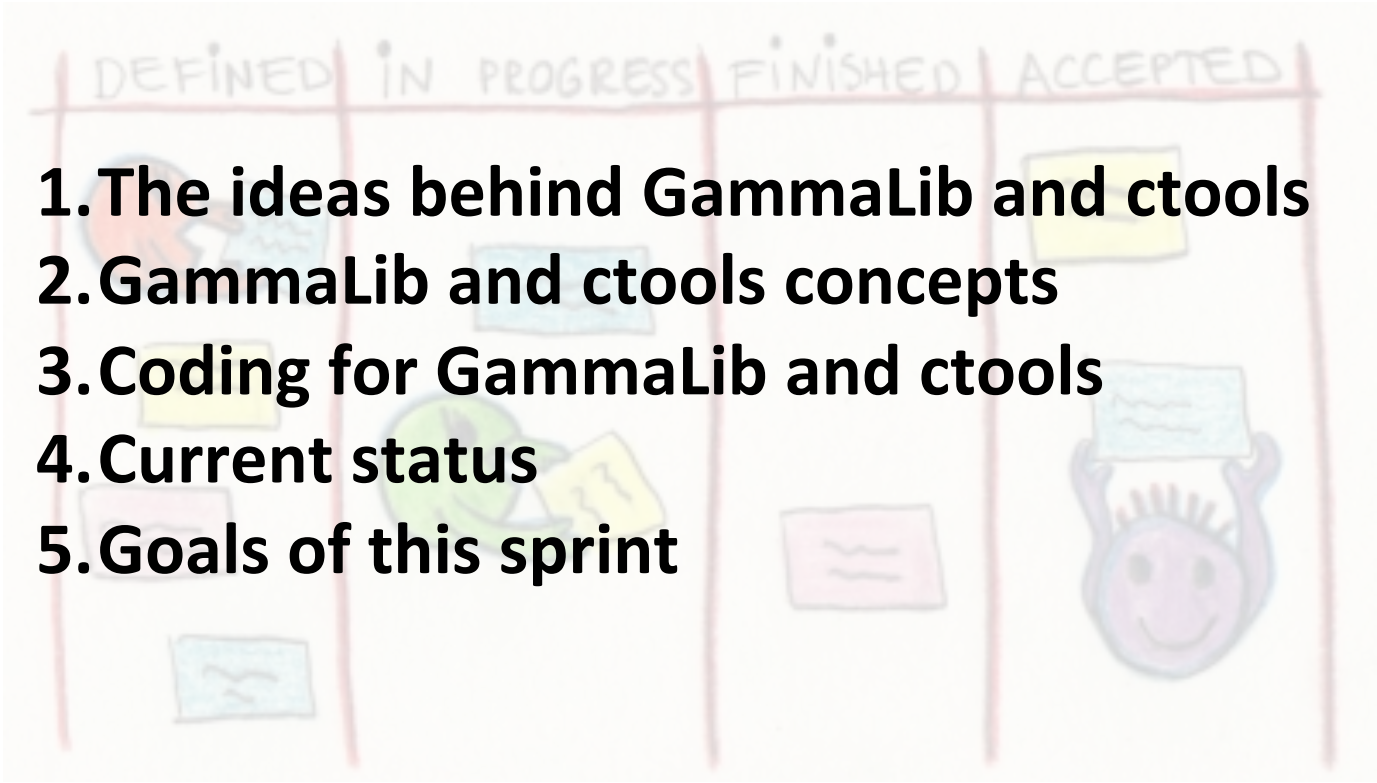
2nd Coding Sprint

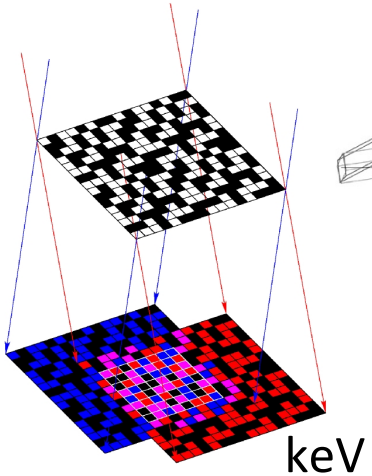


1. The ideas behind GammaLib and ctools
2. GammaLib and ctools concepts
3. Coding for GammaLib and ctools
4. Current status
5. Goals of this sprint

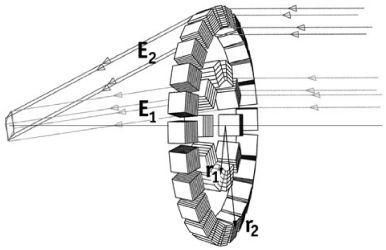
Jürgen Knödlseder (IRAP)

1. The ideas behind GammaLib and ctools

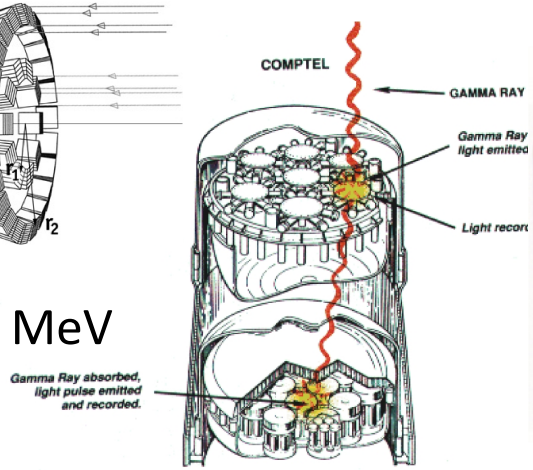
Commonalities

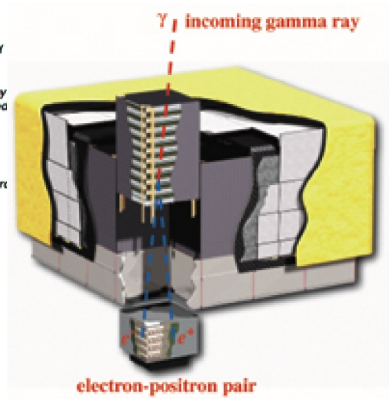


keV



MeV





GeV

TeV

All gamma-ray telescopes measure individual photons as events

Events are characterized by 3 fundamental parameters:

- Localisation
- Energy
- Time

consideration 1: *In principle it should be possible to handle events from all gamma-ray telescopes using a common software framework*

Commonalities

Gamma-ray analysis often consists of (maximum likelihood) fitting of models to the measured event data

consideration 2: *Use maximum likelihood model fitting as central element of the analysis framework*

Existing high-energy analysis frameworks share a number of common features:

- All data are in FITS format (OGIP standard)
- Wide usage of IRAF parameter interface (INTEGRAL, Swift, Chandra, Fermi, ftools, …)
- Modular software based on executables doing a simple and well defined job

consideration 3: *Build a toolbox that provides all the bricks that are required to build any kind of analysis executable*

ctools cherenkov telescope array … is the client that uses the bricks provided by GammaLib

CTA specific *generic*

… to build a set of analysis executables for CTA (and alike)

(Note: from GammaLib you could also build xtools, ytools, ztools, …)

Further considerations

(definitely biased by own experience)

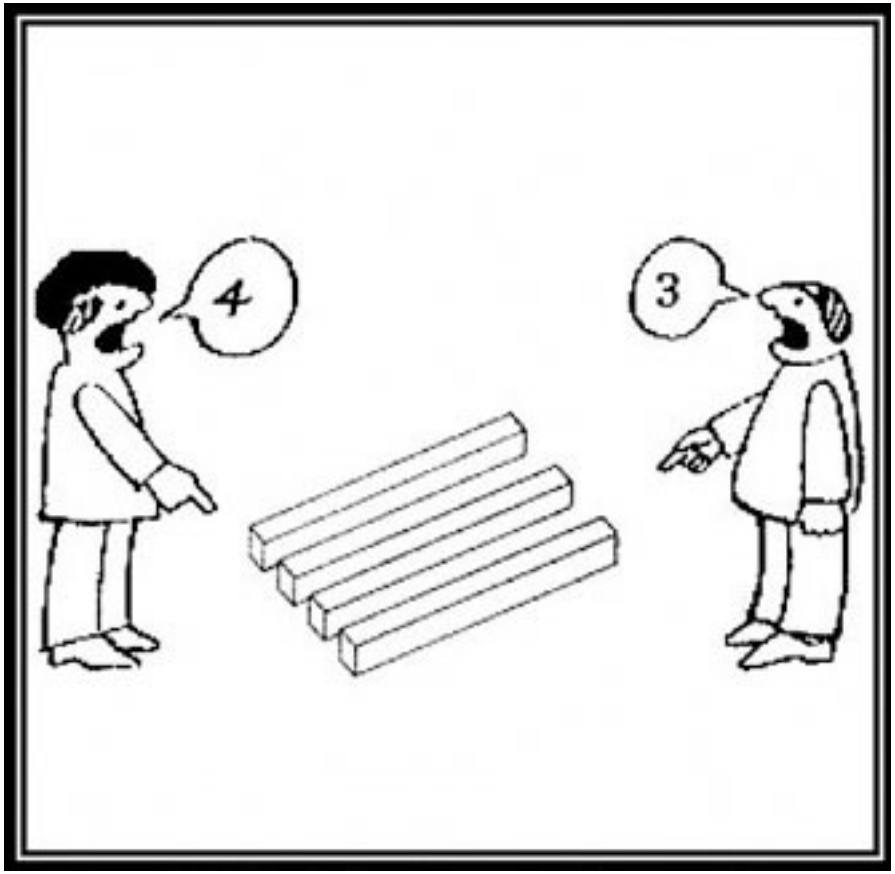
consideration 4: *Avoid dependencies (they make the life very difficult for installation and maintenance)*

consideration 5: *Don't chose a reference platform, try to support as many platforms as possible, and particularly those used by the community at large (enables broad usage, reduces maintenance costs)*

consideration 6: *Software should be open and free of charge (prerequisite for broad acceptance, eases re-use, allows for community contributions)*

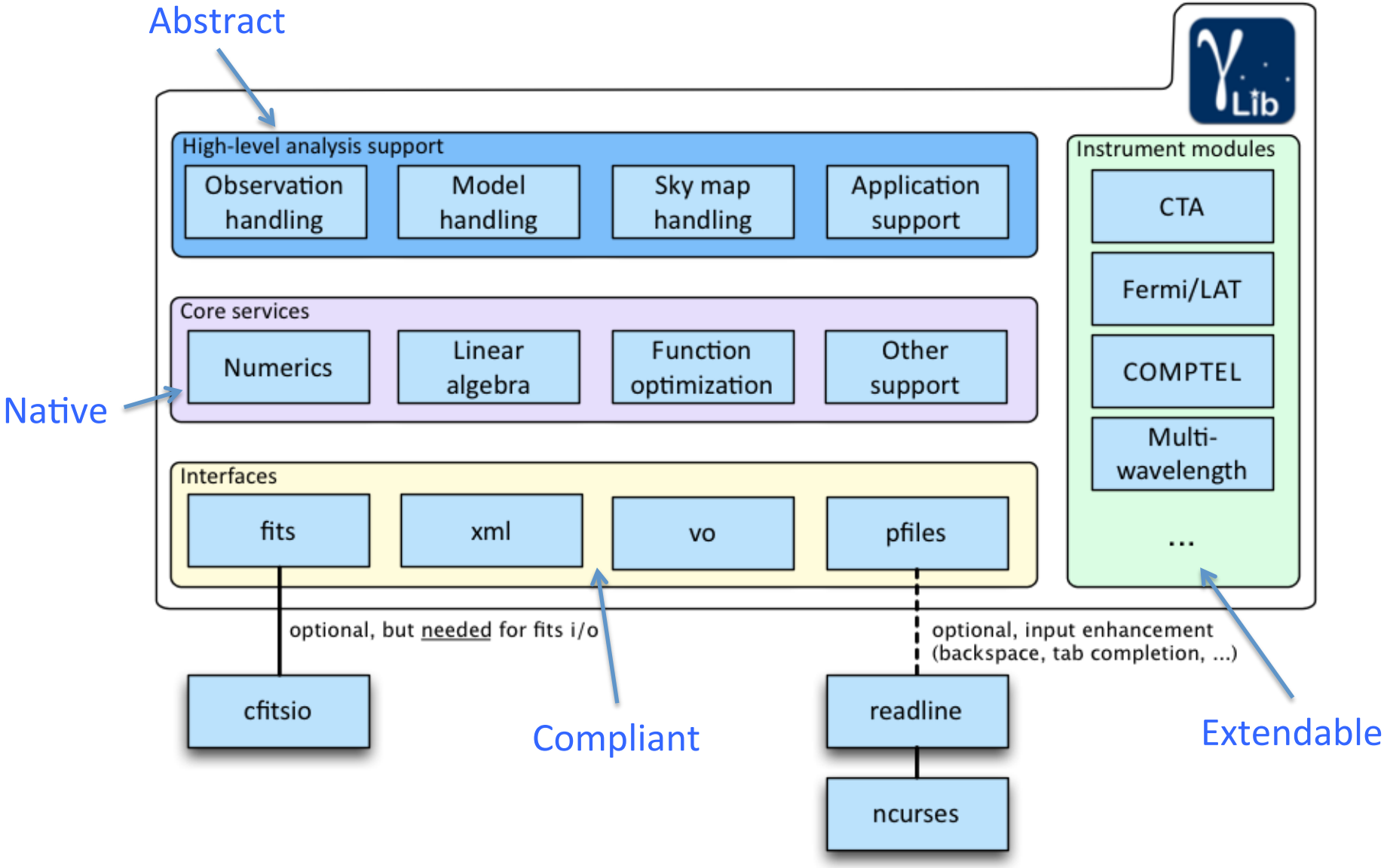
… but it's a challenge

(a Paradigm shift at least for TeV gamma-ray data analysis)



2. GammaLib and ctools concepts

GammaLib overview



It's all C++ classes

```
class GEnergy : public GBase {

    // Operator friends
    friend GEnergy operator+ (const GEnergy &a, const GEnergy &b);
    friend GEnergy operator- (const GEnergy &a, const GEnergy &b);
    friend GEnergy operator* (const double &a, const GEnergy &b);
    friend GEnergy operator* (const GEnergy &a, const double &b);
    friend GEnergy operator/ (const GEnergy &a, const double &b);
    friend bool    operator== (const GEnergy &a, const GEnergy &b);
    friend bool    operator!= (const GEnergy &a, const GEnergy &b);
    friend bool    operator< (const GEnergy &a, const GEnergy &b);
    friend bool    operator<= (const GEnergy &a, const GEnergy &b);
    friend bool    operator> (const GEnergy &a, const GEnergy &b);
    friend bool    operator>= (const GEnergy &a, const GEnergy &b);

public:
    // Constructors and destructors
    GEnergy(void);
    GEnergy(const GEnergy& eng);
    explicit GEnergy(const double& eng, const std::string& unit);
    virtual ~GEnergy(void);

    // Operators
    GEnergy& operator=(const GEnergy& eng);
    GEnergy& operator+=(const GEnergy& eng);
    GEnergy& operator-=(const GEnergy& eng);

    // Methods
    void        clear(void);
    GEnergy*    clone(void) const;
    double      erg(void) const;
    double      keV(void) const;
    double      MeV(void) const;
    double      GeV(void) const;
    double      TeV(void) const;
    double      log10keV(void) const;
    double      log10MeV(void) const;
    double      log10GeV(void) const;
    double      log10TeV(void) const;
    void        erg(const double& eng);
    void        keV(const double& eng);
    void        MeV(const double& eng);
    void        GeV(const double& eng);
    void        TeV(const double& eng);
    void        log10keV(const double& eng);
    void        log10MeV(const double& eng);
    void        log10GeV(const double& eng);
    void        log10TeV(const double& eng);
    std::string print(const GChatter& chatter = NORMAL) const;
```

```
class GApplication : public GBase {

public:
    // Constructors and destructors
    GApplication(void);
    GApplication(const std::string& name, const std::string& version);
    GApplication(const std::string& name, const std::string& version,
                 int argc, char* argv[]);
    GApplication(const GApplication& app);
    ~GApplication(void);

    // Operators
    GApplication& operator=(const GApplication& app);
    GApplicationPar&         operator[](const std::string& name);
    const GApplicationPar&   operator[](const std::string& name) const;

    // Methods
    void                clear(void);
    GApplication*       clone(void) const;
    const std::string&  name(void) const;
    const std::string&  version(void) const;
    double              telapse(void) const;
    double              celapse(void) const;
    void                logFileOpen(const bool& clobber = true);
    bool                logTerse(void) const;
    bool                logNormal(void) const;
    bool                logExplicit(void) const;
    bool                logVerbose(void) const;
    bool                logDebug(void) const;
    bool                clobber(void) const;
    bool                has_par(const std::string& name) const;
    const std::string&  par_filename(void) const;
    const std::string&  log_filename(void) const;
    void                log_header(void);
    void                log_trailer(void);
    void                log_parameters(void);
    std::string         print(const GChatter& chatter = NORMAL) const;

    // Public members
    GLog log;    //!< Application logger
```

Abstract C++ classes for abstract interfaces

```
class GEvent : public GBase {

public:
    // Constructors and destructors
    GEvent(void);
    GEvent(const GEvent& event);
    virtual ~GEvent(void);

    // Operators
    virtual GEvent& operator=(const GEvent& event);

    // Pure virtual methods
    virtual void             clear(void) = 0;
    virtual GEvent*          clone(void) const = 0;
    virtual double           size(void) const = 0;
    virtual const GInstDir&  dir(void) const = 0;
    virtual const GEnergy&   energy(void) const = 0;
    virtual const GTime&     time(void) const = 0;
    virtual double           counts(void) const = 0;
    virtual double           error(void) const = 0;
    virtual bool             is_atom(void) const = 0;
    virtual bool             is_bin(void) const = 0;
    virtual std::string      print(const GChatter& chatter = NORMAL) const = 0;

protected:
    // Protected methods
    void init_members(void);
    void copy_members(const GEvent& event);
    void free_members(void);
};
```

A *ctool* is an executable and a class

```
class ctlike : public GApplication  {
public:
    // Constructors and destructors
    ctlike(void);
    explicit ctlike(GObservations obs);
    ctlike(int argc, char *argv[]);
    ctlike(const ctlike& app);
    virtual ~ctlike(void);

    // Operators
    ctlike& operator= (const ctlike& app);

    // Methods
    void           clear(void);
    void           execute(void);
    void           run(void);
    void           save(void);
    GObservations& obs(void) { return m_obs; }
    GOptimizer*    opt(void) { return m_opt; }
    void           get_parameters(void);
    void           optimize_lm(void);
```

ctlike is a C++ class …

```
int main (int argc, char *argv[])
{
    // Initialise return code
    int rc = 1;

    // Create instance of application
    ctlike application(argc, argv);

    // Run application
    try {
        // Execute application
        application.execute();

        // Signal success
        rc = 0;
    }
    catch (std::exception &e) {

        // Extract error message
        std::string message = e.what();
        std::string signal  = "*** ERROR encounterted in the execution of"
                              " ctlike. Run aborted ...";

        // Write error in logger
        application.log << signal  << std::endl;
        application.log << message << std::endl;

        // Write error on standard output
        std::cout << signal  << std::endl;
        std::cout << message << std::endl;

    } // endcatch: catched any application error

    // Return
    return rc;
}
```

… or as a C++ class in a C++ program (used to build the ctlike executable)

… that can be used as a Python class in a script …

```
# Perform maximum likelihood fitting
like = ctlike()
like.logFileOpen()  # We need this to explicitely open the log file in Python
like["infile"].filename(cntmap_name)
like["srcmdl"].filename(model_name)
like["outmdl"].filename(result_name)
like["caldb"].string(caldb)
like["irf"].string(irf)
like.execute()
sys.stdout.write("Maximum likelihood fitting ("+str(like.celapse())+" CPU sec
```

Running a *ctool* executable

CTA event list simulator

```
$ ctobssim
Model [test/data/crab.xml] $GAMMALIB/share/models/crab.xml
Calibration database [test/irf] $GAMMALIB/share/caldb/cta
Instrument response function [cta_dummy_irf]
RA of pointing (degrees) (0-360) [83.63]
Dec of pointing (degrees) (-90-90) [22.01]
Radius of FOV (degrees) (0-180) [10.0]
Start time (MET in s) (0) [0.0]
End time (MET in s) (0) [1800.0]
Lower energy limit (TeV) (0) [0.1]
Upper energy limit (TeV) (0) [100.0]
Output event data file or observation definition file [events.fits]
```

Wrapping C++ in Python: SWIG

http://www.swig.org/

ctlike.hpp

```
class ctlike : public GApplication  {
public:
    // Constructors and destructors
    ctlike(void);
    explicit ctlike(GObservations obs);
    ctlike(int argc, char *argv[]);
    ctlike(const ctlike& app);
    virtual ~ctlike(void);

    // Operators
    ctlike& operator= (const ctlike& app);

    // Methods
    void           clear(void);
    void           execute(void);
    void           run(void);
    void           save(void);
    GObservations& obs(void) { return m_obs; }
    GOptimizer*    opt(void) { return m_opt; }
    void           get_parameters(void);
    void           optimize_lm(void);
```

ctlike.i

```
class ctlike : public GApplication  {
public:
    // Constructors and destructors
    ctlike(void);
    explicit ctlike(GObservations obs);
    ctlike(int argc, char *argv[]);
    ctlike(const ctlike& app);
    virtual ~ctlike(void);

    // Methods
    void           clear(void);
    void           execute(void);
    void           run(void);
    void           save(void);
    GObservations& obs(void);
    GOptimizer*    opt(void);
    void           get_parameters(void);
    void           optimize_lm(void);
};
%extend ctlike {
    ctlike copy() {
        return (*self);
    }
}
```

```
$ swig -c++ -python -Wall ctlike.i
ctlike.py
ctlike_wrap.cpp
$ gcc ctlike_wrap.cpp
```

Using GammaLib in Python

```
>>> import gammalib
>>> models = gammalib.GModels()
>>> print(models)
=== GModels ===
 Number of models ..........: 0
 Number of parameters ......: 0
>>> pos=gammalib.GSkyDir()
>>> pos.radec_deg(83.6331, 22.0145)
>>> print(pos.l_deg(),pos.b_deg())
(184.55746010138259, -5.7843464490225998)
>>>
```

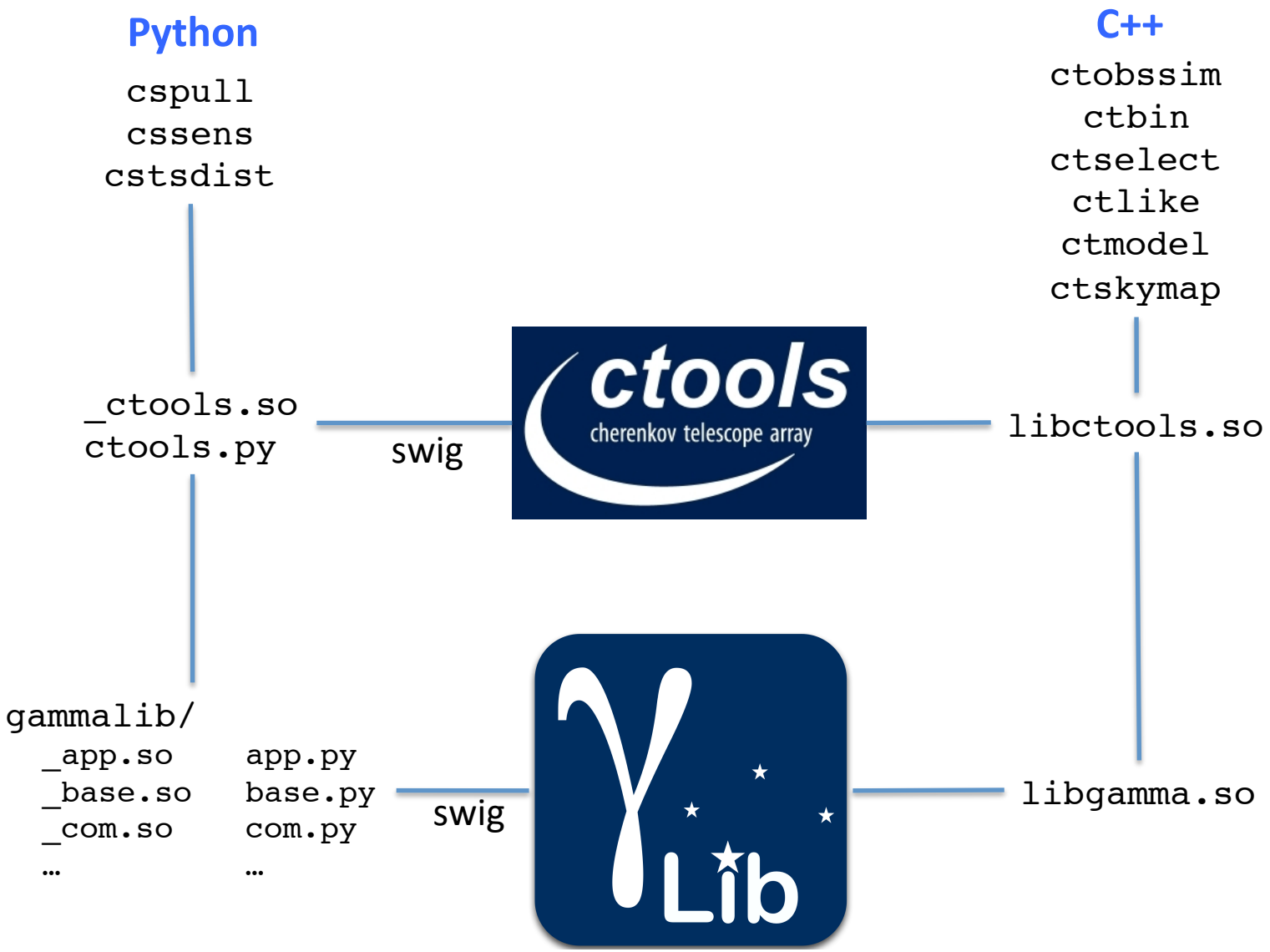
```
>>> from gammalib import *
>>> models = GModels()
>>> print(models)
=== GModels ===
 Number of models ..........: 0
 Number of parameters ......: 0
>>> pos=GSkyDir()
>>> pos.radec_deg(83.6331, 22.0145)
>>> print(pos.l_deg(),pos.b_deg())
(184.55746010138259, -5.7843464490225998)
>>>
```

… same story for *ctools*

A *cscript* is a Python script looking like a *ctool*

```
# ============ #
# cspull class #
# ============ #
class cspull(GApplication):
    """
    This class implements the pull distribution generation script. It derives
    from the GammaLib::GApplication class which provides support for parameter
    files, command line arguments, and logging. In that way the Python
    script behaves just as a regular ctool.
    """
    def __init__(self, *argv):
        """
        Constructor.
        """
        # Set name
        self.name    = "cspull"
        self.version = "0.2.0"

        # Initialise some members
        self.obs      = None
        self.model    = None
        self.m_srcmdl = None

        # Make sure that parfile exists
        file = self.parfile()

        # Initialise application
        if len(argv) == 0:
            GApplication.__init__(self, self.name, self.version)
        elif len(argv) ==1:
            GApplication.__init__(self, self.name, self.version, *argv)
        else:
            raise TypeError("Invalid number of arguments given.")

        # Set logger properties
        self.log_header()
        self.log.date(True)

        # Return
        return
```

The overall picture



What should I do if …

… I need a new spectral model?
Add a new spectral model class to the GammaLib model module.

… I need a new background model for CTA?
Add a new background model class to the GammaLib CTA interface module.

… I want a tool that generates CTA exposure maps?
Create a new ctool that uses the CTA response functions in GammaLib for exposure map computation.

… I want to implement an analysis workflow or pipeline?
Create a Python script that uses the ctools and gammalib Python modules.

… I want to test a new idea (e.g. create a ring background generator)?
Create a new cscript that uses the gammalib Python module.

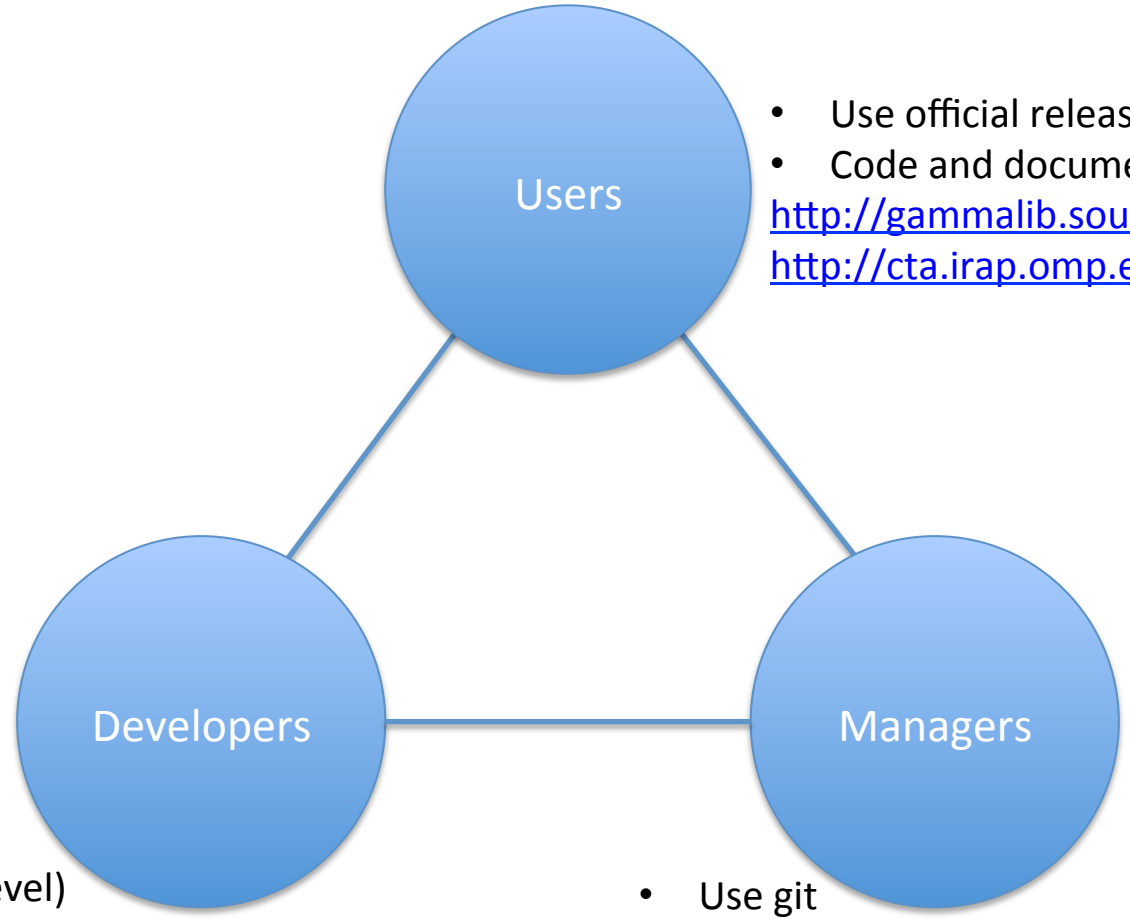
General rule:
All generic and reusable code goes in GammaLib, code that is only needed for one specific task goes in ctools. Quick coding is better done by a cscript.

3. Coding for GammaLib and ctools

- Infrastructure -

Communities

@gammalib



- Use official releases
- Code and documentation at

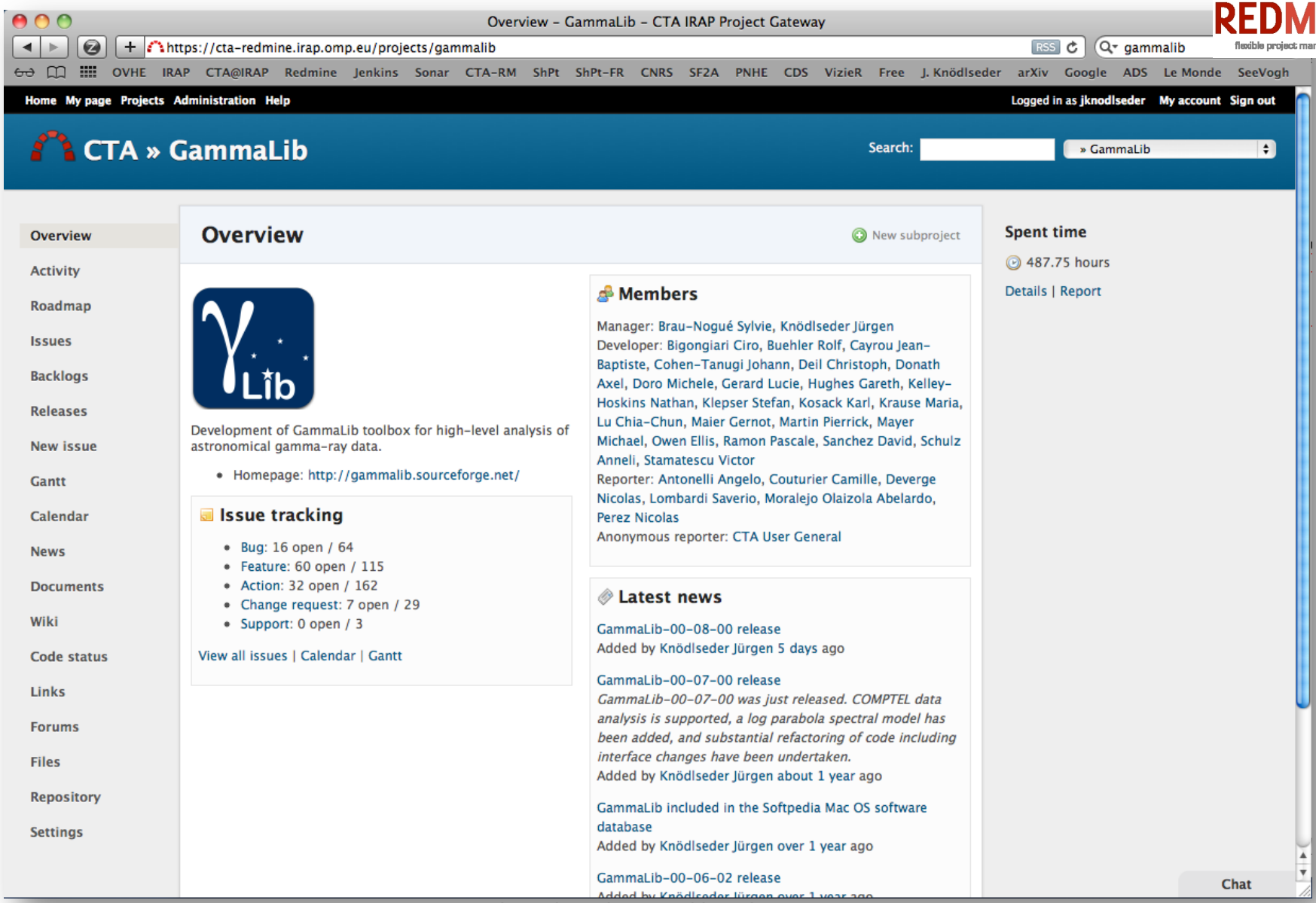
http://gammalib.sourceforge.net/
http://cta.irap.omp.eu/ctools/index.html

- Use git trunk (devel)
- Forge including developer documentation at

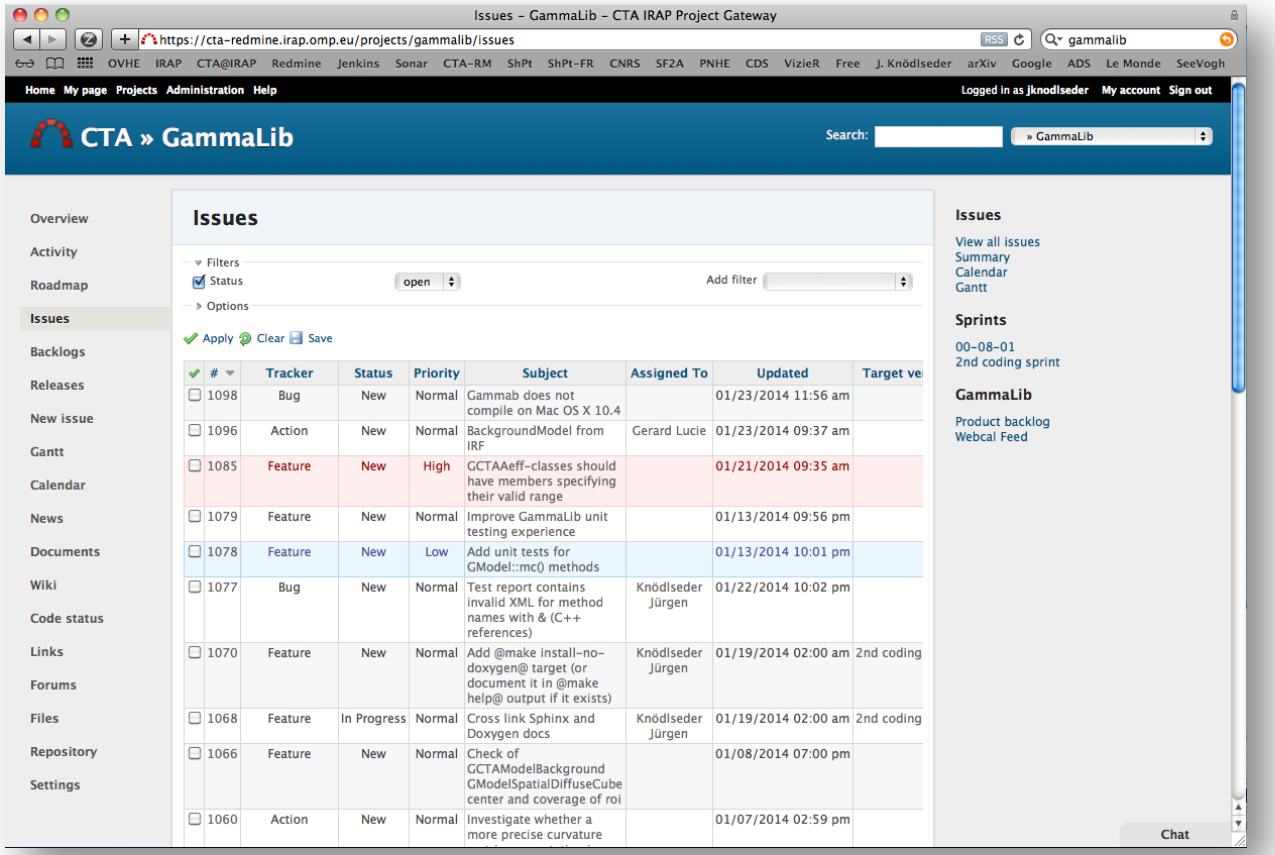
https://cta-redmine.irap.omp.eu/projects/gammalib
https://cta-redmine.irap.omp.eu/projects/ctools

- Use git
- Forge, Jenkins, Sonar
- Unit tests (`make check`)
- Documentation as for developers

Our Forge


Overview - GammaLib - CTA IRAP Project Gateway
https://cta-redmine.irap.omp.eu/projects/gammalib
gammalib
OVHE IRAP CTA@IRAP Redmine Jenkins Sonar CTA-RM ShPt ShPt-FR CNRS SF2A PNHE CDS VizieR Free J. Knödlseder arXiv Google ADS Le Monde SeeVogh
Home My page Projects Administration Help
Logged in as jknodlseder My account Sign out
CTA » GammaLib
Search:
» GammaLib
Overview
Activity
Roadmap
Issues
Backlogs
Releases
New issue
Gantt
Calendar
News
Documents
Wiki
Code status
Links
Forums
Files
Repository
Settings
Overview
New subproject
Development of GammaLib toolbox for high-level analysis of astronomical gamma-ray data.
Homepage: http://gammalib.sourceforge.net/
Issue tracking
Bug: 16 open / 64
Feature: 60 open / 115
Action: 32 open / 162
Change request: 7 open / 29
Support: 0 open / 3
View all issues | Calendar | Gantt
Members
Manager: Brau-Nogué Sylvie, Knödlseder Jürgen
Developer: Bigongiari Ciro, Buehler Rolf, Cayrou Jean-Baptiste, Cohen-Tanugi Johann, Deil Christoph, Donath Axel, Doro Michele, Gerard Lucie, Hughes Gareth, Kelley-Hoskins Nathan, Klepser Stefan, Kosack Karl, Krause Maria, Lu Chia-Chun, Maier Gernot, Martin Pierrick, Mayer Michael, Owen Ellis, Ramon Pascale, Sanchez David, Schulz Anneli, Stamatescu Victor
Reporter: Antonelli Angelo, Couturier Camille, Deverge Nicolas, Lombardi Saverio, Moralejo Olaizola Abelardo, Perez Nicolas
Anonymous reporter: CTA User General
Latest news
GammaLib-00-08-00 release
Added by Knödlseder Jürgen 5 days ago
GammaLib-00-07-00 release
GammaLib-00-07-00 was just released. COMPTEL data analysis is supported, a log parabola spectral model has been added, and substantial refactoring of code including interface changes have been undertaken.
Added by Knödlseder Jürgen about 1 year ago
GammaLib included in the Softpedia Mac OS software database
Added by Knödlseder Jürgen over 1 year ago
GammaLib-00-06-02 release
Spent time
487.75 hours
Details | Report
Chat

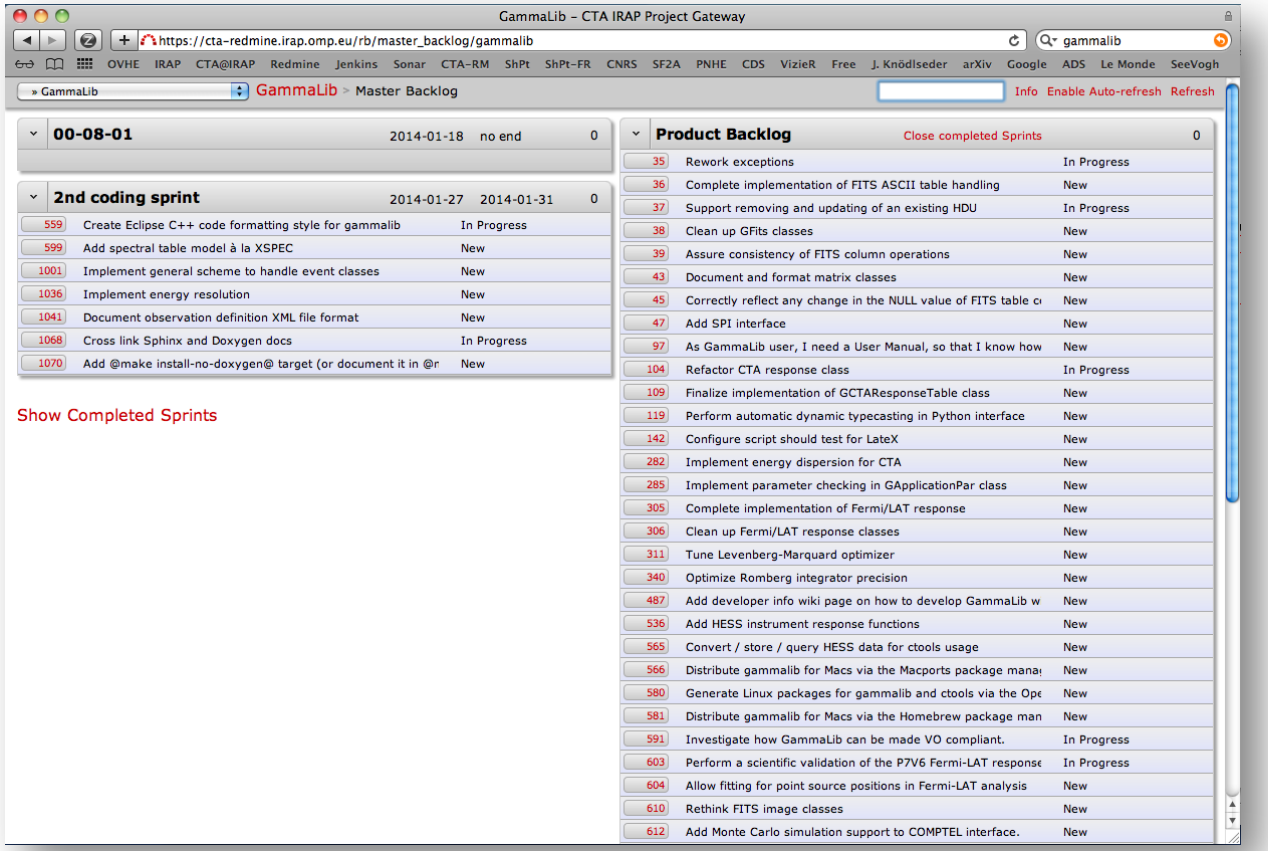
Issue tracking



Please track your work, problems, errors, etc.!

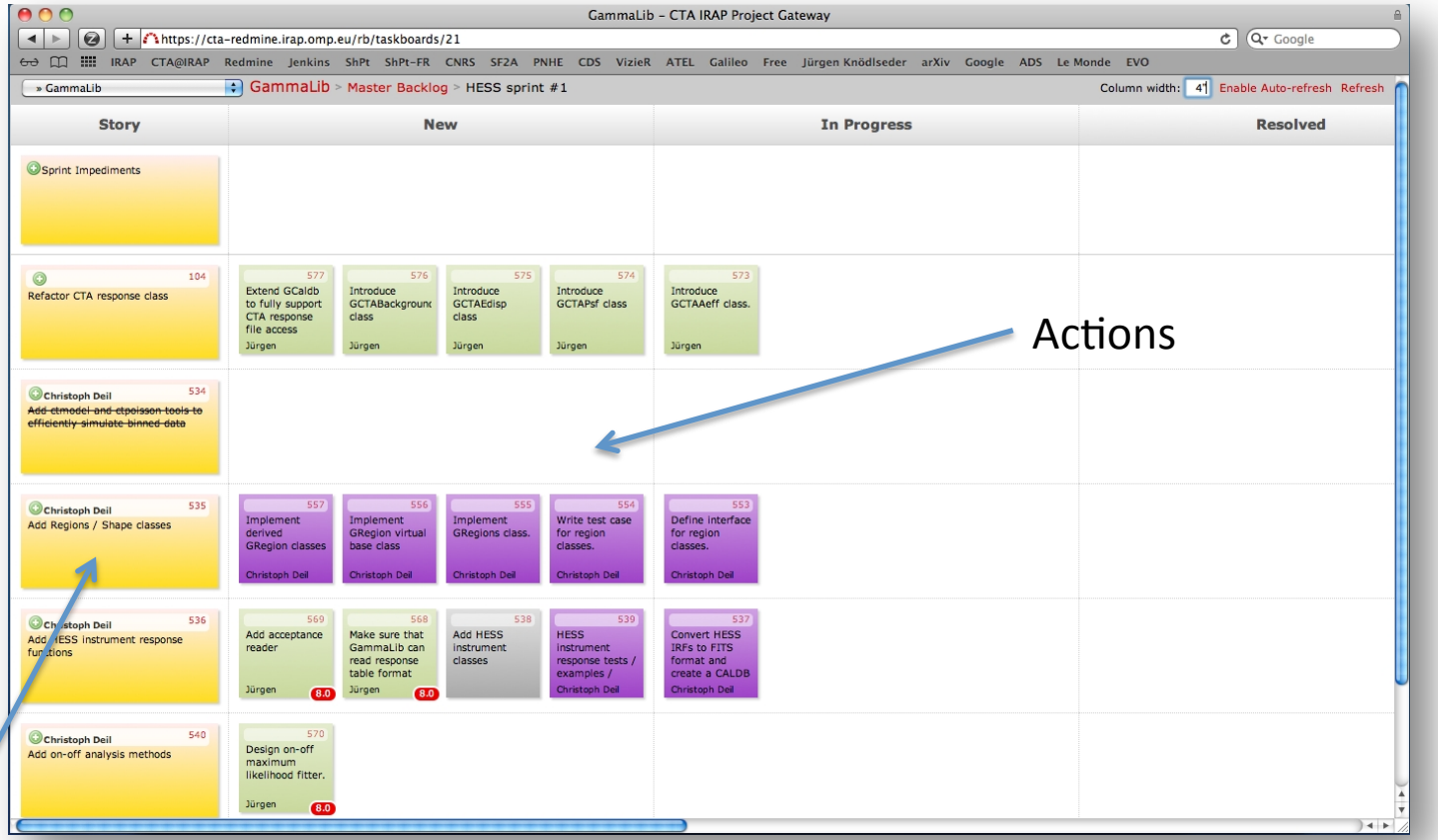
- Bug
- Action (something to do)
- Feature (a new thing, may be composed of several actions)
- Change request

Work planning



Collects all features of the sprint

Task board



Actions

Features

Graphical representation of all features and actions, their status and assigned persons

Repository organization

The GammaLib and ctools source code are version controlled in two *git* repositories at IRAP

https://cta-git.irap.omp.eu/gammalib

https://cta-git.irap.omp.eu/ctools

Protected branches:

master – last release

release – release preparation

devel – developer branch

integration – feature integration

Other branches: can be setup by any developer as required. Will be regularly cleaned-up after pulling in changes.

Gammalib and ctools development can also be based on Github:

https://github.com/gammalib/gammalib

https://github.com/ctools/ctools

- read-only repositories
- synchronized with IRAP repositories

A typical git workflow

https://cta-redmine.irap.omp.eu/projects/gammalib/wiki/Git_workflow_for_GammaLib

```
$ git clone https://jknodlseder@cta-git.irap.omp.eu/gammalib
Cloning into 'gammalib'...
Password:
remote: Counting objects: 32374, done.
remote: Compressing objects: 100% (8231/8231), done.
remote: Total 32374 (delta 25847), reused 30500 (delta 24058)
Receiving objects: 100% (32374/32374), 85.24 MiB | 1.10 MiB/s, done.
Resolving deltas: 100% (25847/25847), done.
$ cd gammalib/
$ git checkout devel
Branch devel set up to track remote branch devel from origin.
Switched to a new branch 'devel'
$ git checkout -b bugfix
Switched to a new branch 'bugfix'
$ nano AUTHORS
$ git add AUTHORS
$ git commit -m "Remove extra line"
[bugfix 3c7ecaf] Remove extra line
 1 files changed, 0 insertions(+), 1 deletions(-)
$ git push origin bugfix
Password:
Counting objects: 5, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 307 bytes, done.
Total 3 (delta 2), reused 1 (delta 1)
remote: To https://github.com/gammalib/gammalib.git
remote:  * [new branch]      bugfix -> bugfix
remote:  * [new branch]      github/007-my-new-feature -> github/007-my-new-feature
To https://jknodlseder@cta-git.irap.omp.eu/gammalib
 * [new branch]      bugfix -> bugfix
$
```

Some frequently asked questions

Should I use IRAP git or github for development?
Whatever you prefer. Both repositories should be 100% synchronized at any time.

Which branch should I start from?
Always branch from devel. Never branch from master.

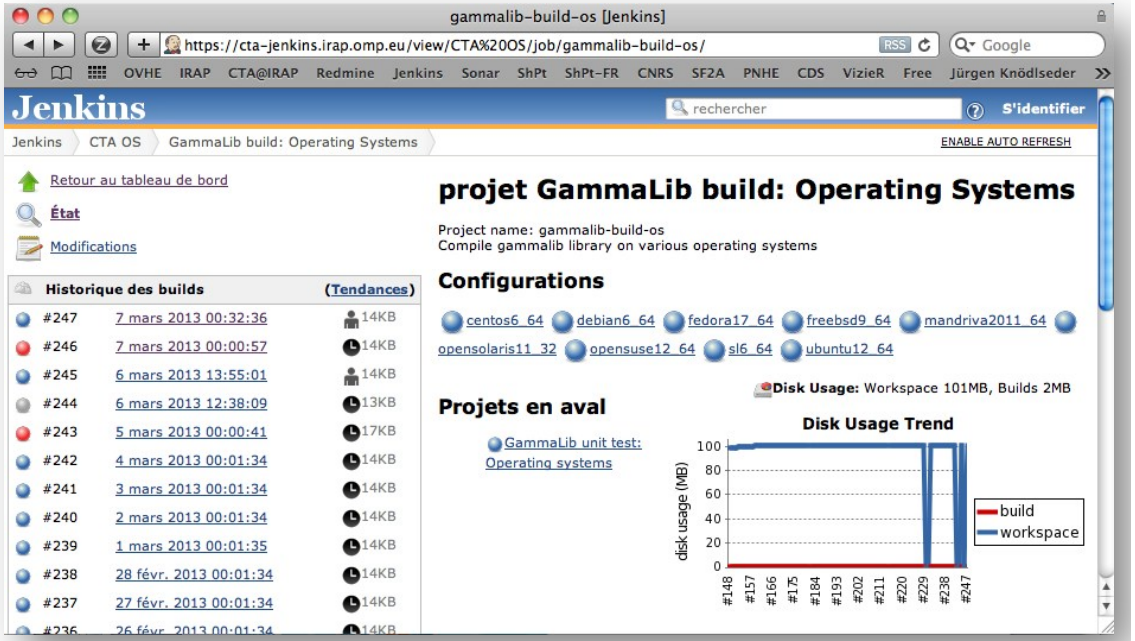
How often should I commit?
Whenever you feel necessary. Note that the more often you commit the better all changes are tracked and the easier it is to go back to a certain stage of your code development. However, before committing, please check that the code at least compiles (best make also a unit test).

Why can't I push to master, release, devel or integration?
These branches are protected from pushing. Only the integration manager is allowed to push to them. See it from the good side: this puts a lower work burden on your side, and prevents you from destroying the repository.

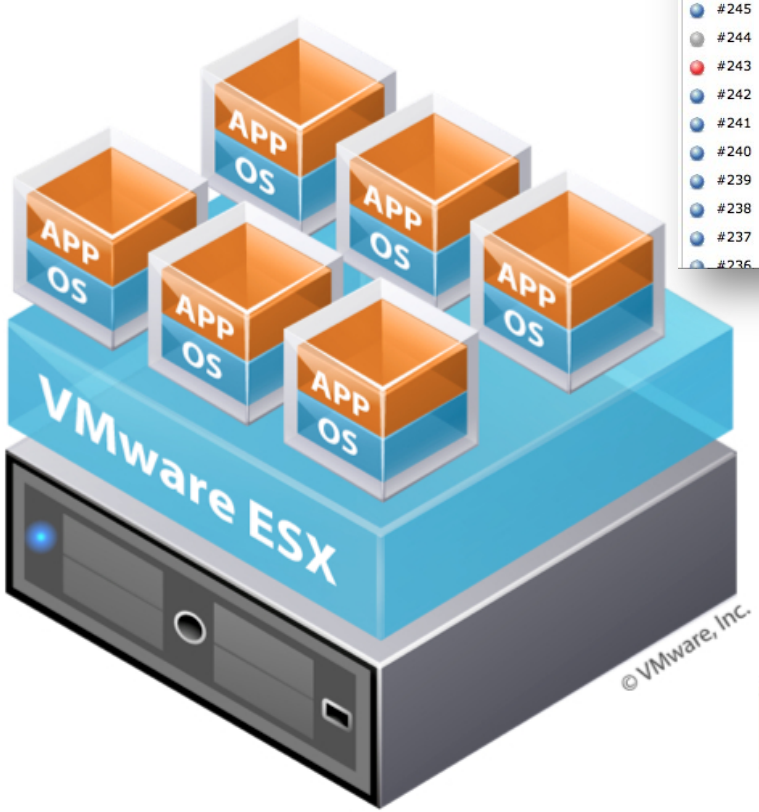
Code quality (continuous integration)

Jenkins

Continuous integration
(build, check, install, analysis)



Multi-platform, multi-compiler, 32/64 Bit,
Python version, swig version, …



Virtual box with relevant OS

vmware®

Unit testing

Jenkins

Configuration centos6_64

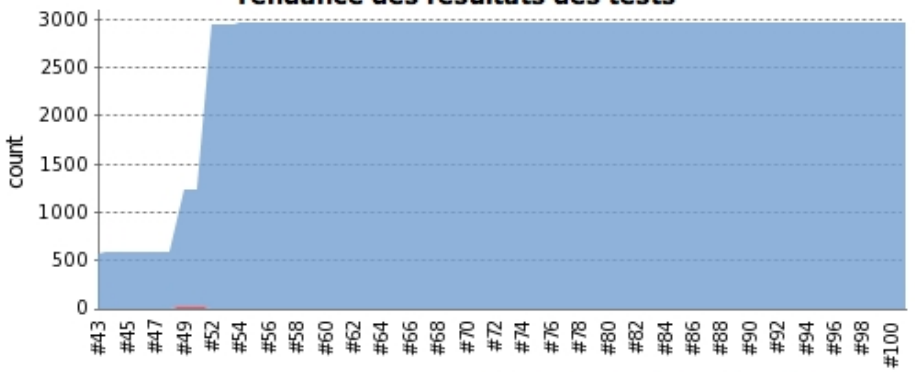
ajouter une description

Espace de travail

Changements récents

Derniers résultats des tests (aucun échec)

Tendance des résultats des tests



(montrer les échecs seulement) agrandir

Liens permanents

- Dernier build (#101), il y a 21 h
- Dernier build stable (#101), il y a 21 h
- Dernier build avec succès (#101), il y a 21 h
- Dernier build en échec (#51), il y a 1 mo. 15 j
- Last unsuccessful build (#51), il y a 1 mo. 15 j

Compilation #49 (5 sept. 2012 00:08:17)

Conserver ce build sans limite de temps

Démarré il y a 1 mo. 16 j
A pris 2 mn 55 s sur CentOS 6

ajouter une description

Aucun changement.

Lancé par le projet amont gammalib-check-os avec le numéro de build 49

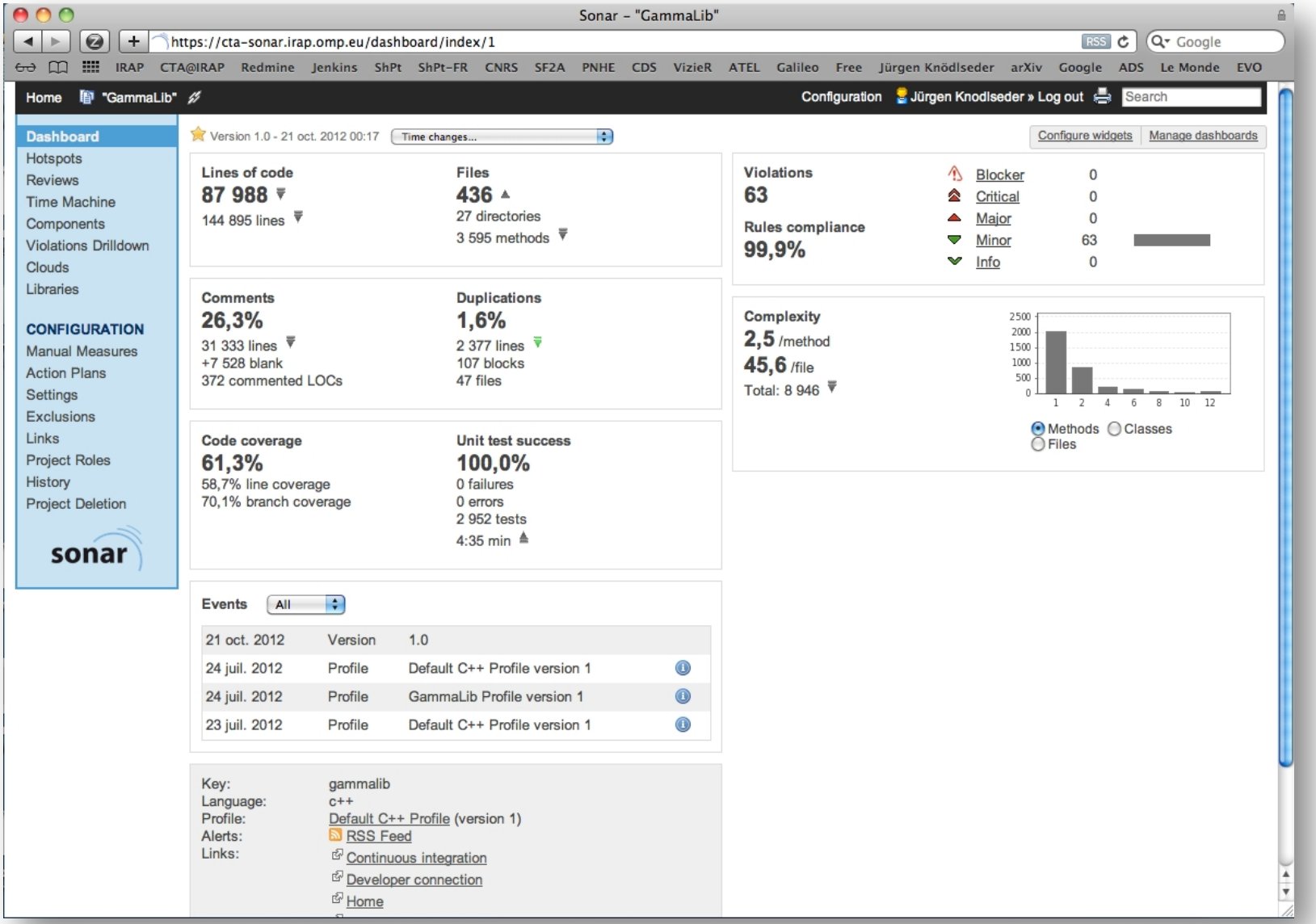
Résultats des tests (2 échecs / +2)
Multi-wavelength instrument specific class testing.Test optimizer: Verify optimization result for Index .....: -3.11112 +/- 0.127403 [-5,5] (free,scale=1,gradient)
Multi-wavelength instrument specific class testing.Test optimizer: Verify optimization result for Index .....: -2.1374 +/- 0.00578641 [-5,5] (free,scale=1,gradient)

Parameters

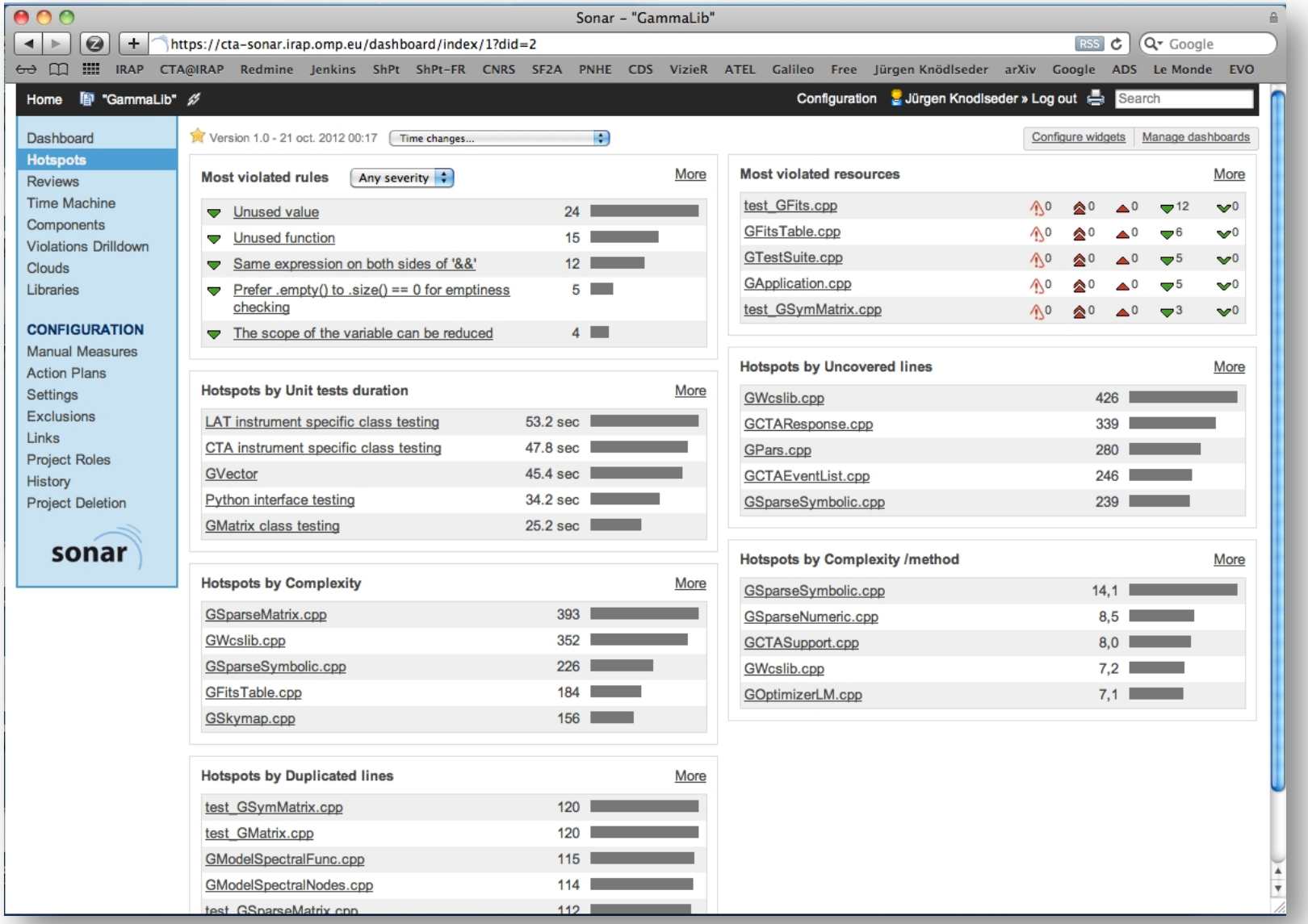
Branch: devel

Please select the Git branch

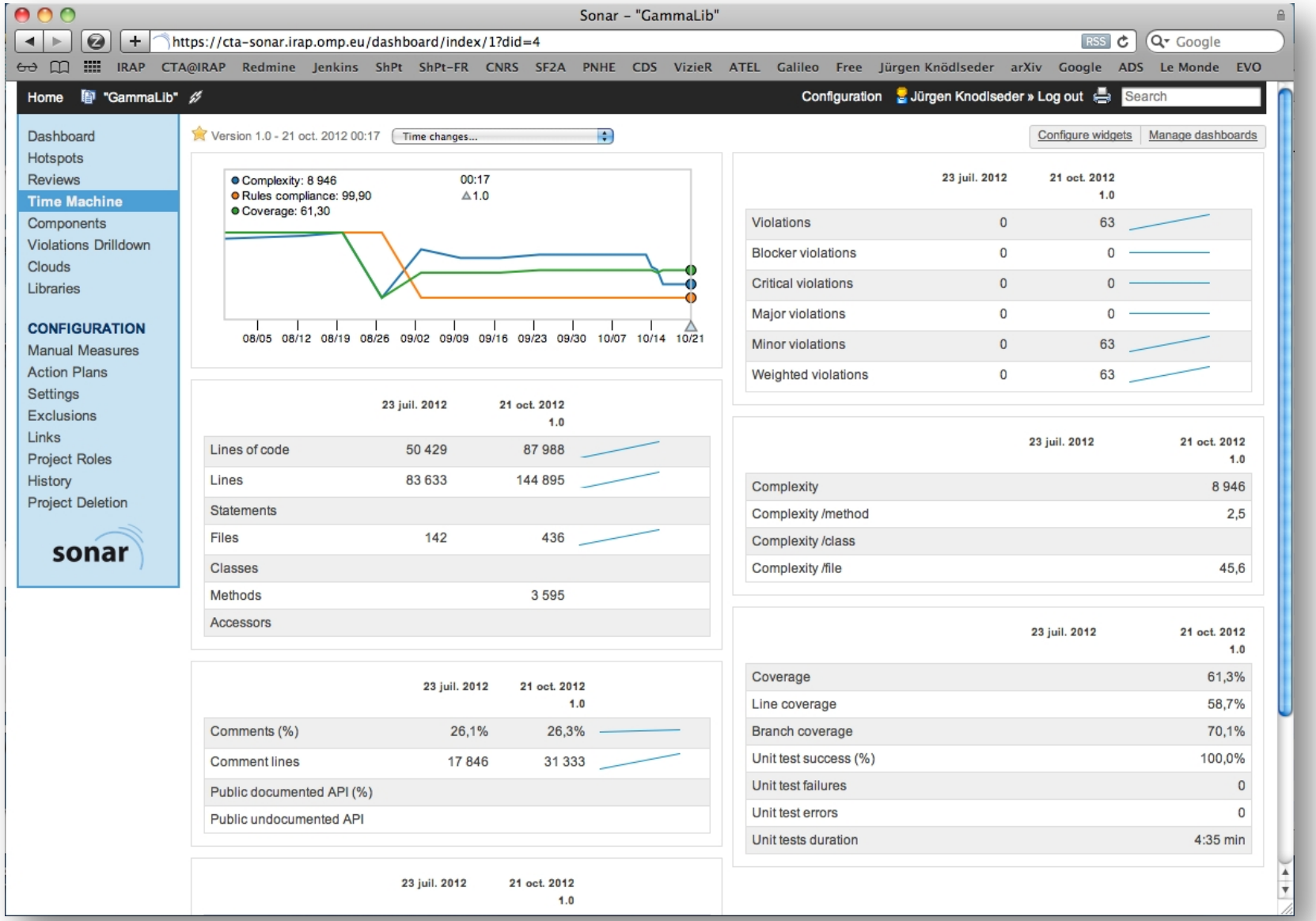
Code analysis



Hotspots


Sonar - "GammaLib"
https://cta-sonar.irap.omp.eu/dashboard/index/1?did=2
Home
"GammaLib"
Configuration
Jürgen Knodlseder » Log out
Search
Dashboard
Hotspots
Reviews
Time Machine
Components
Violations Drilldown
Clouds
Libraries
CONFIGURATION
Manual Measures
Action Plans
Settings
Exclusions
Links
Project Roles
History
Project Deletion
Version 1.0 - 21 oct. 2012 00:17
Time changes...
Configure widgets
Manage dashboards
Most violated rules
Any severity
More
Unused value 24
Unused function 15
Same expression on both sides of '&&' 12
Prefer .empty() to .size() == 0 for emptiness checking 5
The scope of the variable can be reduced 4
Most violated resources
test_GFits.cpp
GFitsTable.cpp
GTestSuite.cpp
GApplication.cpp
test_GSymMatrix.cpp
Hotspots by Unit tests duration
LAT instrument specific class testing 53.2 sec
CTA instrument specific class testing 47.8 sec
GVector 45.4 sec
Python interface testing 34.2 sec
GMatrix class testing 25.2 sec
Hotspots by Uncovered lines
GWcslib.cpp 426
GCTAResponse.cpp 339
GPars.cpp 280
GCTAEventList.cpp 246
GSparseSymbolic.cpp 239
Hotspots by Complexity
GSparseMatrix.cpp 393
GWcslib.cpp 352
GSparseSymbolic.cpp 226
GFitsTable.cpp 184
GSkymap.cpp 156
Hotspots by Complexity /method
GSparseSymbolic.cpp 14,1
GSparseNumeric.cpp 8,5
GCTASupport.cpp 8,0
GWcslib.cpp 7,2
GOptimizerLM.cpp 7,1
Hotspots by Duplicated lines
test_GSymMatrix.cpp 120
test_GMatrix.cpp 120
GModelSpectralFunc.cpp 115
GModelSpectralNodes.cpp 114

Evolutions


Sonar - "GammaLib"
https://cta-sonar.irap.omp.eu/dashboard/index/1?did=4
Home
"GammaLib"
Configuration
Jürgen Knodlseder » Log out
Search
Dashboard
Hotspots
Reviews
Time Machine
Components
Violations Drilldown
Clouds
Libraries
CONFIGURATION
Manual Measures
Action Plans
Settings
Exclusions
Links
Project Roles
History
Project Deletion
Version 1.0 - 21 oct. 2012 00:17
Time changes...
Configure widgets
Manage dashboards
Complexity: 8 946
Rules compliance: 99,90
Coverage: 61,30
00:17
1.0
08/05 08/12 08/19 08/26 09/02 09/09 09/16 09/23 09/30 10/07 10/14 10/21
23 juil. 2012
21 oct. 2012
1.0
Violations 0 63
Blocker violations 0 0
Critical violations 0 0
Major violations 0 0
Minor violations 0 63
Weighted violations 0 63
Lines of code 50 429 87 988
Lines 83 633 144 895
Statements
Files 142 436
Classes
Methods 3 595
Accessors
Complexity 8 946
Complexity /method 2,5
Complexity /class
Complexity /file 45,6
Comments (%) 26,1% 26,3%
Comment lines 17 846 31 333
Public documented API (%)
Public undocumented API
Coverage 61,3%
Line coverage 58,7%
Branch coverage 70,1%
Unit test success (%) 100,0%
Unit test failures 0
Unit test errors 0
Unit tests duration 4:35 min

Documentation

Code documentation

Doxygen

Extracts code documentation directly from source files. Latest version of devel branch online at
http://gammalib.sourceforge.net/doxygen/
http://cta.irap.omp.eu/ctools/doxygen/

User documentation

SPHINX
Python Documentation Generator

Generates documents from reStructuredText files (markup language). Latest version of devel branch online at
http://gammalib.sourceforge.net/
http://cta.irap.omp.eu/ctools/

```
/*****************************************************************************//**
 * @brief Evaluate function
 *
 * @param[in] srcEng True photon energy.
 * @param[in] srcTime True photon arrival time.
 * @return Model value (ph/cm2/s/MeV).
 *
 * Evaluates
 *
 * \f[
 *    S_{\rm E}(E | t) = {\tt m\_norm}
 *    \left( \frac{E}{\tt m\_pivot} \right)^{\tt m\_index}
 * \f]
 *
 * where
 * - \f${\tt m\_norm}\f$ is the normalization or prefactor,
 * - \f${\tt m\_index}\f$ is the spectral index, and
 * - \f${\tt m\_pivot}\f$ is the pivot energy.
 *
 * @todo The method expects that energy!=0. Otherwise Inf or NaN may result.
 *****************************************************************************/
double GModelSpectralPlaw::eval(const GEnergy& srcEng,
                                const GTime&   srcTime) const
{
```

Getting help

Python

```
>>> help(GModels)
Help on class GModels in module gammalib.model:

class GModels(gammalib.base.GContainer)
 |  Proxy of C++ GModels class
 |
 |  Method resolution order:
 |      GModels
 |      gammalib.base.GContainer
 |      gammalib.base.GBase
 |      __builtin__.object
 |
 |  Methods defined here:
 |
 |  __del__ lambda self
 |
 |  __getattr__ lambda self, name
 |
 |  __getitem__(self, *args)
 |      __getitem__(GModels self, int const & index) -> GModel
 |      __getitem__(GModels self, std::string const & name) -> GModel
 |
 |  __init__(self, *args)
 |      __init__(GModels self) -> GModels
 |      __init__(GModels self, GModels models) -> GModels
 |      __init__(GModels self, std::string const & filename) -> GModels
 |
 |  __repr__ = _swig_repr(self)
 |
 |  __setattr__ lambda self, name, value
 |
 |  __setitem__(self, *args)
 |      __setitem__(GModels self, int const & index, GModel val)
 |      __setitem__(GModels self, std::string const & name, GModel val)
:
```

C++

```
Jurgen-Knodlseders-Mac-Book:~ jurgen$ man GModels

GModels(3)                                        GammaLib

NAME
       GModels -

       Model container class.

SYNOPSIS
       #include <GModels.hpp>

       Inherits GContainer.

   Public Member Functions
       GModels (void)
           Void constructor.
       GModels (const GModels &models)
           Copy constructor.
       GModels (const std::string &filename)
           Load constructor.
       virtual ~GModels (void)
           Destructor.
       GModels & operator= (const GModels &models)
           Assignment operator.
       GModel * operator[] (const int &index)
           Return pointer to model.
       const GModel * operator[] (const int &index) const
           Return pointer to model (const version)
       GModel * operator[] (const std::string &name)
           Return pointer to model.
       const GModel * operator[] (const std::string &name) const
           Return pointer to model (const version)
       void clear (void)
           Clear object.
       GModels * clone (void) const
           Clone instance.
       GModel * at (const int &index)
           Return pointer to model.
       const GModel * at (const int &index) const
           Return pointer to model (const version)
```

3. Coding for GammaLib and ctools

- Under the hood -

Code organisation

https://cta-git.irap.omp.eu/gammalib

gammalib/	
dev	Developer material
doc	Code and user documentation
examples	Example code
include	Core* header files (.hpp)
inst	Instrument modules
m4	Code configuration macros
pyext	Core* Python extension files (.i)
src	Core* source files (.cpp)
test	Code for unit testing

**Core means instrument independent code*

https://cta-git.irap.omp.eu/ctools

ctools/	
caldb	Calibration data
doc	Code and user documentation
examples	Example code
m4	Code configuration macros
models	Source and background models
pyext	Python extension files (.i)
scripts	cscripts and Python scripts
src	ctools
test	Code for unit testing

Configuring GammaLib

https://cta-redmine.irap.omp.eu/projects/gammalib/wiki/Contributing_to_GammaLib

```
$ ./autogen.sh
$ ./configure
  ...
  GammaLib configuration summary
  ==============================
  * FITS I/O support              (yes)   /usr/local/gamma/lib /usr/local/gamma/include
  * Readline support              (yes)   /usr/local/gamma/lib /usr/local/gamma/include/readline
  * Ncurses support               (yes)
  * Make Python binding           (yes)   use swig for building
  * Python                        (yes)
  * Python.h                      (yes)
  - Python wrappers               (no)
  * swig                          (yes)
  * Multiwavelength interface     (yes)
  * Fermi-LAT interface           (yes)
  * CTA interface                 (yes)
  * COMPTEL interface             (yes)
  * Doxygen                       (yes)   /opt/local/bin/doxygen
  * Perform NaN/Inf checks        (yes)   (default)
  * Perform range checking        (yes)   (default)
  * Optimize memory usage         (yes)   (default)
  * Enable OpenMP                 (yes)   (default)
  - Compile in debug code         (no)    (default)
  - Enable code for profiling     (no)    (default)

Now type 'make'
```

generates configure script from configure.ac

configures GammaLib for your system

Building, checking, installing

https://cta-redmine.irap.omp.eu/projects/gammalib/wiki/Contributing_to_GammaLib

```
$ make -j4
  ...
$ make check
...
PASS: test_GSupport
PASS: test_GVector
PASS: test_GMatrix
PASS: test_GMatrixSparse
PASS: test_GMatrixSymmetric
PASS: test_GNumerics
PASS: test_GFits
PASS: test_GXml
PASS: test_GVO
PASS: test_GXspec
PASS: test_GApplication
PASS: test_GModel
PASS: test_GSky
PASS: test_GOptimizer
PASS: test_GObservation
PASS: test_MWL
PASS: test_CTA
PASS: test_LAT
PASS: test_COM
PASS: test_python.py
make[4]: Nothing to be done for `all'.
==================================================================
Testsuite summary for gammalib 0.8.0
==================================================================
# TOTAL: 20
# PASS:  20
# SKIP:  0
# XFAIL: 0
# FAIL:  0
# XPASS: 0
# ERROR: 0
==================================================================
$ make install
  ...
```

compiles code (using 4 cores at maximum)

compiles and executes unit test code

installs code (copy of build result)

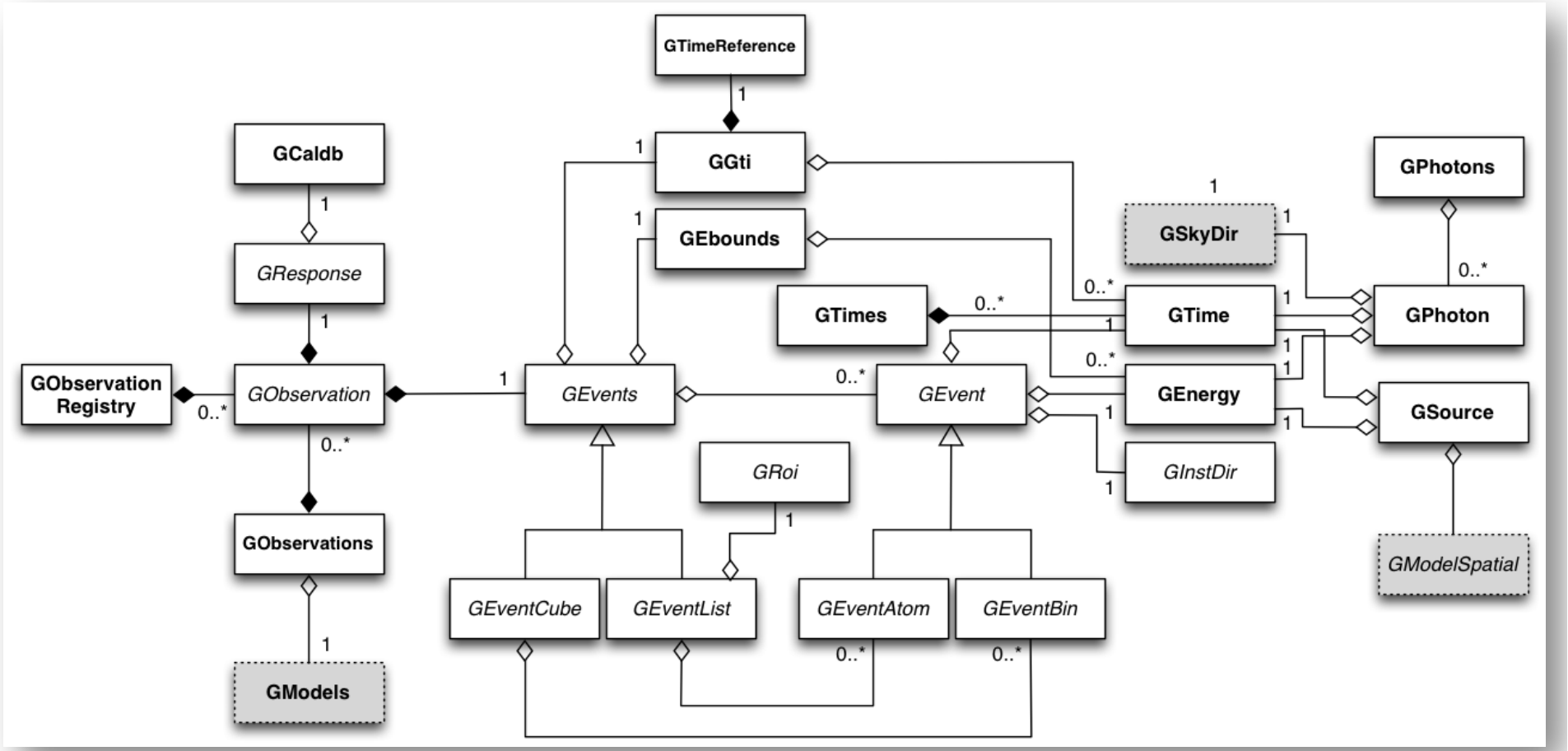
Note: if you switch branches you may need to issue

```
$ make clean
$ make –j4
```

for a full recompliation of the library. Also, if some symbols are missing when doing a unit check, make a full recompilation (and get some coffee).

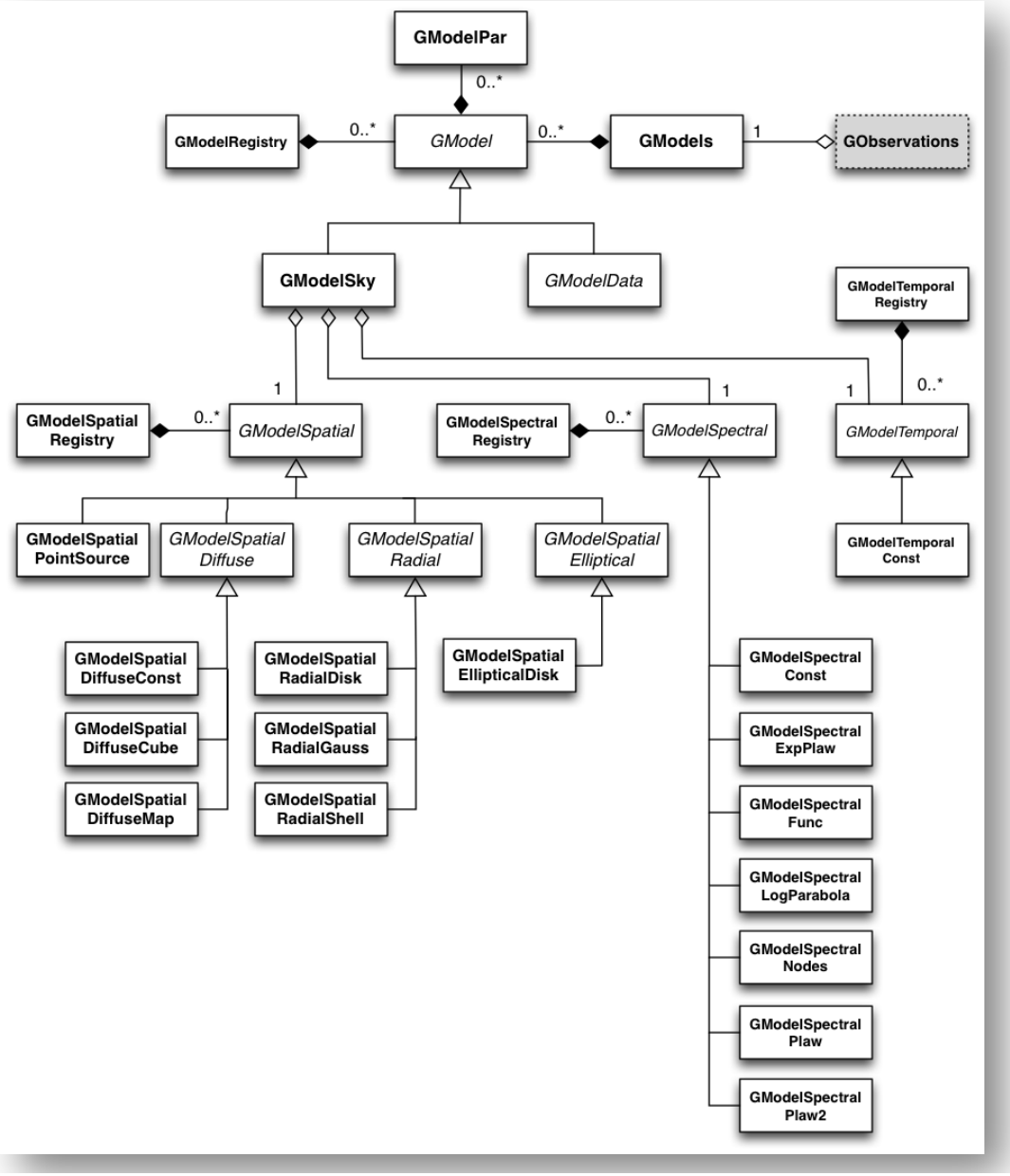
Observation handling

http://gammalib.sourceforge.net/user_manual/modules/obs.html



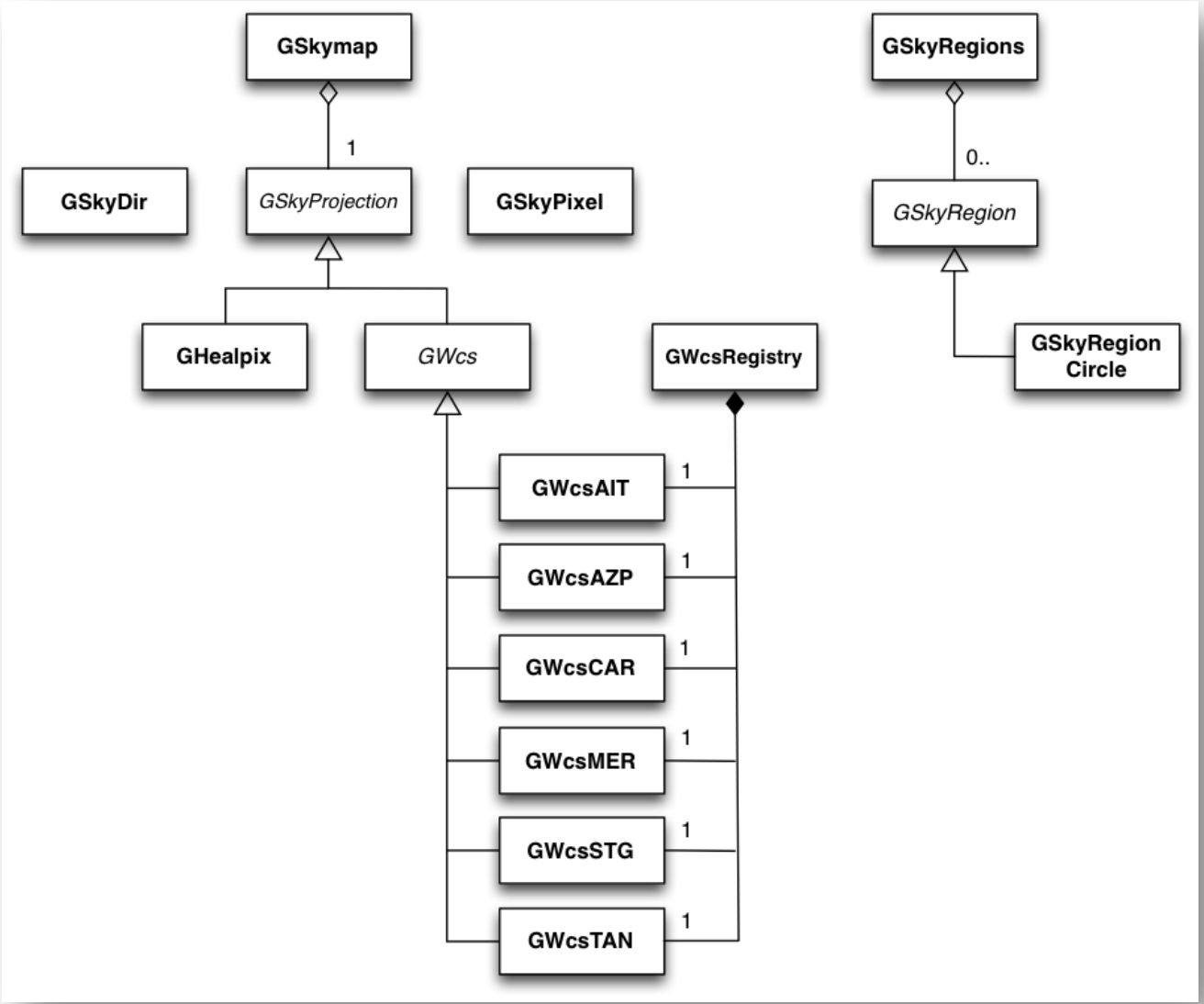
Model handling

http://gammalib.sourceforge.net/user_manual/modules/model.html



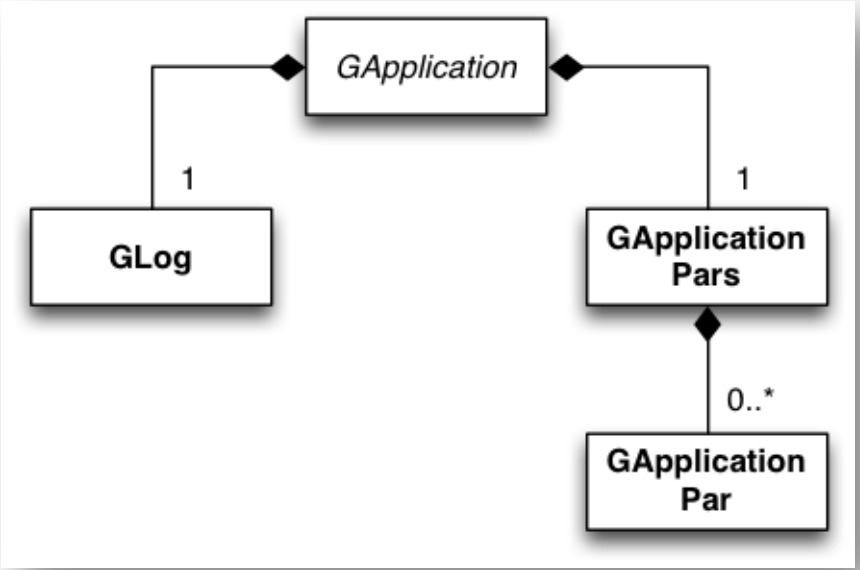
Sky maps, coordinates and regions

http://gammalib.sourceforge.net/user_manual/modules/sky.html



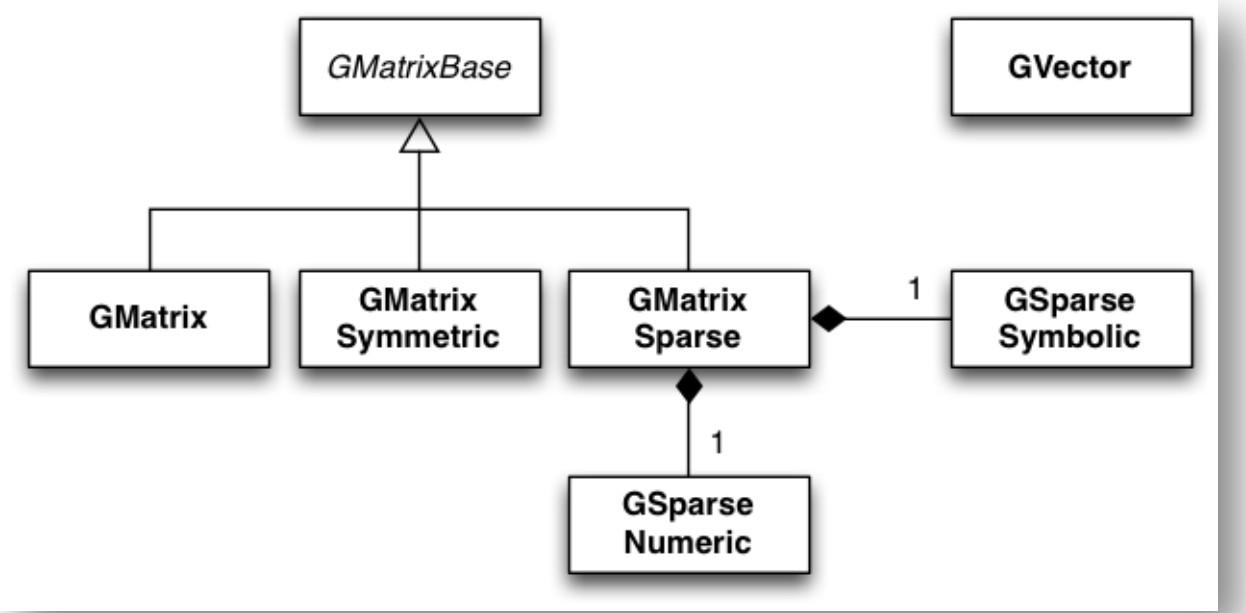
Creation of applications

http://gammalib.sourceforge.net/user_manual/modules/app.html



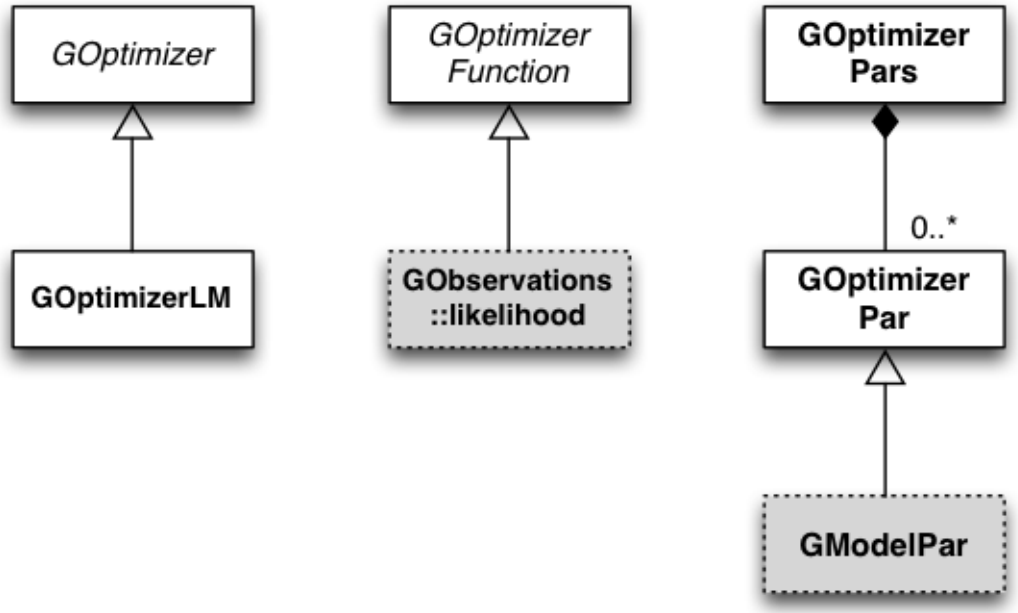
Linear algebra

http://gammalib.sourceforge.net/user_manual/modules/linalg.html



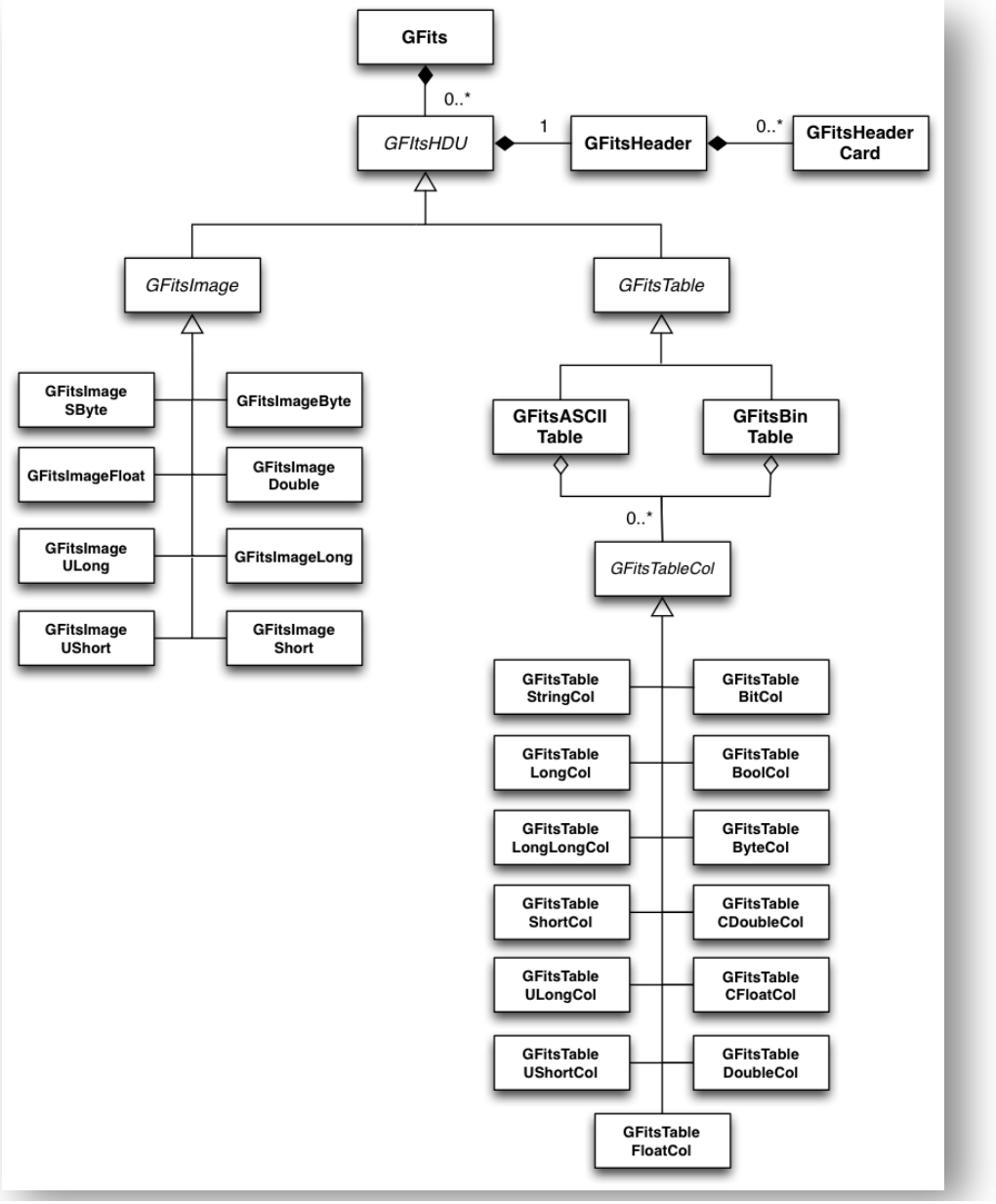
Optimizers

http://gammalib.sourceforge.net/user_manual/modules/opt.html



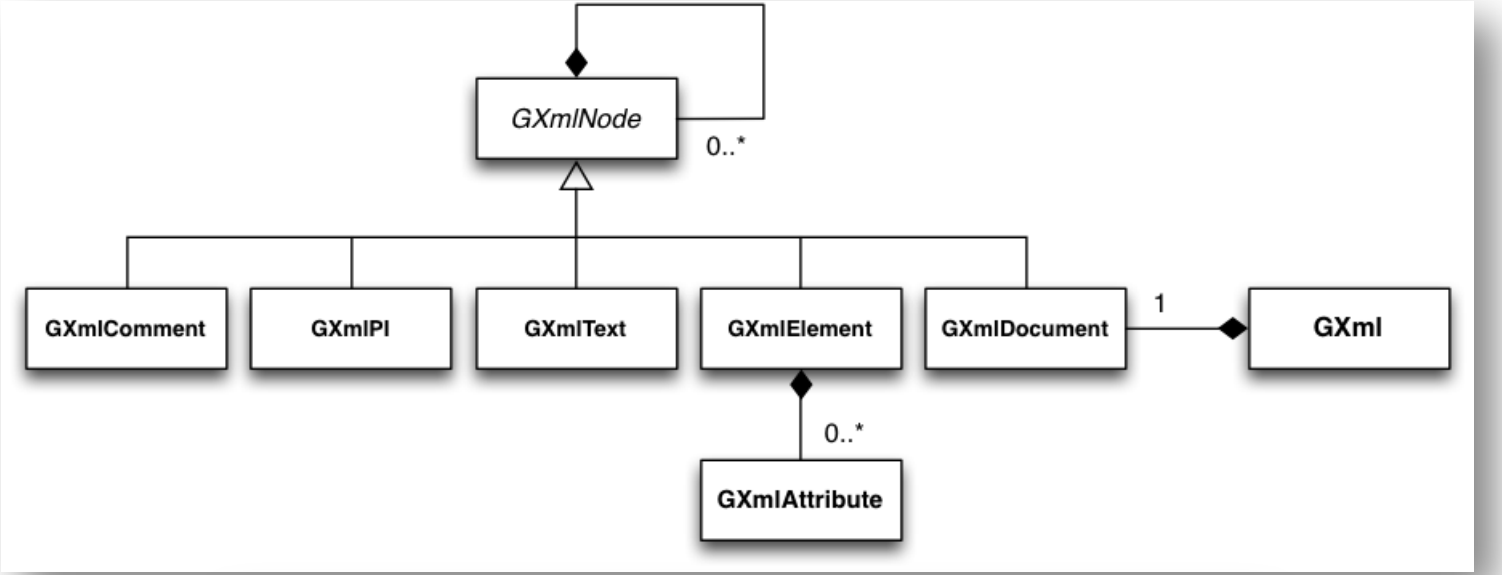
FITS file interface

http://gammalib.sourceforge.net/user_manual/modules/fits.html



XML file interface

http://gammalib.sourceforge.net/user_manual/modules/xml.html



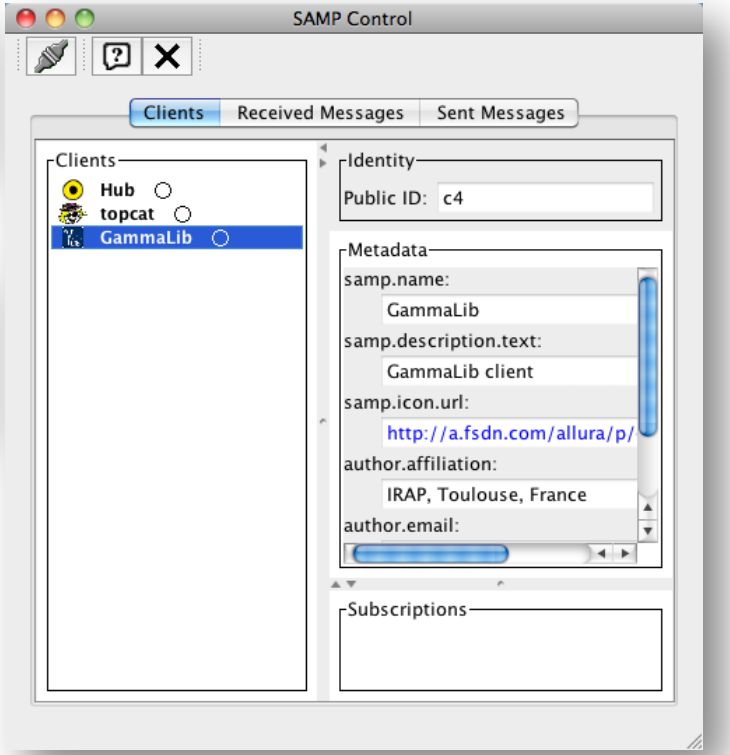

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!--Now 2 nodes with spatial and spectral info-->
<spatial type="Position">
 <parameter ra="83.0" />
 <parameter dec="22.0" />
</spatial>
<spectrum type="PowerLaw">
 <parameter prefactor="1e-7" />
 <parameter index="-2.1" />
</spectrum>
<text>Finish with text</text>
<?process now?>
```

Xspec and virtual observatory interfaces

http://gammalib.sourceforge.net/user_manual/modules/xspec.html

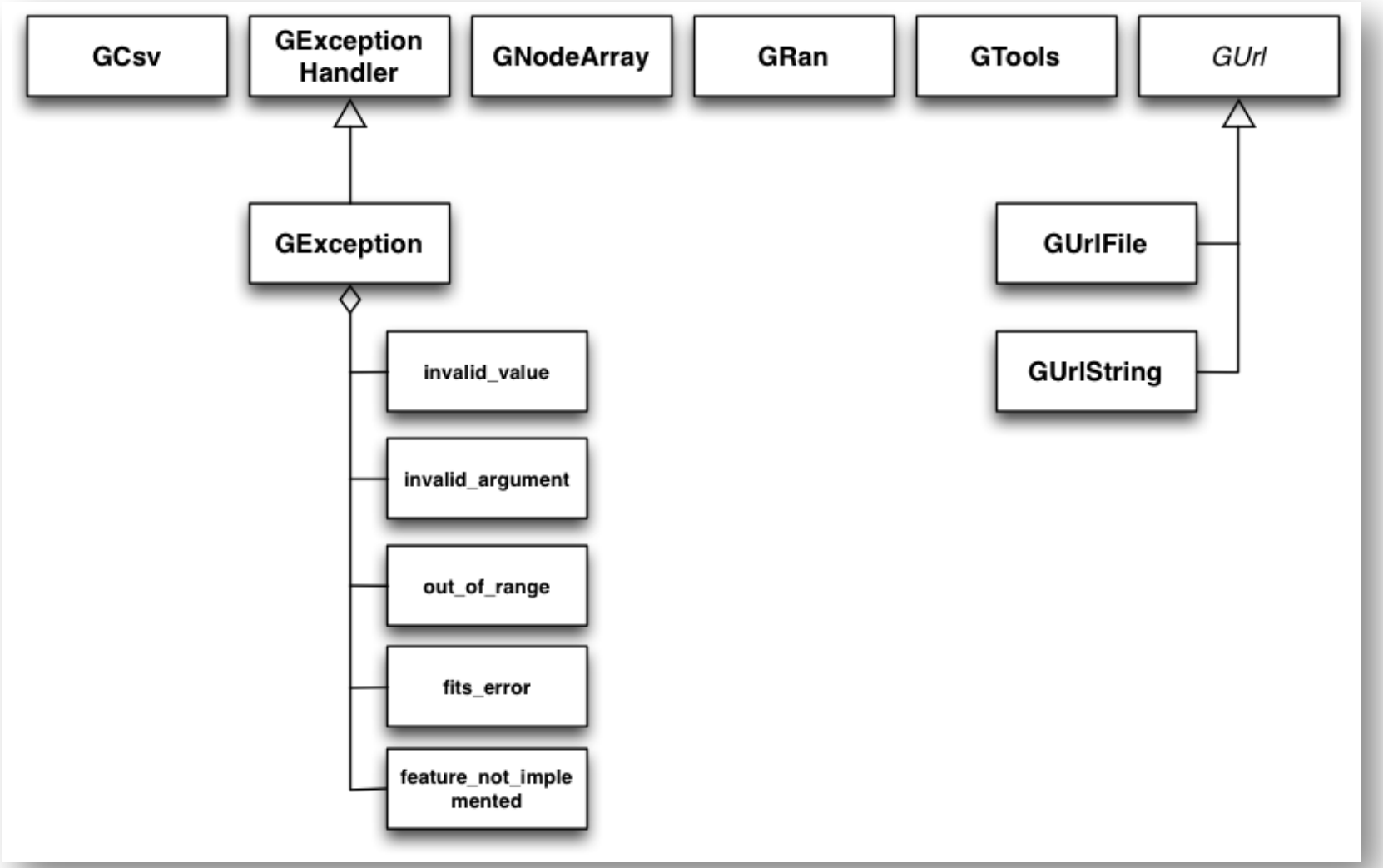
http://gammalib.sourceforge.net/user_manual/modules/vo.html


```
$ python
>>> import gammalib
>>> client=gammalib.GVOClient()
>>> client.connect()
```



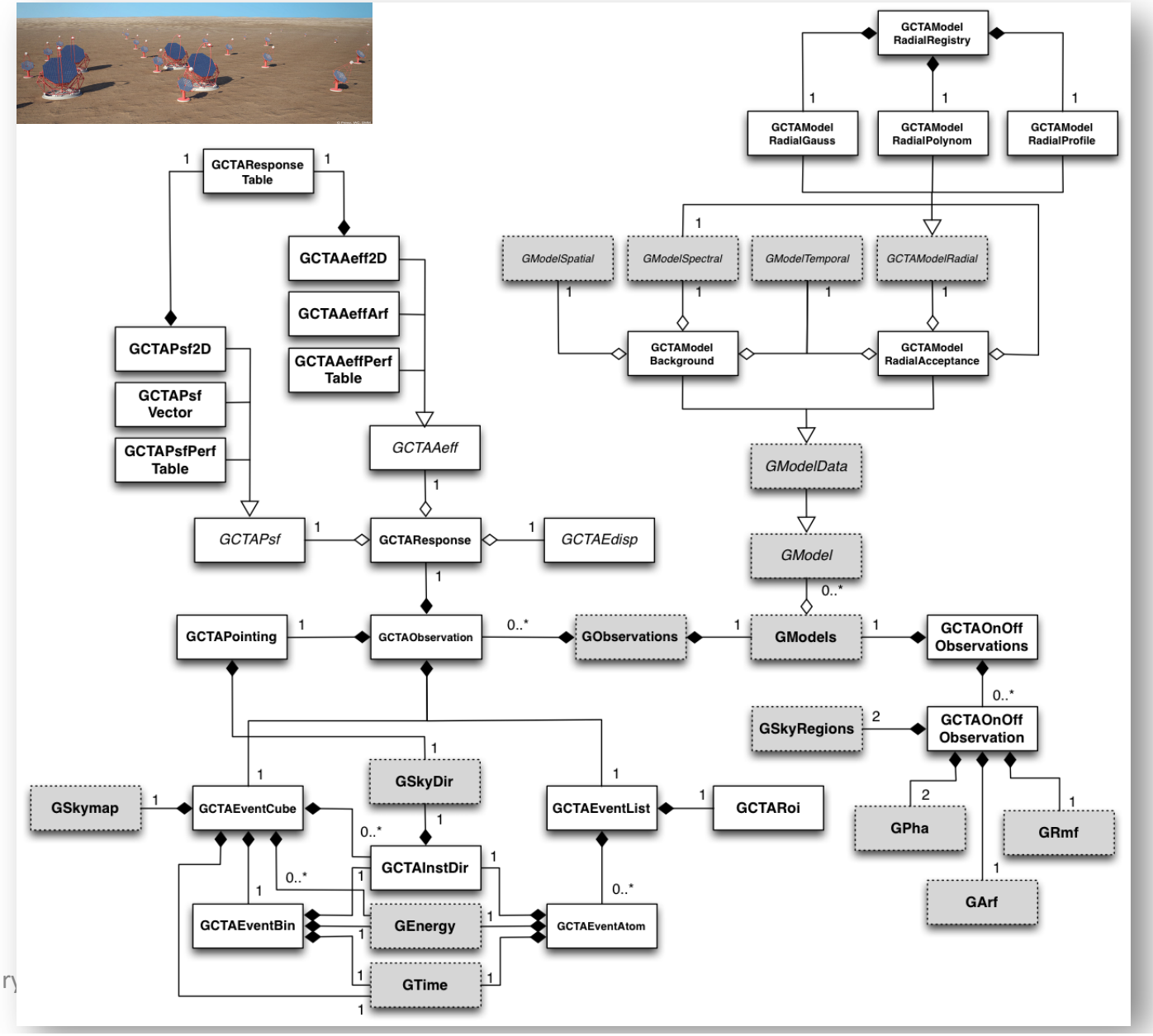
Support functions and classes

http://gammalib.sourceforge.net/user_manual/modules/support.html



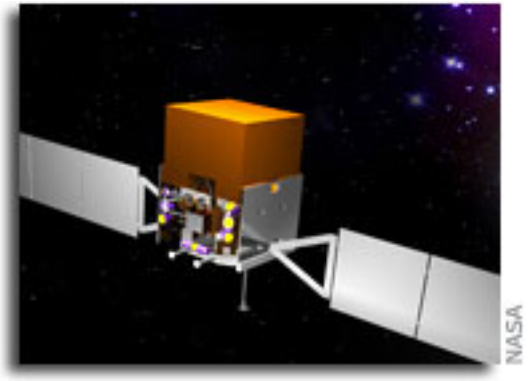
CTA interface

http://gammalib.sourceforge.net/user_manual/modules/cta.html

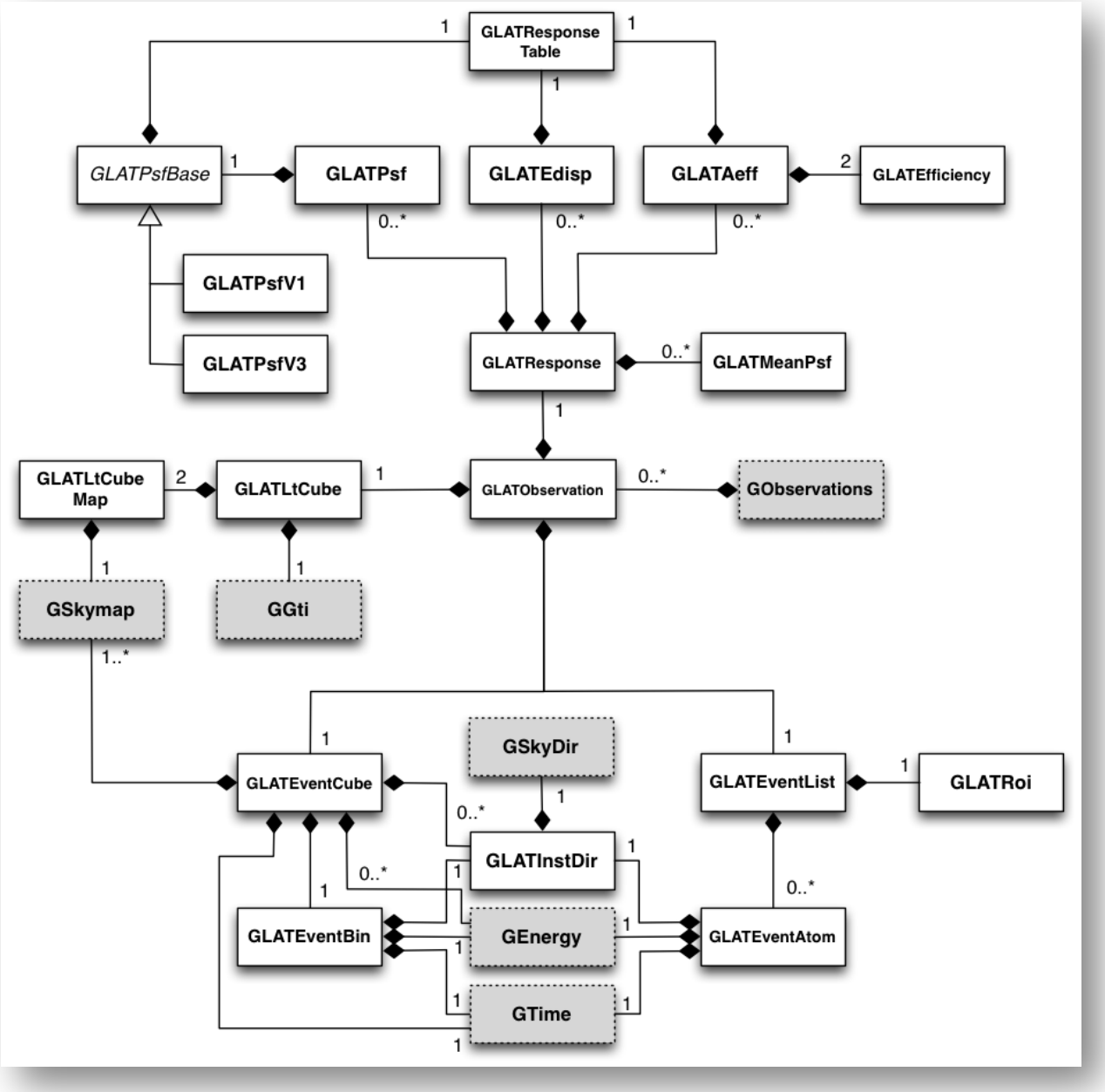


Fermi/LAT interface

http://gammalib.sourceforge.net/user_manual/modules/lat.html

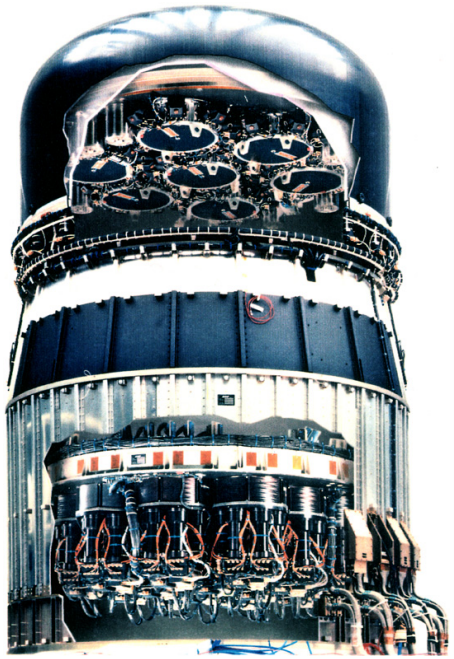


20 MeV – 300 GeV
2008 -

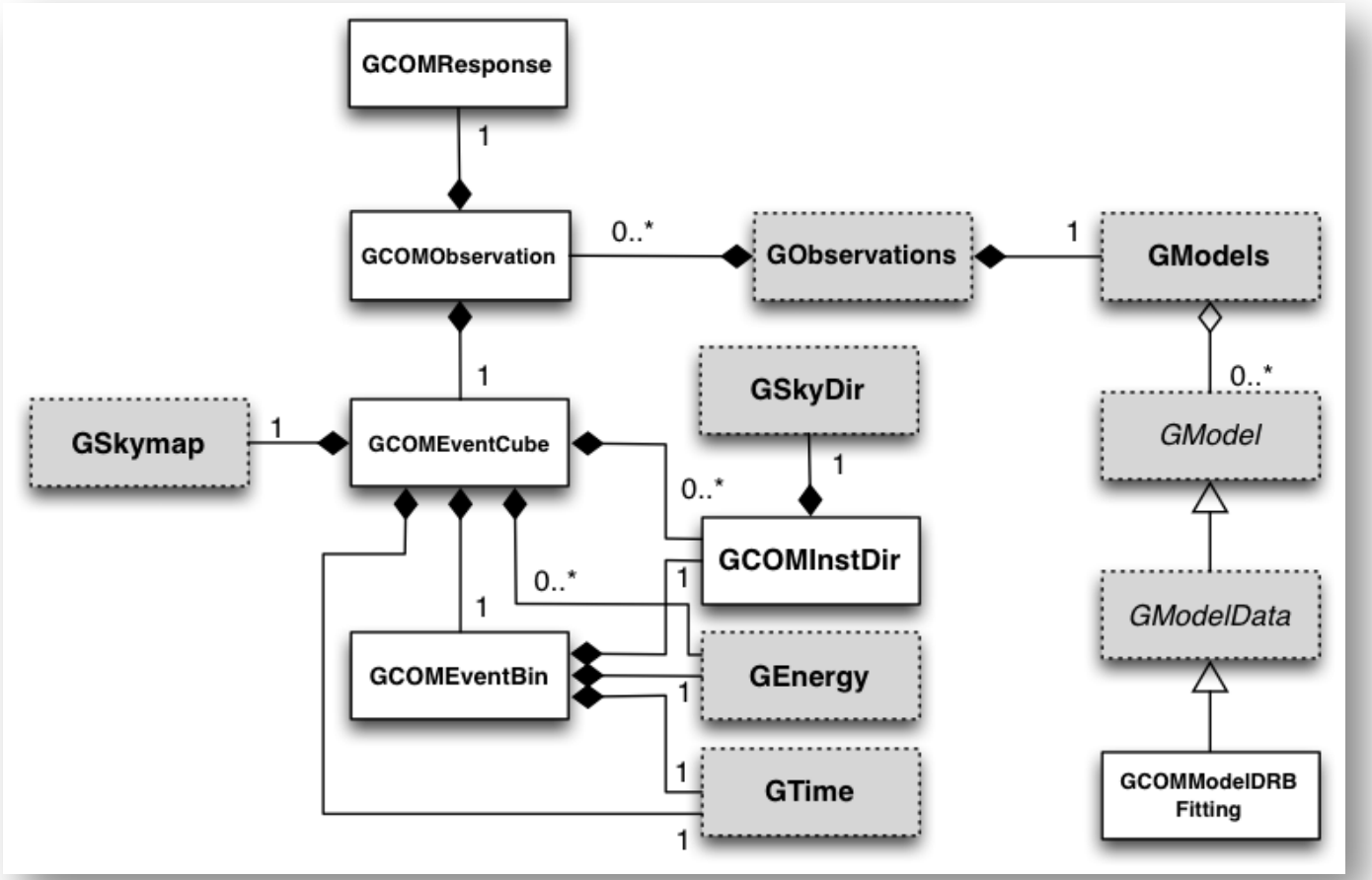


COMPTEL interface

http://gammalib.sourceforge.net/user_manual/modules/comptel.html

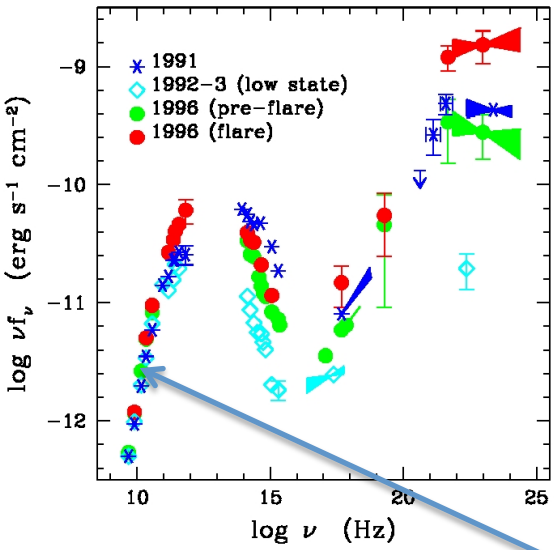


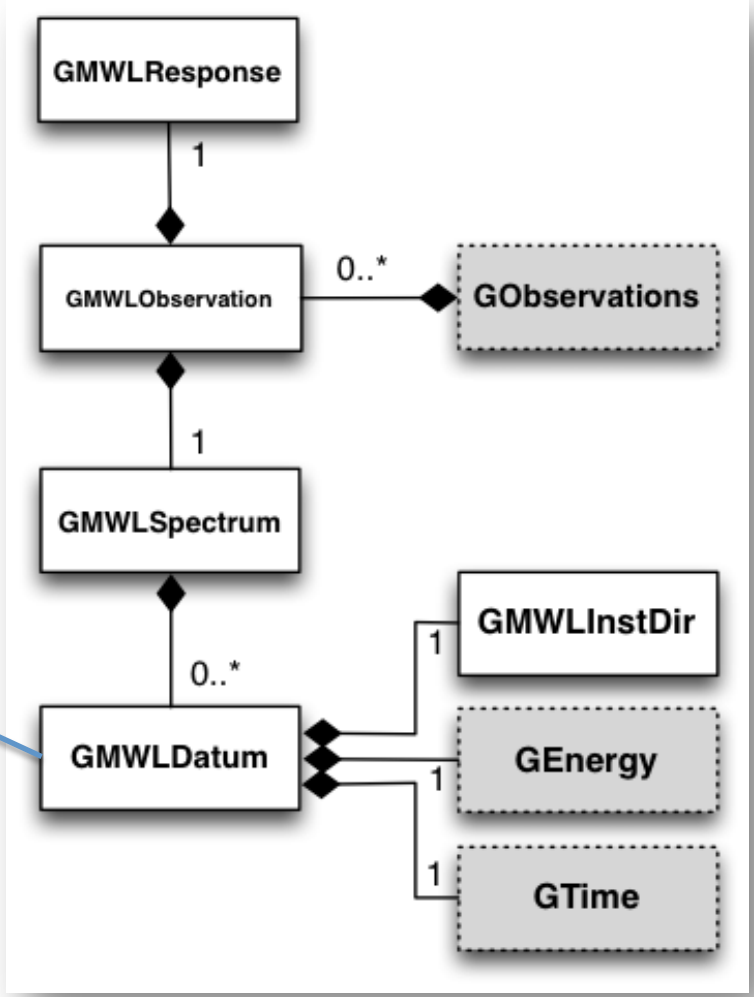
0.75 – 30 MeV
1991 - 2000



Multi-wavelength interface

http://gammalib.sourceforge.net/user_manual/modules/mwl.html





Constants

Constant	Value
`gammalib::pi`	π
`gammalib::twopi`	2π
`gammalib::fourpi`	4π
`gammalib::pihalf`	$\pi/2$
`gammalib::inv_pihalf`	$(\pi/2)^{-1}$
`gammalib::inv_sqrt4pi`	$(\sqrt{4\pi})^{-1}$
`gammalib::pi2`	π^2
`gammalib::deg2rad`	$\pi/180$ (multiplication converts degrees to radians)
`gammalib::rad2deg`	$180/\pi$ (multiplication converts radians to degrees)
`gammalib::ln2`	$\log 2$ (natural logarithm of 2)
`gammalib::ln10`	$\log 10$ (natural logarithm of 10)
`gammalib::inv_ln2`	$(\log 2)^{-1}$
`gammalib::onethird`	$1/3$
`gammalib::twothird`	$2/3$
`gammalib::fourthird`	$4/3$
`gammalib::sqrt_onehalf`	$\sqrt{1/2}$
`gammalib::sqrt_pihalf`	$\sqrt{\pi/2}$;
`gammalib::sqrt_two`	$\sqrt{2}$;

GMath.hpp

Constant	Value
`gammalib::MeV2erg`	1.6021765×10^{-6} (converts MeV to erg)
`gammalib::erg2MeV`	624150.96 (converts erg to MeV)
`gammalib::pc2cm`	$3.08568025 \times 10^{18}$ (converts pc to cm)
`gammalib::sec_in_day`	86400.0 (number of seconds in one day)

GTools.hpp

Functions

GMath.hpp

Function	Description
gammalib::acos	Arc cosine function that avoids NaN due to roundin
gammalib::cosd	Cosine function for argument given in degrees
gammalib::sind	Sine function for argument given in degrees
gammalib::tand	Tangens function for argument given in degrees
gammalib::asind	Arc sine function for argument given in degrees
gammalib::acosd	Arc cosine function for argument given in degrees
gammalib::atand	Arc tangens function for argument given in degree
gammalib::atan2d	Arc tangens function for argument x/y given in deg
gammalib::sincosd	Returns sine and cosine for argument given in deg
gammalib::gammln	Natural logarithm of gamma function
gammalib::modulo	Remainder of division x/y

GTools.hpp

Function	Description
gammalib::strip_whitespace	Strips all leading and trailing whitespace from string.
gammalib::strip_chars	Strips all leading and trailing characters from string.
gammalib::expand_env	Replace any environment variables in string by its value.
gammalib::str	Conversion of C-types to strings.
gammalib::tochar	Conversion of string to `char`.
gammalib::toshort	Conversion of string to `short`.
gammalib::toushort	Conversion of string to `unsigned short`.
gammalib::toint	Conversion of string to `int`.
gammalib::touint	Conversion of string to `unsigned int`.
gammalib::tolong	Conversion of string to `long`.
gammalib::toulong	Conversion of string to `unsigned long`.
gammalib::tolonglong	Conversion of string to `long long`.
gammalib::toulonglong	Conversion of string to `unsigned long long`.
gammalib::tofloat	Conversion of string to `float`.
gammalib::todouble	Conversion of string to `double`.
gammalib::toupper	Conversion of string to upper case letters.
gammalib::tolower	Conversion of string to lower case letters.
gammalib::split	Split string in vector of strings.
gammalib::fill	Fill string with a number of replications of a string.
gammalib::left	Left justify string to achieve a given length of characters.
gammalib::right	Right justify string to achieve a given length of characters.
gammalib::centre	Centre string to achieve a given length of characters.
gammalib::parformat	Format string for parameter value display.
gammalib::plaw_photon_flux	Compute photon flux under a power law.
gammalib::plaw_energy_flux	Compute energy flux under a power law.
gammalib::file_exists	Check whether a file exists.
gammalib::is_infinite	Check whether a double precision value is infinite.
gammalib::is_notanumber	Check whether a double precision value is not a number.
gammalib::contains	Check whether a string contains a sub-string.
gammalib::warning	Dump warning in console.

3. Coding for GammaLib and ctools

- Coding conventions -

Why coding conventions?

From the Java Programming Language, Sun Microsystems:

Code conventions are important to programmers for a number of reasons:

- *40%-80% of the **lifetime cost** of a piece of software goes to maintenance.*
- *Hardly any software is **maintained** for its whole life by the original author.*
- *Code conventions improve the **readability** of the software, allowing engineers to **understand new code more quickly and thoroughly**.*
- *If you ship your source code as a product, you need to make sure it is as **well packaged** and **clean** as any other product you create.*

Is there a unique and best C++ style?

Coding style can affect performance and even code correctness, but there are also rules that mainly affect readability (indentation, placement of brackets, etc.), hence coding style is also a matter of taste (you can certainly argue endless nights about the best coding style).

Take home message:

GammaLib and ctools are both developed following coding conventions. Please follow them as good as you can as they may prevent errors, can lead to better code, and will help newcomers to understand the code base.

General coding rules

(apply to GammaLib and ctools)

Code format

- Blocks are indented by 4 characters
- No tabs, use spaces
- Try to not exceed 80 characters per line
- Separate by spaces, e.g. `int i = 0;`

Use C++98 standard
Do not use C++11 features

Function format

```
int function(void)
{
    int i = 0;
    ...
    return i;
}
```

Curly opening bracket at new line

Block format

Curly opening bracket at end

```
for (int i = 0; i < 10; ++i) {
    sum += i;
}
```

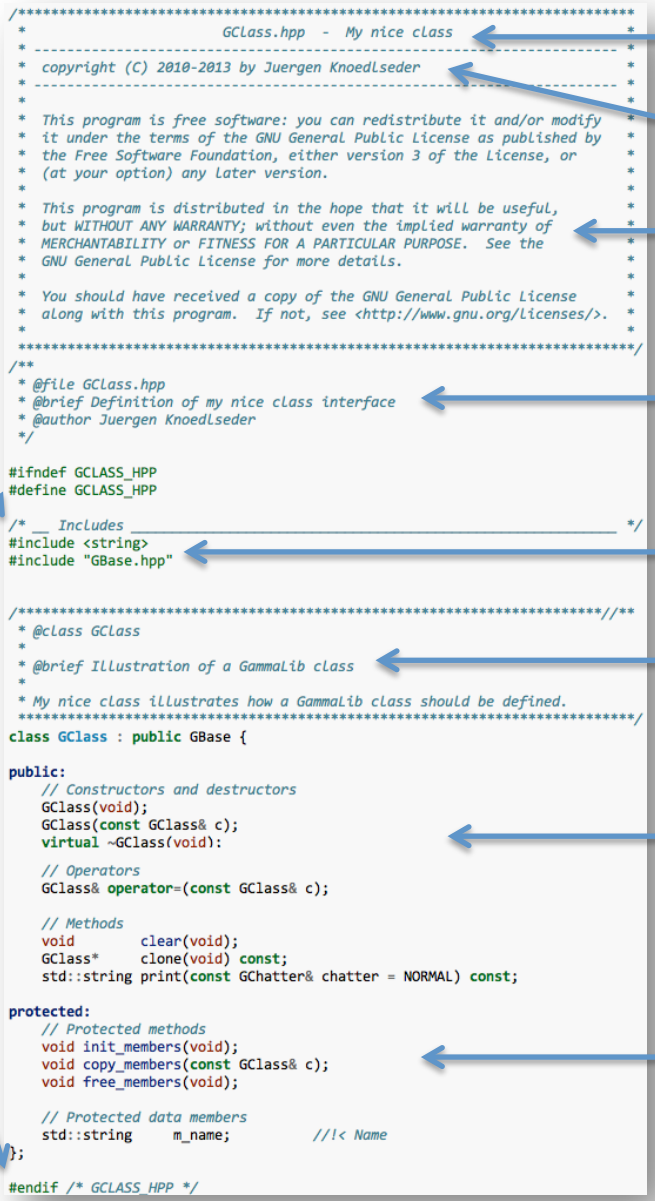
Always use brackets if a block splits over more than a single line

Code alignment

```
void        log10GeV(const double& eng);
void        log10TeV(const double& eng);
std::string print(void) const;
```

```
m_max    = par.m_max;
m_prompt = par.m_prompt;
sum     += par.m_sum;
```

C++ classes (.hpp file) - definition



```
/*************************************************************************
 *                       GClass.hpp  -  My nice class                    *
 * --------------------------------------------------------------------- *
 *  copyright (C) 2010-2013 by Juergen Knoedlseder                       *
 * --------------------------------------------------------------------- *
 *                                                                       *
 *  This program is free software: you can redistribute it and/or modify *
 *  it under the terms of the GNU General Public License as published by *
 *  the Free Software Foundation, either version 3 of the License, or    *
 *  (at your option) any later version.                                  *
 *                                                                       *
 *  This program is distributed in the hope that it will be useful,      *
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of       *
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the        *
 *  GNU General Public License for more details.                         *
 *                                                                       *
 *  You should have received a copy of the GNU General Public License    *
 *  along with this program.  If not, see <http://www.gnu.org/licenses/>.*
 *                                                                       *
 *************************************************************************/
/**
 * @file GClass.hpp
 * @brief Definition of my nice class interface
 * @author Juergen Knoedlseder
 */

#ifndef GCLASS_HPP
#define GCLASS_HPP

/* __ Includes ___________________________________________________________ */
#include <string>
#include "GBase.hpp"


/*************************************************************************//**
 * @class GClass
 *
 * @brief Illustration of a GammaLib class
 *
 * My nice class illustrates how a GammaLib class should be defined.
 *************************************************************************/
class GClass : public GBase {

public:
    // Constructors and destructors
    GClass(void);
    GClass(const GClass& c);
    virtual ~GClass(void);

    // Operators
    GClass& operator=(const GClass& c);

    // Methods
    void        clear(void);
    GClass*     clone(void) const;
    std::string print(const GChatter& chatter = NORMAL) const;

protected:
    // Protected methods
    void init_members(void);
    void copy_members(const GClass& c);
    void free_members(void);

    // Protected data members
    std::string     m_name;          //!< Name
};

#endif /* GCLASS_HPP */
```

File name and short class description

Dates from creation to last editing;
Person who created the file initially

Copyright (GPL 3)

File name, brief description, person who created file (Doxygen syntax)

Includes (C, C++ using < >; GammaLib using " ")

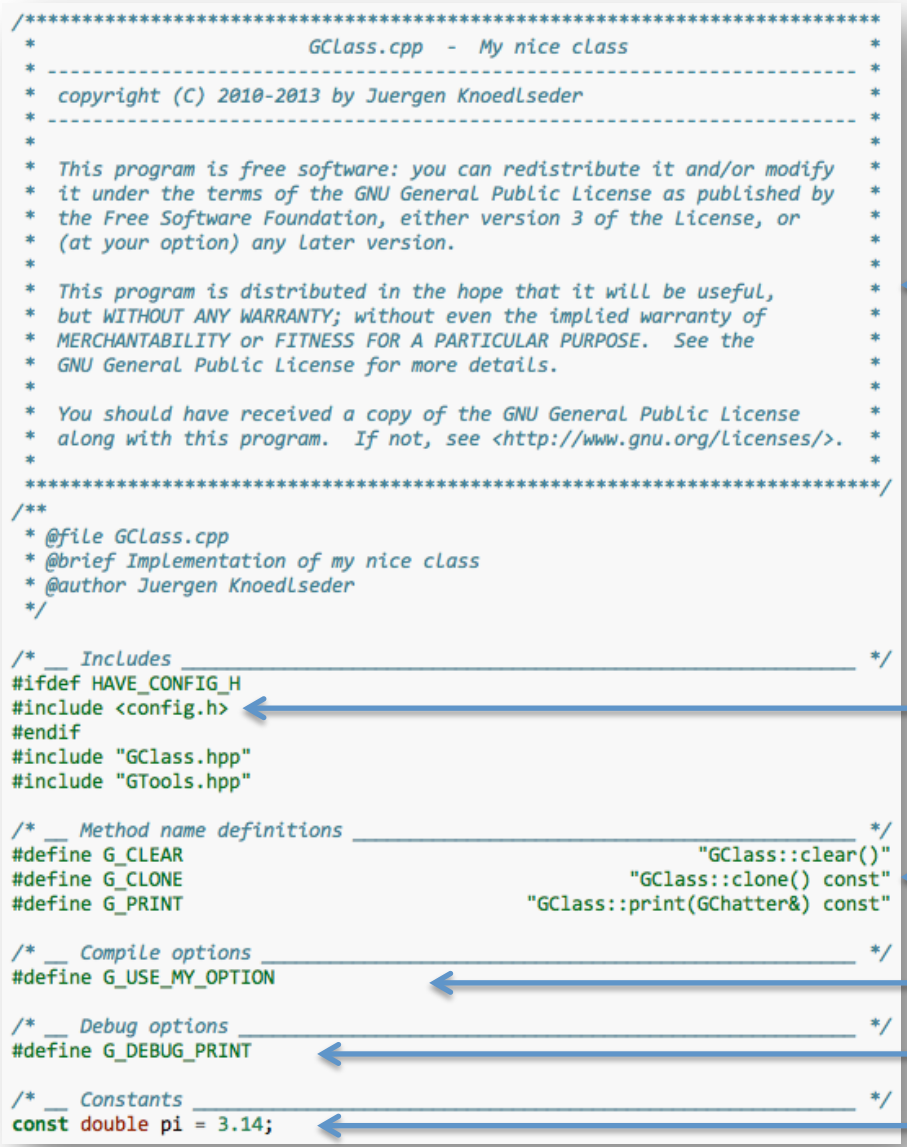
Class description (Doxygen syntax)

Protection

Public
- Constructors
- Operators
- Methods

Protected or private
- Methods
- Members

C++ classes (.cpp file) - implementation



```
/***************************************************************************
 *                        GClass.cpp  -  My nice class                     *
 * ----------------------------------------------------------------------- *
 *  copyright (C) 2010-2013 by Juergen Knoedlseder                         *
 * ----------------------------------------------------------------------- *
 *                                                                         *
 *  This program is free software: you can redistribute it and/or modify   *
 *  it under the terms of the GNU General Public License as published by   *
 *  the Free Software Foundation, either version 3 of the License, or      *
 *  (at your option) any later version.                                    *
 *                                                                         *
 *  This program is distributed in the hope that it will be useful,        *
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of         *
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the          *
 *  GNU General Public License for more details.                           *
 *                                                                         *
 *  You should have received a copy of the GNU General Public License      *
 *  along with this program.  If not, see <http://www.gnu.org/licenses/>.  *
 *                                                                         *
 ***************************************************************************/
/**
 * @file GClass.cpp
 * @brief Implementation of my nice class
 * @author Juergen Knoedlseder
 */

/* __ Includes ___________________________________________________________ */
#ifdef HAVE_CONFIG_H
#include <config.h>
#endif
#include "GClass.hpp"
#include "GTools.hpp"

/* __ Method name definitions ____________________________________________ */
#define G_CLEAR                                    "GClass::clear()"
#define G_CLONE                                  "GClass::clone() const"
#define G_PRINT                        "GClass::print(GChatter&) const"

/* __ Compile options ____________________________________________________ */
#define G_USE_MY_OPTION

/* __ Debug options ______________________________________________________ */
#define G_DEBUG_PRINT

/* __ Constants __________________________________________________________ */
const double pi = 3.14;
```

Header equivalent to .hpp file

Makes compile configuration available to the source code file.

Method names used in exceptions

Compile options

Compile options for debugging

Global constants

Python classes (.i file) - extension

```
/***************************************************************************
 *                        GClass.i  -  My nice class                       *
 * ----------------------------------------------------------------------- *
 *  copyright (C) 2010-2012 by Juergen Knoedlseder                         *
 * ----------------------------------------------------------------------- *
 *                                                                         *
 *  This program is free software: you can redistribute it and/or modify   *
 *  it under the terms of the GNU General Public License as published by   *
 *  the Free Software Foundation, either version 3 of the License, or      *
 *  (at your option) any later version.                                    *
 *                                                                         *
 *  This program is distributed in the hope that it will be useful,        *
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of         *
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the          *
 *  GNU General Public License for more details.                           *
 *                                                                         *
 *  You should have received a copy of the GNU General Public License      *
 *  along with this program.  If not, see <http://www.gnu.org/licenses/>.  *
 *                                                                         *
 ***************************************************************************/
/**
 * @file GClass.i
 * @brief Python interface of my nice class
 * @author Juergen Knoedlseder
 */
%{
/* Put headers and other declarations here that are needed for compilation */
#include "GClass.hpp"
%}

/***********************************************************************//**
 * @class GClass
 *
 * @brief Illustration of a GammaLib class
 *
 * My nice class illustrates how a GammaLib class should be defined.
 ***************************************************************************/
class GClass : public GBase {
public:
    // Constructors and destructors
    GClass(void);
    GClass(const GClass& c);
    virtual ~GClass(void);

    // Methods
    void        clear(void);
    GClass*     clone(void) const;
};


/***********************************************************************//**
 * @brief GClass class extension
 ***************************************************************************/
%extend GClass {
    GClass copy() {
        return (*self);
    }
};
```

Header equivalent to .hpp file

SWIG directive to include corresponding .hpp file (and whatever else is needed for compilation)

Basically a copy of the public class definition from the .hpp file
without:

- operators
- print() method
- const versions of methods

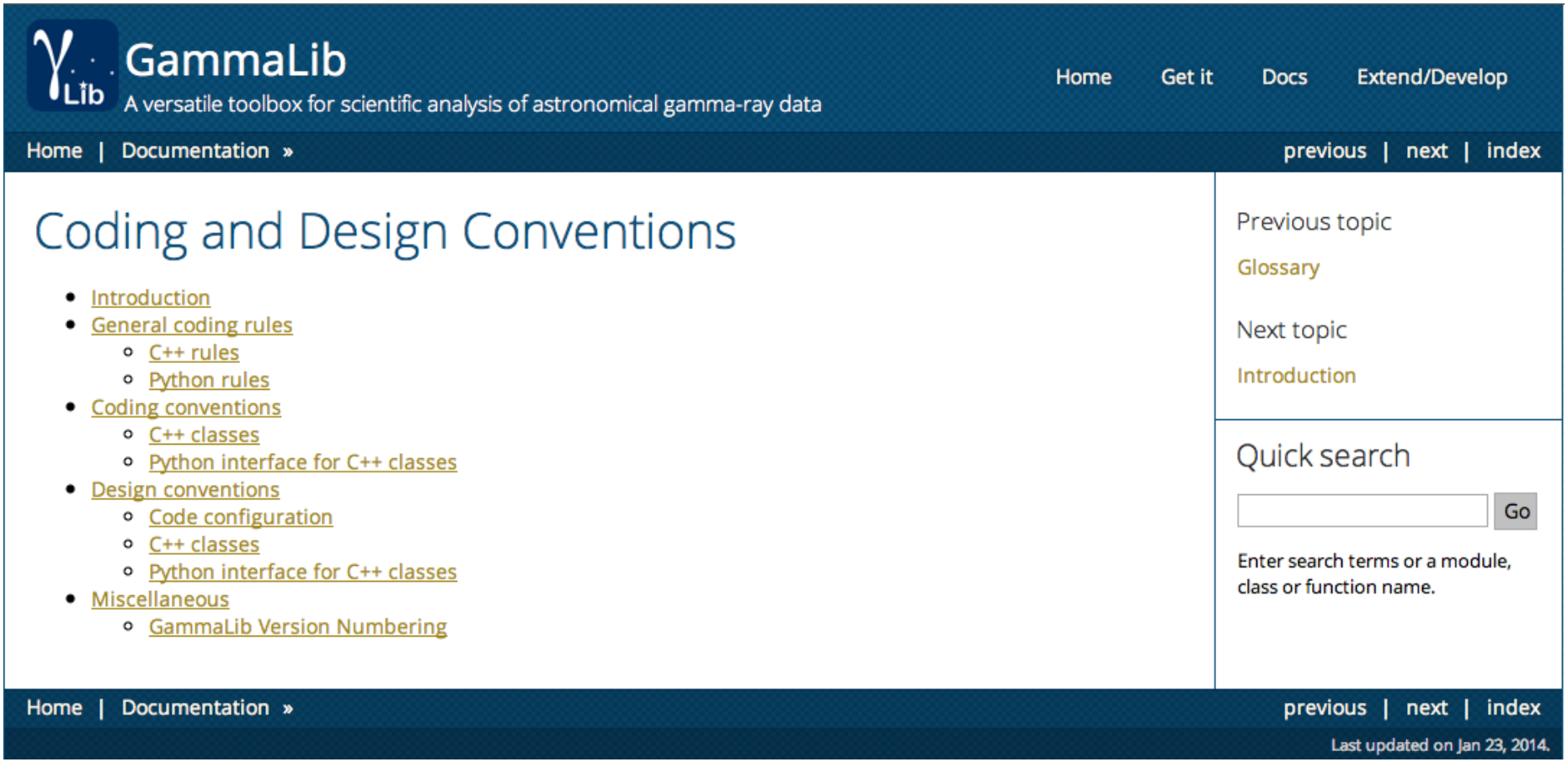
Extensions to the class only available in Python

Add [] operator to Python interface

```
/***************************************************************************//**
 * @brief GObservations class extension
 ******************************************************************************/
%extend GObservations {
    GObservation* __getitem__(const int& index) {
        if (index >= 0 && index < self->size()) {
            return (*self)[index];
        }
        else {
            throw GException::out_of_range("__getitem__(int)", index, self->size());
        }
    }
    void __setitem__(const int& index, const GObservation& val) {
        if (index >= 0 && index < self->size()) {
            self->set(index, val);
            return;
        }
        else {
            throw GException::out_of_range("__setitem__(int)", index, self->size());
        }
    }
};
```

More reading

http://gammalib.sourceforge.net/coding/



Benchmarks

2.66 GHz Intel Core i7 Mac OS X 10.6.8, executing 100000000 (one hundred million) times a given computation (execution times in seconds)

Computation	double	float	Comment
pow(x,2)	1.90	1.19	
x*x	0.49	0.48	**Prefer multiplication over pow(x,2)**
pow(x,2.01)	7.96	4.19	pow is a very time consuming operation
x/a	0.98	1.22	
x*b (where b=1/a)	0.46	0.66	**Prefer multiplication by the inverse over division**
x+1.5	0.40	0.40	
x-1.5	0.49	0.49	**Prefer addition over subtraction**
sin(x)	4.66	2.39	
cos(x)	4.64	2.46	
tan(x)	5.40	2.84	tan is pretty time consuming
acos(x)	2.18	0.94	
sqrt(x)	1.29	1.37	
log10(x)	2.60	2.48	
log(x)	2.72	2.33	
exp(x)	7.17	7.20	exp is a very time consuming operation (comparable to pow)

Top mistake!

Was true 20 years ago, still is true.

Benchmarks

2.66 GHz Intel Core i7 Mac OS X 10.6.8, executing 100000000 (one hundred million) times a given computation (execution times in seconds)

Computation	double	float	Comment
pow(x,2)	1.90	1.19	
x*x	0.49	0.48	**Prefer multiplication over pow(x,2)**
pow(x,2.01)	7.96	4.19	pow is a very time consuming operation
x/a	0.98	1.22	
x*b (where b=1/a)	0.46	0.66	**Prefer multiplication by the inverse over division**
x+1.5	0.40	0.40	
x-1.5	0.49	0.49	**Prefer addition over subtraction**
sin(x)	4.66	2.39	
cos(x)	4.64	2.46	
tan(x)	5.40	2.84	tan is pretty time consuming
acos(x)	2.18	0.94	
sqrt(x)	1.29	1.37	
log10(x)	2.60	2.48	
log(x)	2.72	2.33	
exp(x)	7.17	7.20	exp is a very time consuming operation (comparable to pow)

Top mistake!

Was true 20 years ago, still is true.

Benchmarks for various systems

double precision computations

		32 Bit Intel	64 Bit AMD	64 Bit AMD	64 Bit intel	64 Bit Intel (VM)	64 Bit Intel (VM)
Computation	**Mac OS X**	**galileo**	**kepler**	**dirac**	**fermi**	**CI13 (gcc 4.8.0)**	**CI13 (clang 3.1)**
pow(x,2)	1.90	**5.73**	4.83	3.5	2.65	1.94	1.99
x*x	0.49	0.31	**1.04**	**1.06**	0.5	0.58	0.57
pow(x,2.01)	7.96	10.96	**17.53**	**17.73**	11.11	8.71	8.44
x/a	0.98	1.24	**1.87**	**1.92**	1.03	1.15	1.16
x*b (where b=1/a)	0.46	0.27	**0.99**	**0.99**	0.51	0.54	0.54
x+1.5	0.40	0.27	**0.96**	**1.02**	0.43	0.47	0.47
x-1.5	0.49	0.27	**1.08**	**1.1**	0.57	0.47	0.47
sin(x)	4.66	4.76	**10.46**	**10.44**	6.72	5.62	5.52
cos(x)	4.64	4.68	**10.16**	**10.28**	6.35	5.65	5.62
tan(x)	5.40	6.27	**15.23**	**15.4**	8.61	8.11	7.98
acos(x)	2.18	**9.57**	7.49	7.75	4.48	3.86	2.93
sqrt(x)	1.29	**3.29**	2.33	2.4	0.97	2.02	1.84
log10(x)	2.60	5.33	**12.91**	**12.58**	7.71	6.54	6.47
log(x)	2.72	5.15	**10.64**	**10.66**	6.32	5.26	5.09
exp(x)	7.17	**10**	4.78	4.8	1.85	2.03	2.02

Benchmarks for various systems

single precision computations

		32 Bit Intel	64 Bit AMD	64 Bit AMD	64 Bit intel	64 Bit Intel (VM)	64 Bit Intel (VM)
Computation	**Mac OS X**	**galileo**	**kepler**	**dirac**	**fermi**	**CI13 (gcc 4.8.0)**	**CI13 (clang 3.1)**
pow(x,2)	1.19	1.77	**3.27**	3	1.35	1.54	0.9
x*x	0.48	0.3	**0.99**	1	0.47	0.54	0.54
pow(x,2.01)	4.19	10.64	**29.81**	**30.21**	14.42	13	12.29
x/a	1.22	1.24	**2.77**	**2.79**	1.2	1.37	1.4
x*b (where b=1/a)	0.66	0.27	**1.72**	**1.74**	0.67	0.76	0.79
x+1.5	0.40	0.27	**1.03**	**1.04**	0.4	0.46	0.47
x-1.5	0.49	0.27	**1.13**	**1.14**	0.54	0.47	0.47
sin(x)	2.39	4.92	**116.41**	**119.06**	54	41.2	40.22
cos(x)	2.46	4.85	**116.47**	**119.27**	53.93	40.91	40.3
tan(x)	2.84	6.47	**120.69**	**122**	55.14	42.36	41.83
acos(x)	0.94	**9.02**	8.6	8.71	3.86	2.81	2.38
sqrt(x)	1.37	2.27	**3.77**	**3.75**	1.5	1.84	1.55
log10(x)	2.48	4.15	**12.74**	**12.59**	6.28	5.74	4.97
log(x)	2.33	3.83	**10.07**	**10.42**	5.16	4.88	4.11
exp(x)	7.20	9.96	**17.51**	**18.32**	10.77	10.21	10.18

Benchmarks lessons

https://cta-redmine.irap.omp.eu/projects/gammalib/wiki/Computation_Benchmarks

Use double precision

Double precision computations are generally not slower than single precision computations, but on some systems, single precision are much slower than doubles.

Never use pow(x, 2), always use x*x

This brings typically a factor of 4 in speed increase.

Don't: y = pow(x,2)
Do: y = x*x

If possible, use multiplication instead of division

This brings typically a factor of 2 in speed increase.

Don't: y = x/2.0
Do: y = 0.5*x

Use std::sin(), std::cos(), etc. instead of sin(), cos(), etc.

Execution speed of trigonometric function depends on how they are called because the compiler generates different codes; the std:: versions have never shown to be slow.

… and of course: *never make the same operation twice! Break down your formula into pieces and store them in variables that can be re-used.*

… and: *if you want speed, buy a Mac* ☺

3. Coding for GammaLib and ctools

- Testing -

Unit testing

Unit Tests Coverage	Unit test success
64,1%	**100,0%**
62,4% line coverage	0 failures
69,8% branch coverage	0 errors
	4 765 tests
	6:56 min

Code testing is an integrated feature of gammalib (and ctools), but not all code is yet covered …

`make check` →

```
Test event bin: ............ ok
Test event cube: ..................... ok
Test binned optimizer: ..................................................................................................
PASS: test_COM

****************************
* Python interface testing *
****************************
Test GLog: ..................... ok
Test GApplicationPars: .. ok
Test GFits: ............... ok
Test GMatrix: ................ ok
Test GMatrixSparse: ................. ok
Test GMatrixSymmetric: .......... ok
Model module dummy test: . ok
Numerics module dummy test: . ok
Observation module dummy test: . ok
Optimizer module dummy test: . ok
Test HEALPix map: ..................................................................................................... ok
Test AIT projection map: ..................................................... ok
Test AZP projection map: ..................................................... ok
Test CAR projection map: ..................................................... ok
Test MER projection map: ..................................................... ok
Test STG projection map: ..................................................... ok
Test TAN projection map: ..................................................... ok
Test FK5 to Galactic coordinate conversion: .. ok
Test GNodeArray: ..................... ok
Test GUrlFile: ... ok
Test GUrlString: ... ok
Test module dummy test: . ok
XML module dummy test: . ok
Test GPha: ... ok
Test GArf: ... ok
Test GRmf: ..... ok
MWL dummy test: . ok
Test CTA effective area classes: ................ ok
Test CTA PSF classes: ..................... ok
Test CTA ON/OFF analysis: .... ok
LAT dummy test: . ok
COMPTEL dummy test: . ok
PASS: test_python.py
====================
All 20 tests passed
====================
```

Each dot is an individual test case:
. = okay
F = failure (unexpected result)
E = error (unexpected behaviour, e.g. seg. fault)

Note: for automake >= 1.13, console dumps end up in `test/test_*.log`

Unit testing

Unit Tests Coverage	Unit test success
64,1%	**100,0%**
62,4% line coverage	0 failures
69,8% branch coverage	0 errors
	4 765 tests
	6:56 min

Code testing is an integrated feature of gammalib (and ctools), but not all code is yet covered …

`make check` →

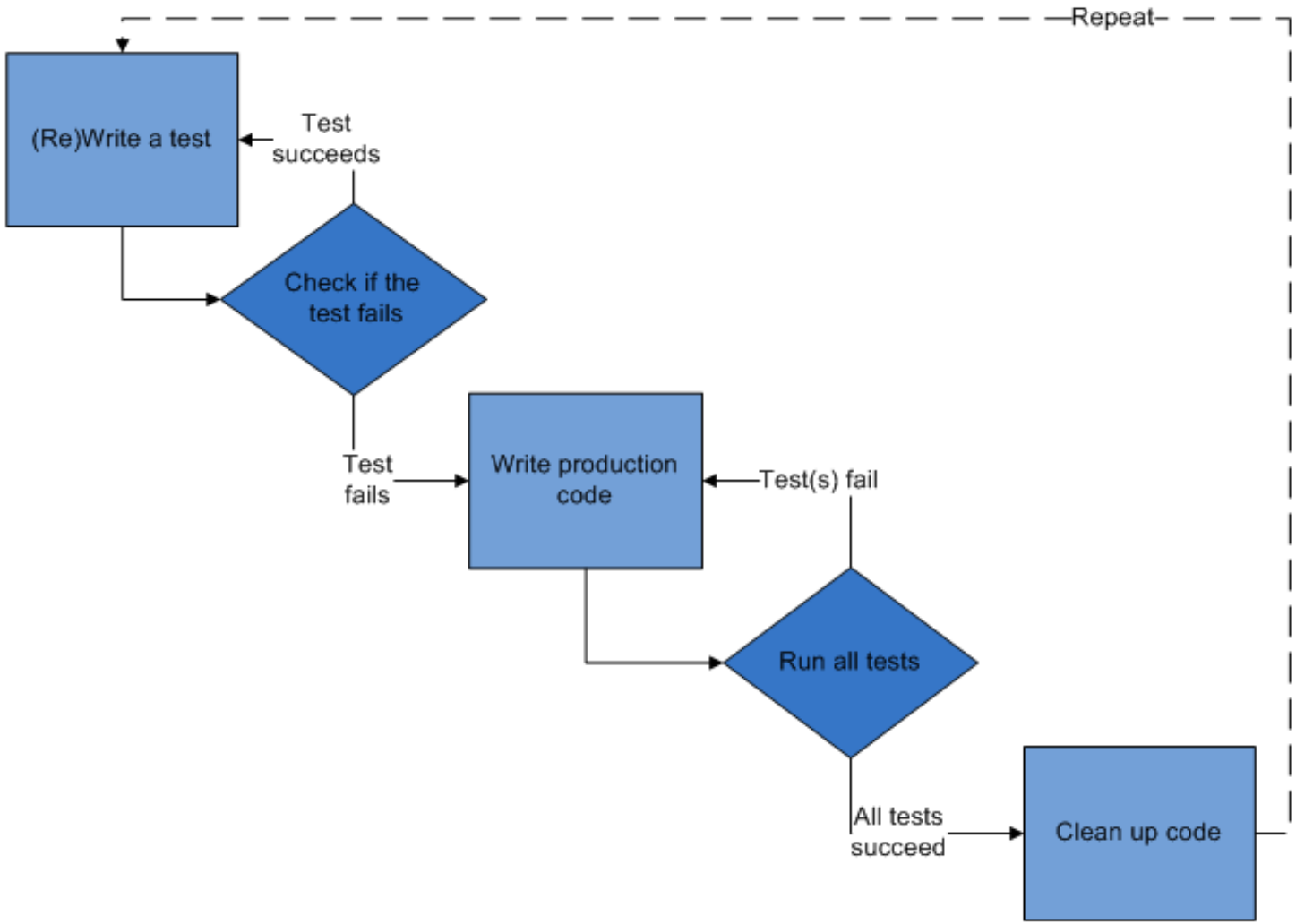
```
Test event bin: ............ ok
Test event cube: ..................... ok
Test binned optimizer: ..................................................................................................
PASS: test_COM

****************************
* Python interface testing *
****************************
Test GLog: ..................... ok
Test GApplicationPars: .. ok
Test GFits: ............... ok
Test GMatrix: ................ ok
Test GMatrixSparse: ................. ok
Test GMatrixSymmetric: .......... ok
Model module dummy test: . ok
Numerics module dummy test: . ok
Observation module dummy test: . ok
Optimizer module dummy test: . ok
Test HEALPix map: ..................................................................................................... ok
Test AIT projection map: ..................................................... ok
Test AZP projection map: ..................................................... ok
Test CAR projection map: ..................................................... ok
Test MER projection map: ..................................................... ok
Test STG projection map: ..................................................... ok
Test TAN projection map: ..................................................... ok
Test FK5 to Galactic coordinate conversion: .. ok
Test GNodeArray: ..................... ok
Test GUrlFile: ... ok
Test GUrlString: ... ok
Test module dummy test: . ok
XML module dummy test: . ok
Test GPha: ... ok
Test GArf: ... ok
Test GRmf: ..... ok
MWL dummy test: . ok
Test CTA effective area classes: ................. ok
Test CTA PSF classes: ...................... ok
Test CTA ON/OFF analysis: .... ok
LAT dummy test: . ok
COMPTEL dummy test: . ok
PASS: test_python.py
====================
All 20 tests passed
====================
```

Each dot is an individual test case:
. = okay
F = failure (unexpected result)
E = error (unexpected behaviour, e.g. seg. fault)

Note: for automake >= 1.13, console dumps end up in `test/test_*.log`

Test driven development

You may give it a try …



How to write a new C++ unit test? As C++ class!

(see `inst/test/test_CTA.hpp` and `inst/test/test_CTA.cpp`)

Create a class that derived from GTestSuite

```
class TestGCTAResponse : public GTestSuite {
public:
    // Constructors and destructors
    TestGCTAResponse(void) : GTestSuite() {}
    virtual ~TestGCTAResponse(void) {}

    // Methods
    virtual void                set(void);
    virtual TestGCTAResponse* clone(void) const;
    void                        test_response_aeff(void);
    void                        test_response_psf(void);
    void                        test_response_psf_king(void);
    void                        test_response_npsf(void);
    void                        test_response_irf_diffuse(void);
    void                        test_response_npred_diffuse(void);
    void                        test_response(void);
};
```

Implement set method

```
void TestGCTAResponse::set(void)
{
    // Set test name
    name("GCTAResponse");

    // Append tests to test suite
    append(static_cast<pfunction>(&TestGCTAResponse::test_response), "Test resp
    append(static_cast<pfunction>(&TestGCTAResponse::test_response_aeff), "Test
    append(static_cast<pfunction>(&TestGCTAResponse::test_response_psf), "Test
    append(static_cast<pfunction>(&TestGCTAResponse::test_response_psf_king), "Test King profile PSF");
    append(static_cast<pfunction>(&TestGCTAResponse::test_response_npsf), "Test integrated PSF");
    append(static_cast<pfunction>(&TestGCTAResponse::test_response_irf_diffuse), "Test diffuse IRF");
    append(static_cast<pfunction>(&TestGCTAResponse::test_response_npred_diffuse), "Test diffuse IRF integration");

    // Return
    return;
}
```

Append GTestSuite to container, run tests and save results

```
int main(void)
{
    // Allocate test suit container
    GTestSuites testsuites("CTA instrument specific class testing");

    // Check if data directory exists
    bool has_data = (access(datadir.c_str(), R_OK) == 0);
    if (has_data) {
        std::string caldb = "CALDB="+cta_caldb;
        putenv((char*)caldb.c_str());
    }

    // Initially assume that we pass all tests
    bool success = true;

    // Create test suites and append them to the container
    TestGCTAResponse        rsp;
    TestGCTAObservation     obs;
    TestGCTAModelBackground bck;
    TestGCTAOptimize        opt;
    testsuites.append(rsp);
    if (has_data) {
        testsuites.append(bck);
        testsuites.append(obs);
        testsuites.append(opt);
    }

    // Run the testsuites
    success = testsuites.run();

    // Save test report
    testsuites.save("reports/GCTA.xml");

    // Return success status
    return (success ? 0 : 1);
}
```

And in Python?

Create class derived from GPythonTestSuite

```
# ================================== #
# Test class for GammaLib CTA module #
# ================================== #
class Test(GPythonTestSuite):
    """
    Test class for GammaLib CTA module.
    """
    # Constructor
    def __init__(self):
        """
        Constructor.
        """
        # Call base class constructor
        GPythonTestSuite.__init__(self)

        # Return
        return

    # Set test functions
    def set(self):
        """
        Set all test functions.
        """
        # Set test name
        self.name("CTA")

        # Append tests
        self.append(self.test_aeff, "Test CTA effective area classes")
        self.append(self.test_psf, "Test CTA PSF classes")
        self.append(self.test_onoff, "Test CTA ON/OFF analysis")

        # Return
        return
```

Allocate test class, append to container, run tests and save results

```
# ======================== #
# Main routine entry point #
# ======================== #
if __name__ == '__main__':
    """
    Perform unit testing for Python interface.
    """
    # Allocate test suites
    suites = GTestSuites("Python interface testing")

    # Allocate test suite and append them to the container
    suite_cta = test_CTA.Test()
    suite_cta.set()
    suites.append(suite_cta)

    # Run test suite
    success = suites.run()

    # Save test results
    suites.save("reports/GPython.xml")

    # Set return code
    if success:
        rc = 0
    else:
        rc = 1

    # Exit with return code
    sys.exit(rc)
```

Why can't I see the test results?

Test results are written as XML files that can be found in the test/reports directory. XML files are in JUnit format that can be exploited by Jenkins and Sonar.

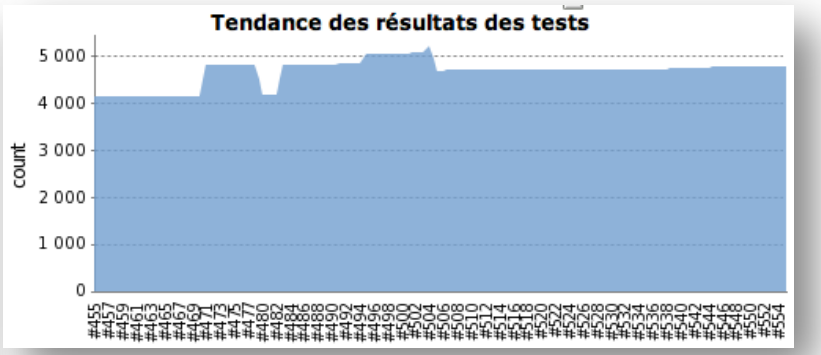
Résultats des tests : (root)

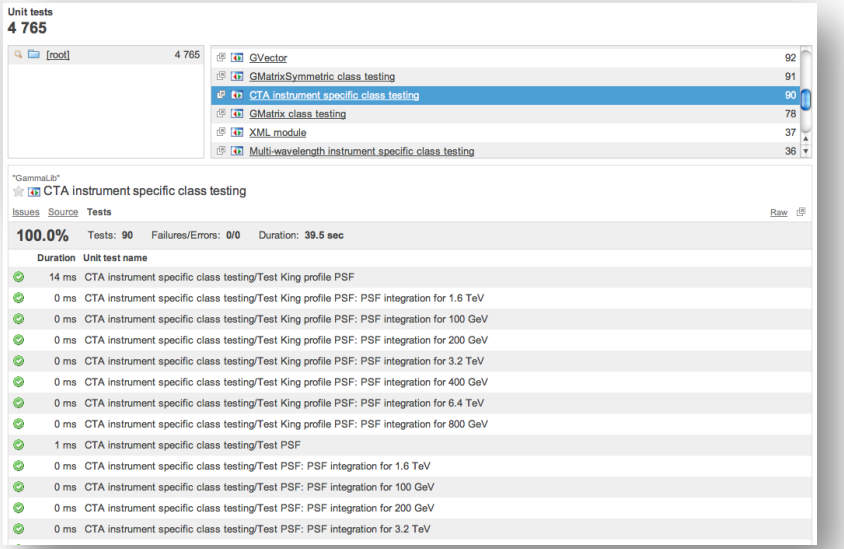
0 échecs (±0)

4 765 tests (±0)
A pris 1 mn 19 s.
ajouter une description

Tous les tests

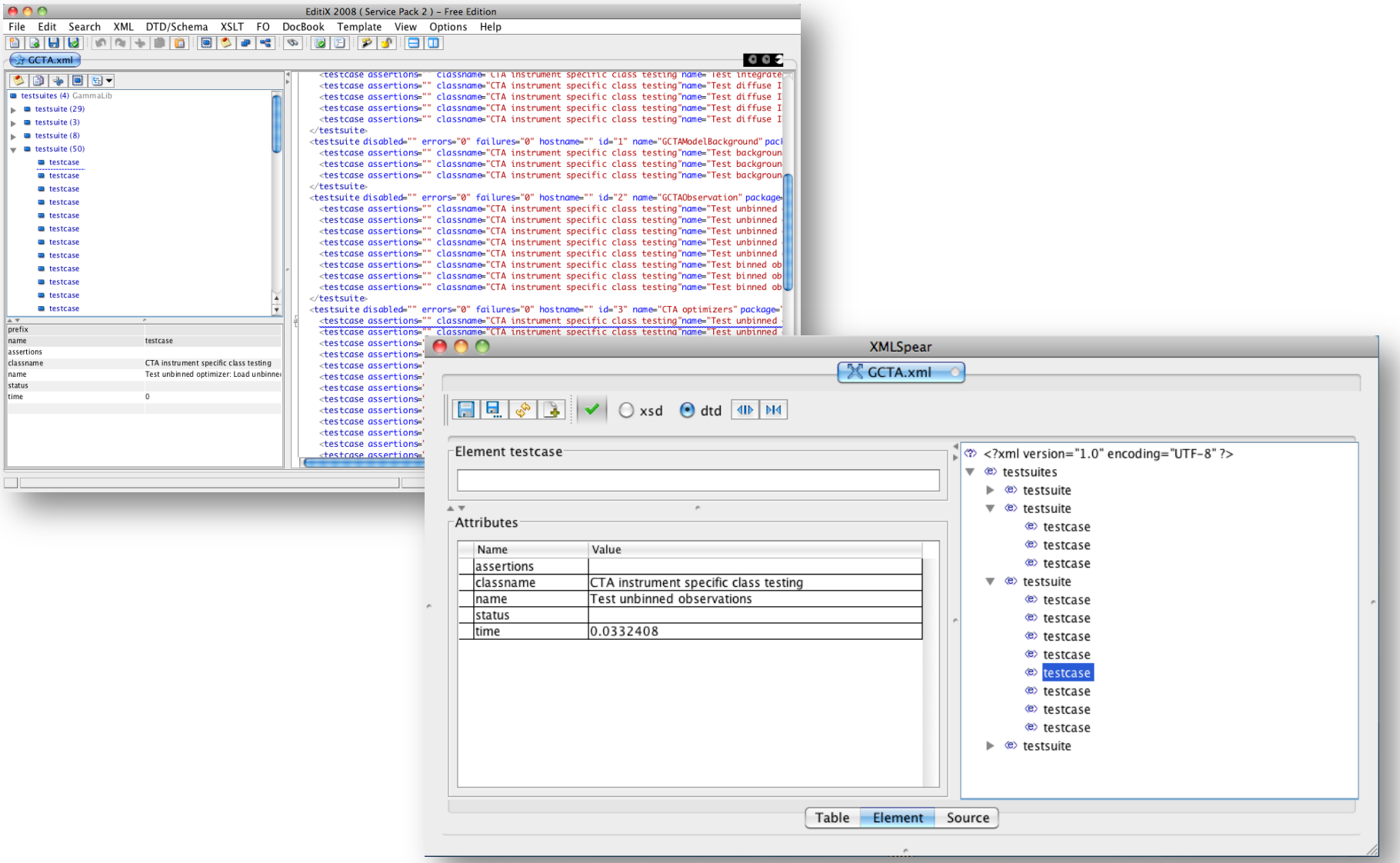
Class	Durée	Échec	(diff)	Passé	(diff)	Total	(diff)
COMPTEL instrument specific class testing	42 s	0		0		166	
CTA instrument specific class testing	12 s	0		0		90	
FITS module	93 ms	0		0		2091	
GApplication	6 ms	0		0		18	
GMatrix class testing	2 ms	0		0		78	
GMatrixSparse class testing	9 ms	0		0		121	
GMatrixSymmetric class testing	2 ms	0		0		91	
GModel	1,1 s	0		0		680	
GNumerics	0 ms	0		0		5	
GSky	1,4 s	0		0		126	
GVector	14 ms	0		0		92	
LAT instrument specific class testing	19 s	0		0		102	
Multi-wavelength instrument specific class testing	38 ms	0		0		36	
Observation module	1,1 s	0		0		254	
Optimizer module	0,6 s	0		0		6	
Python interface testing	0,18 s	0		0		32	
Support module	15 ms	0		0		117	
VO module	2 ms	0		0		3	
XML module	12 ms	0		0		37	
Xspec module	13 ms	0		0		620	





Disclaimer: the unit test suite has been geared towards code quality assurance (continuous integration) and not towards the user or the developer. Additional support for developers is probably needed …

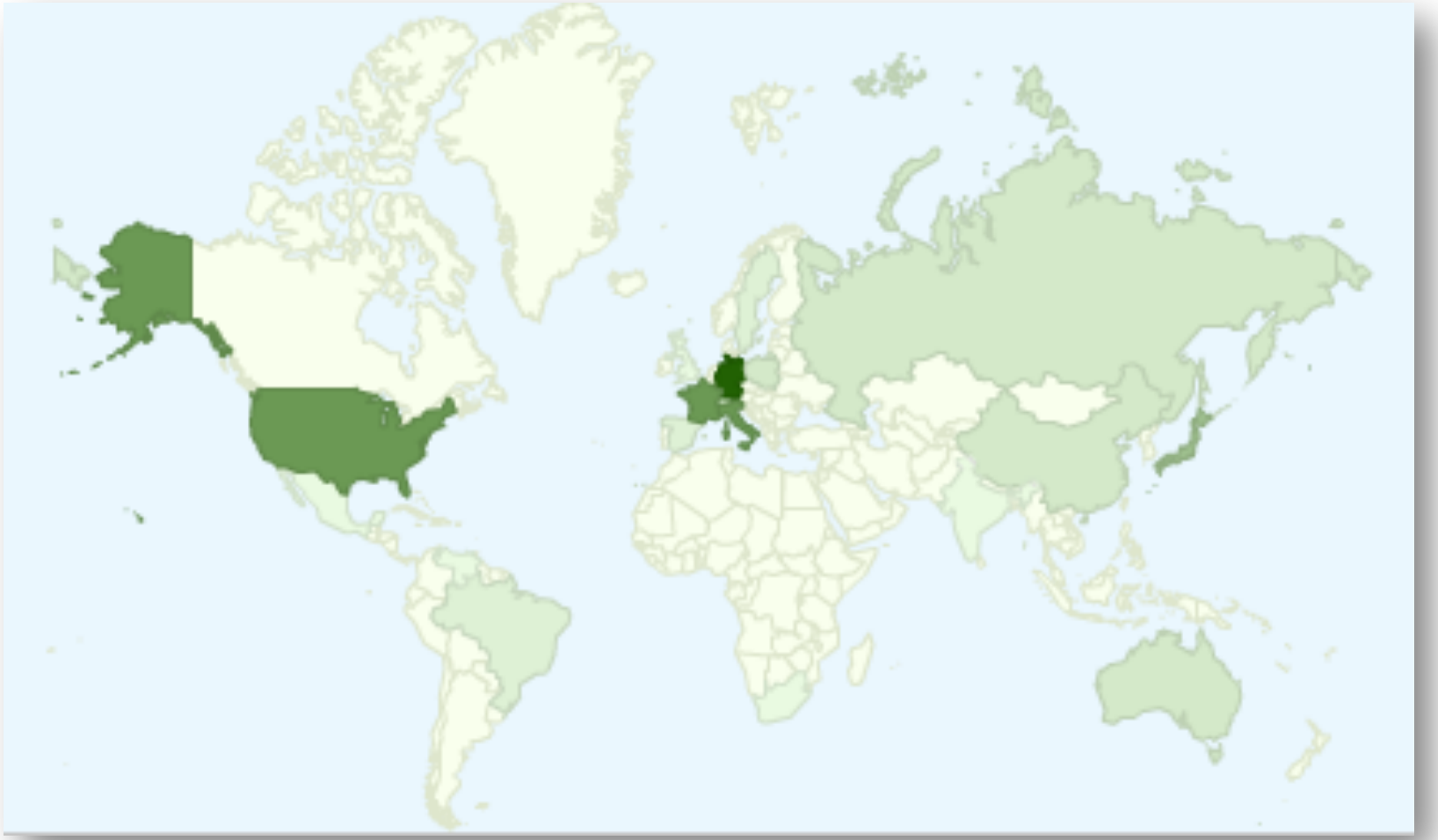
Test result display with XML editors



4. Current status

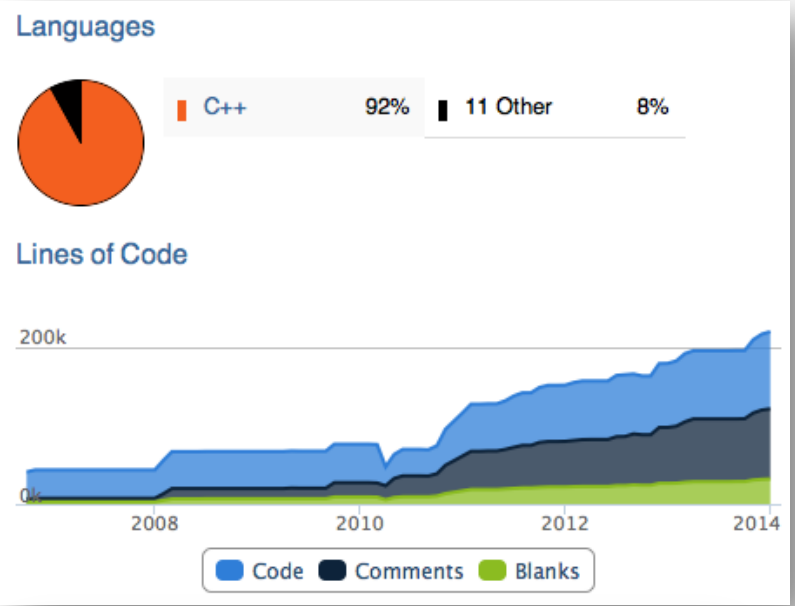
GammaLib world map 2013

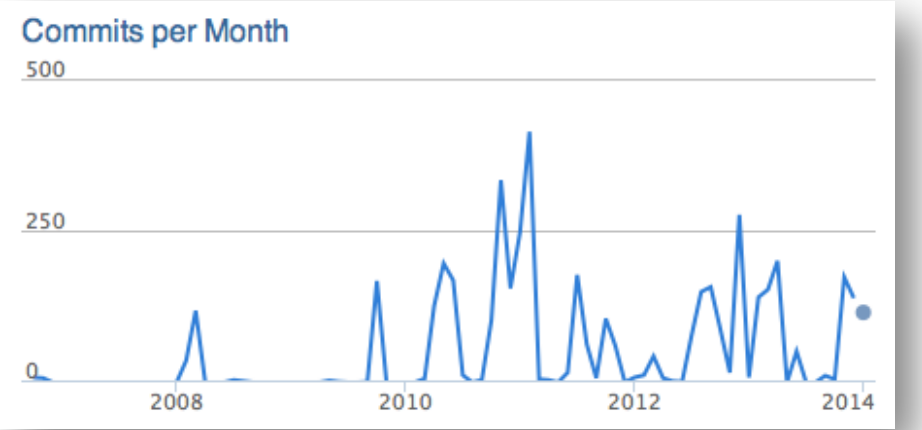
SourceForge downloads



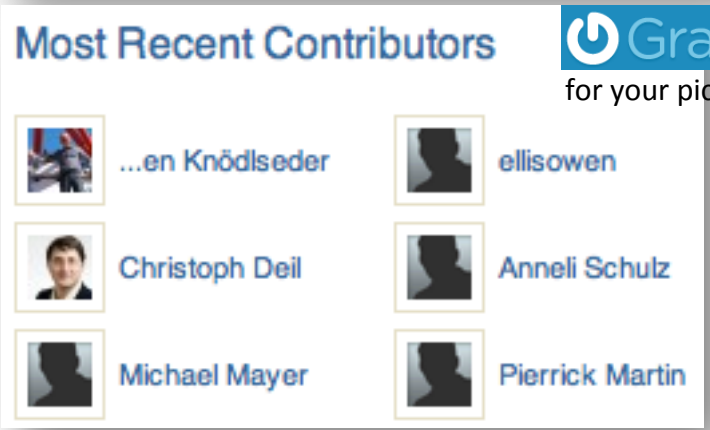
GammaLib statistics

https://www.ohloh.net/p/GammaLib



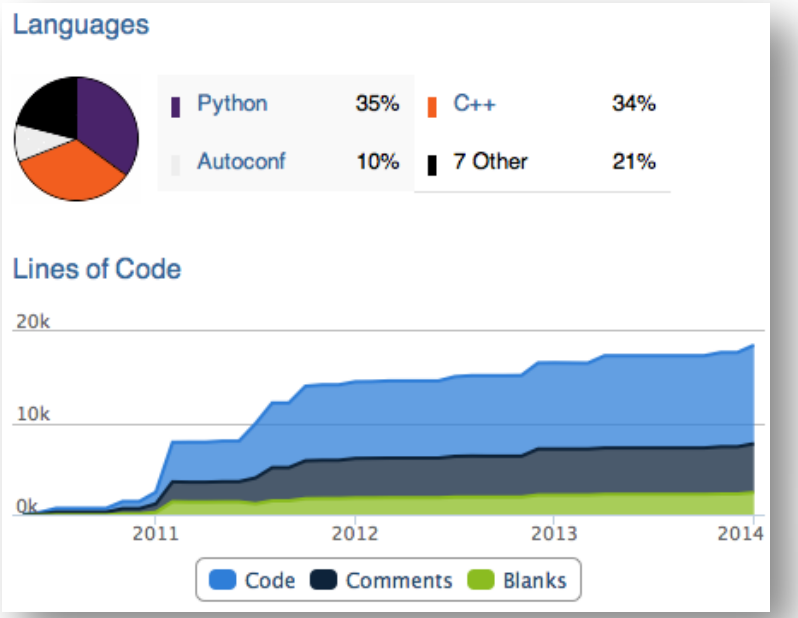


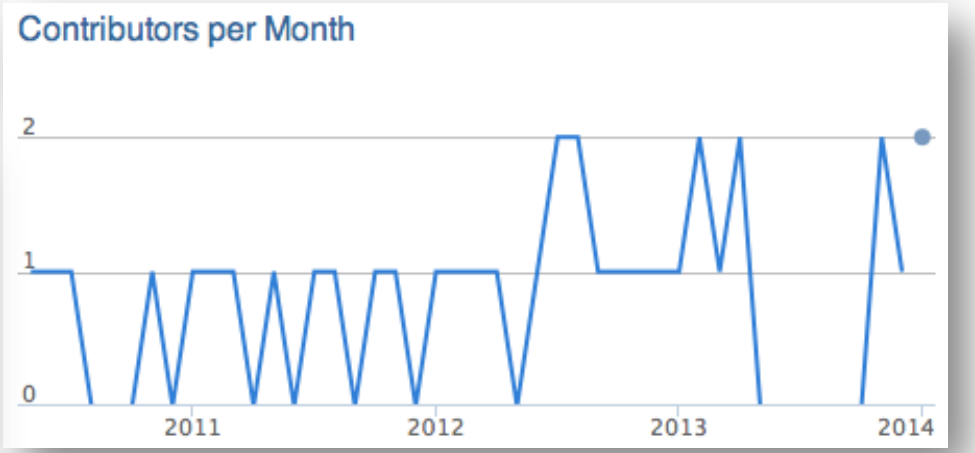
the community is growing …



ctools statistics

https://www.ohloh.net/p/ctools_





Current status

GammaLib

- Release 00-08-00
- Includes abstract observation handling, data modelling, model fitting, application support
- Provides FITS and XML interfaces
- Homogeneous class interfaces
- Support for
  - CTA (binned and unbinned)
  - Fermi/LAT (binned)
  - COMPTEL (binned)
  - Multi-wavelength

ctools

- Release 00-07-00
- Provides observation simulation, event selection, binning, model fitting, sky mapping
- Supports multi-instrument fitting

Next steps

GammaLib

- Freeze interfaces
- Reduce number of exception classes
- Optimize code for binned analysis
- Extend unit tests to full code coverage
- Implement performance tests
- Implement regression tests
- Complete user and developer documentation
- Provide binary packages
- … and extend the package https://cta-redmine.irap.omp.eu/rb/master_backlog/gammalib

ctools

- Extend unit tests
- Implement performance tests
- Implement regression tests
- Complete user and developer documentation
- Provide binary packages
- … and extend the package https://cta-redmine.irap.omp.eu/rb/master_backlog/ctools

5. Goals of this sprint

Goals of this sprint

- Implement energy resolution (#1036): **Christoph, Ellis**
- Implement general scheme to handle event classes (#1001): **Chia-Chun**
- Implement ctool to combine run-wise event lists and IRFs (#1037): **Chia-Chun**
- Implement ctool to create cube background model from off run list (#1038): **Christoph**
- Implement reflected regions background method (#1044): **Maria, Pierrick, Anneli**
- Implement handling of background information in instrument coordinate and derivation from CTA simulations (#1096): **Lucie, Rolf**
- Likelihood fits to Fermi, HESS and other MWL data using gammalib-external physical SED modeling packages:
- Implement SED fitting using external "model cubes" (N+1 dimensional data cube, containing SED templates for N parameters; gammalib determines best fitting values of these N parameters, see #599):
- Optimize binned CTA analysis:
- Implement some selected GammaLib issues (https://cta-redmine.irap.omp.eu/projects/gammalib/issues):
- Implement unbinned analysis for Fermi-LAT:
- Check / discuss results from ctools analyses against HESS software. (Discussed and agreed to not do this at the coding sprint):
  - create ctool to create background models from off runs: **Karl**
    This tool should take a off run list as input and built background models from the data. For this, the `GSkymap` functionality might be enhanced to allow smoothing, oversampling etc.

… and of course: having lots of fun!