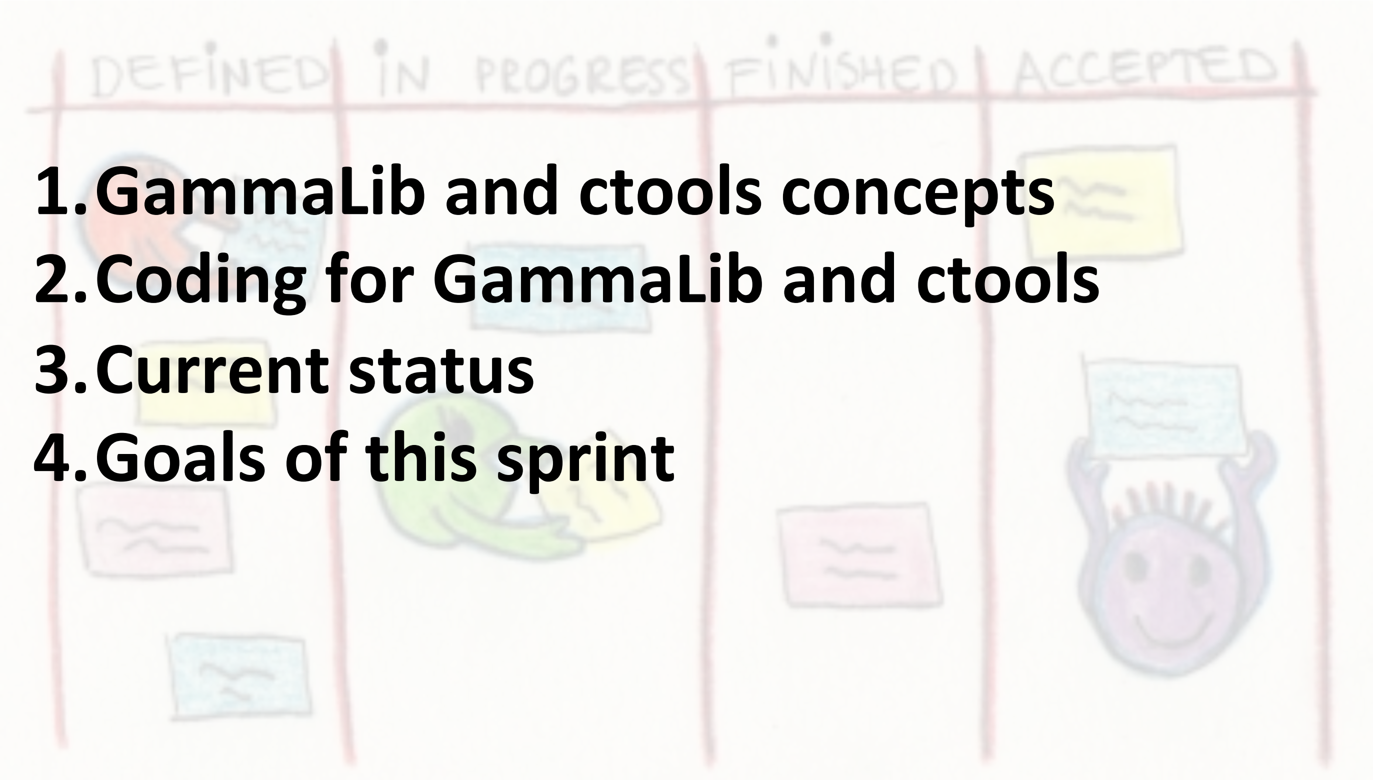
3nd Coding Sprint

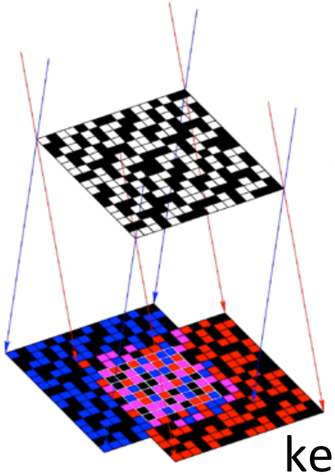


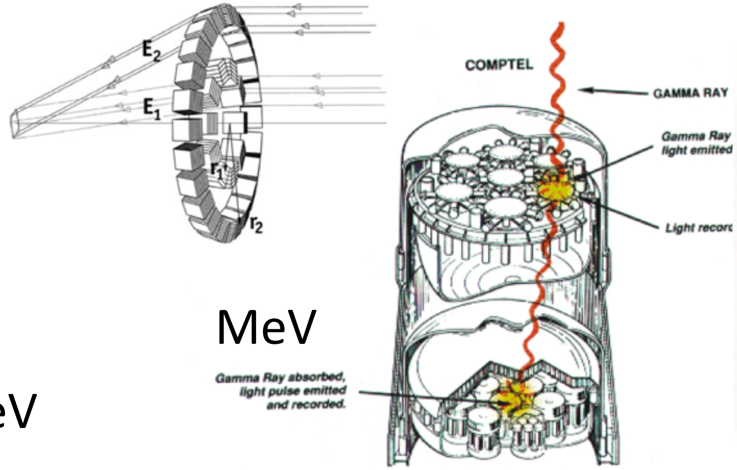
1. GammaLib and ctools concepts
2. Coding for GammaLib and ctools
3. Current status
4. Goals of this sprint

Jürgen Knödlseder (IRAP)

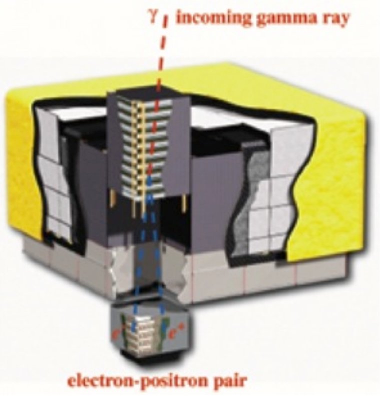
1. GammaLib and ctools concepts

Concept


keV



MeV



GeV

TeV

All gamma-ray telescopes measure individual photons as events. In principle it should be possible to **handle events from gamma-ray telescopes in an abstract and common software framework**.

Existing high-energy analysis frameworks share a number of **common features** (FITS files, likelihood fitting, modular design).

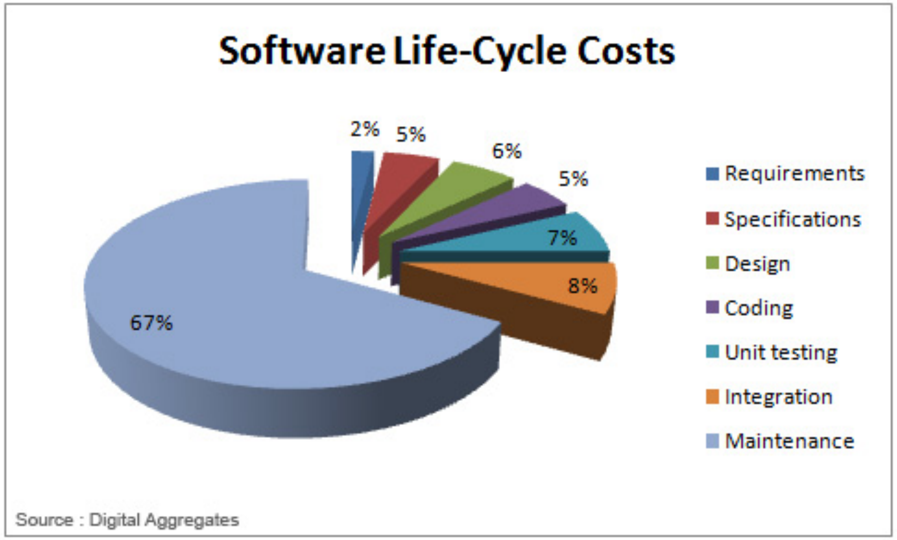
ctools cherenkov telescope array … is the client that uses the bricks provided by γLib

CTA specific *generic*

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

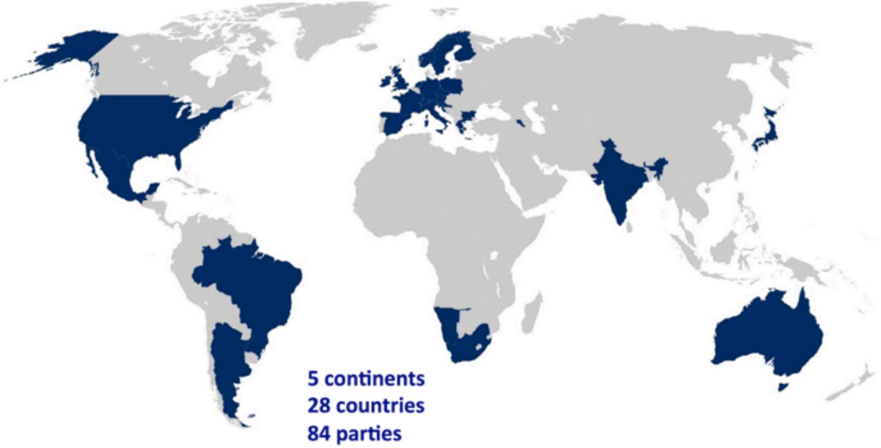
Design considerations

Minimise maintenance costs and maximise community involvement



Source : Digital Aggregates

CTA Consortium

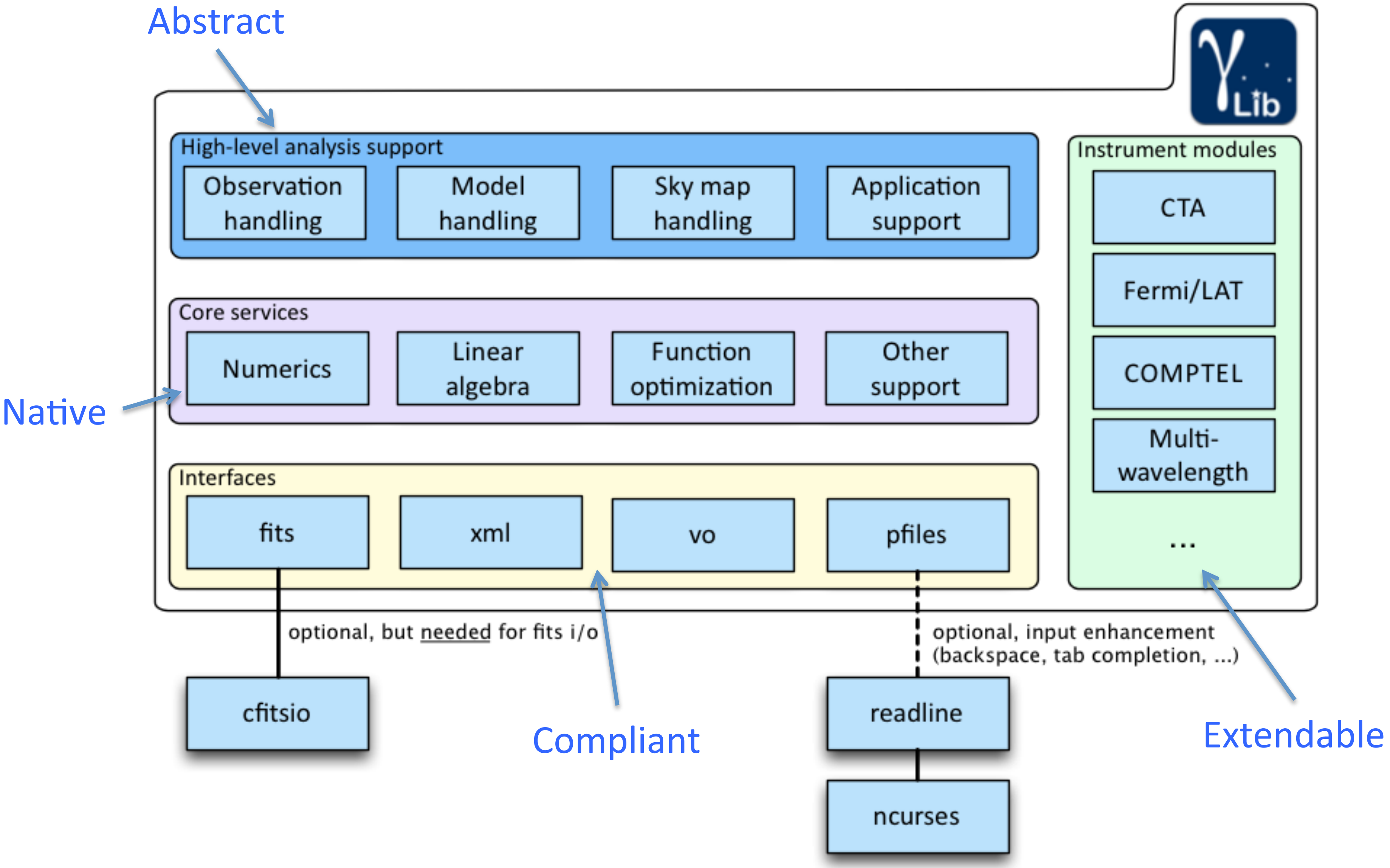


5 continents
28 countries
84 parties
173 institutes
1178 members (378 FTE)

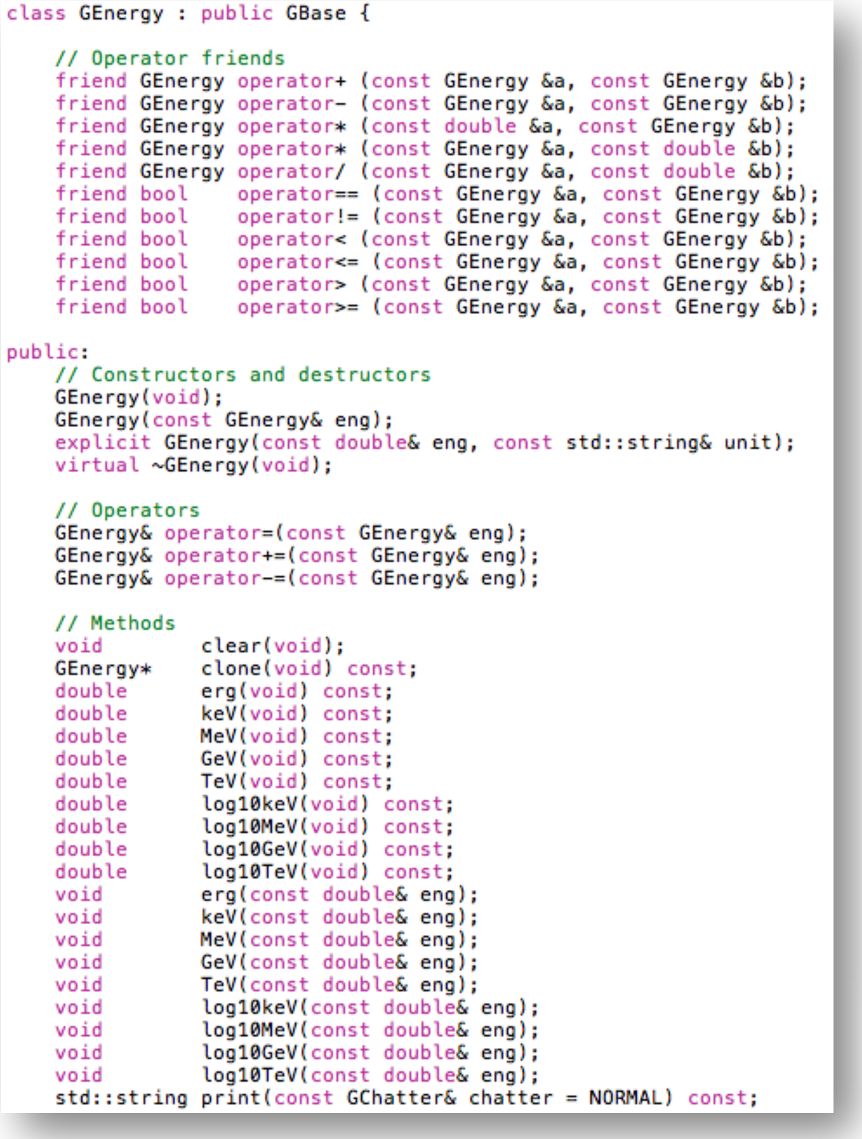
- Define and enforce coding rules (code quality)
- Avoid dependencies (full control over product)
- Support widely used platforms (Linux, Mac OS X, Solaris)
- Automatize unit testing, integration and deployment (continuous integration system & quality check)

- Open source development (end users develop the code)
- Follow an AGILE development model (implement what end users need)
- Follow analysis models used in the high-energy astronomy domain (Fermi, INTEGRAL, XMM, Chandra, etc.)

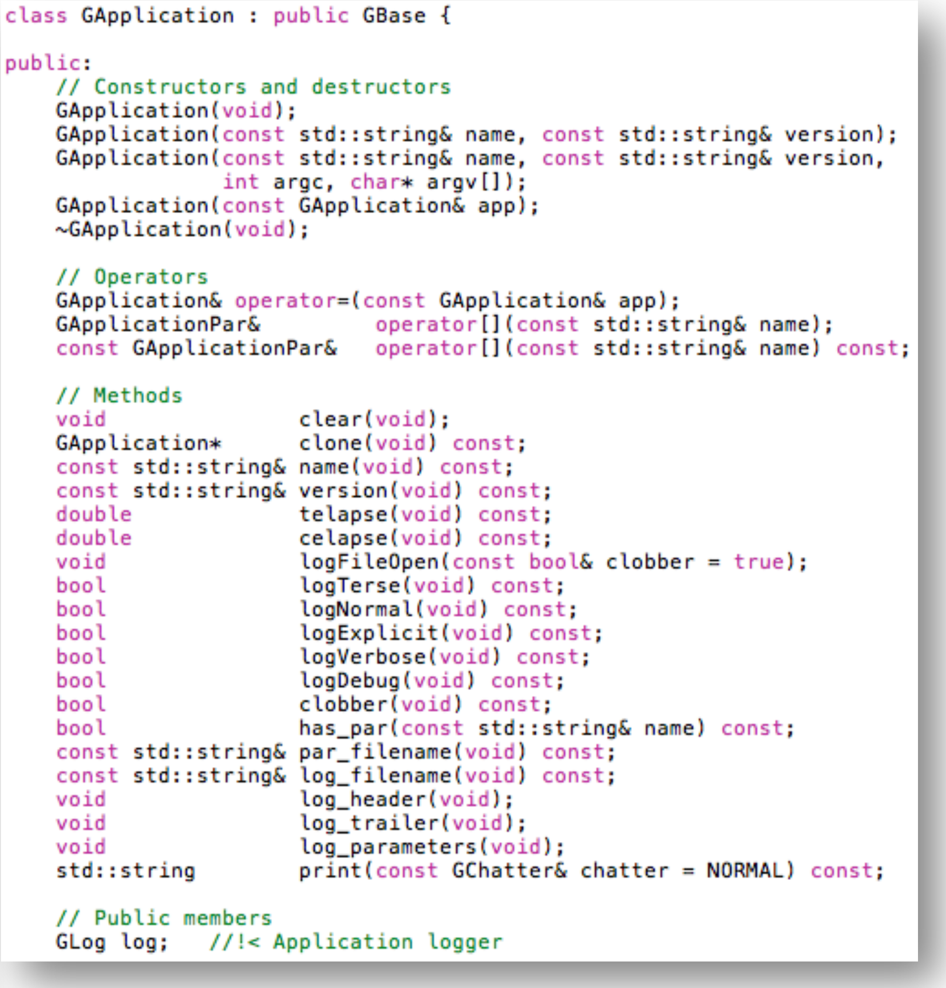
GammaLib overview



It’s all C++ classes

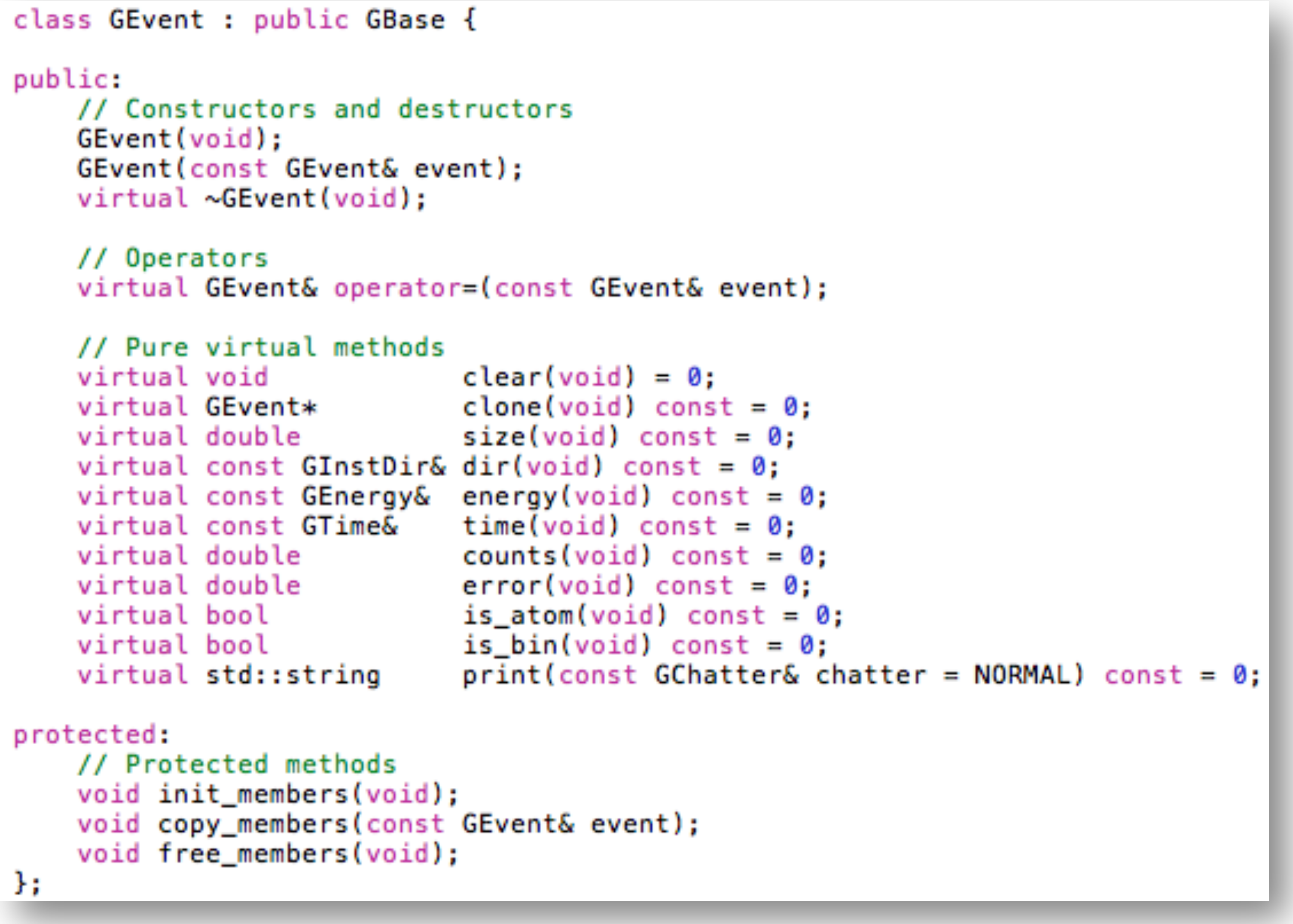


```
class GEnergy : public GBase {

    // 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;
```



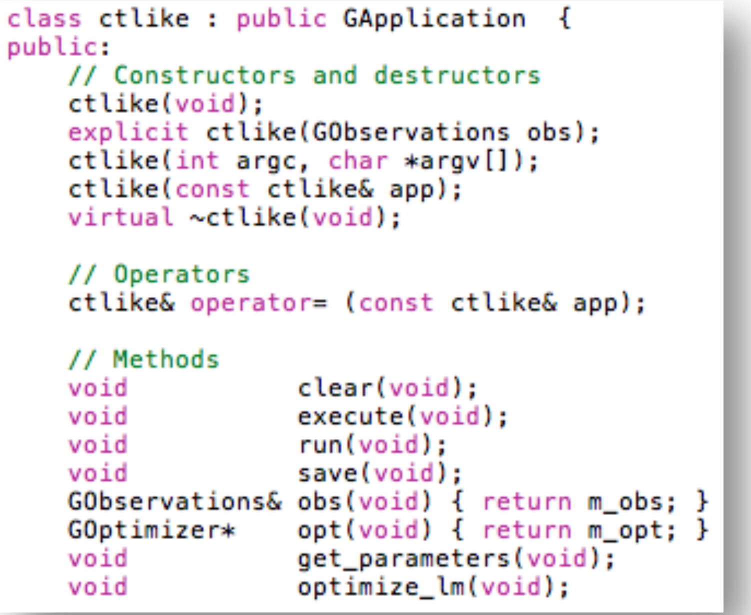
```
class GApplication : public GBase {

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



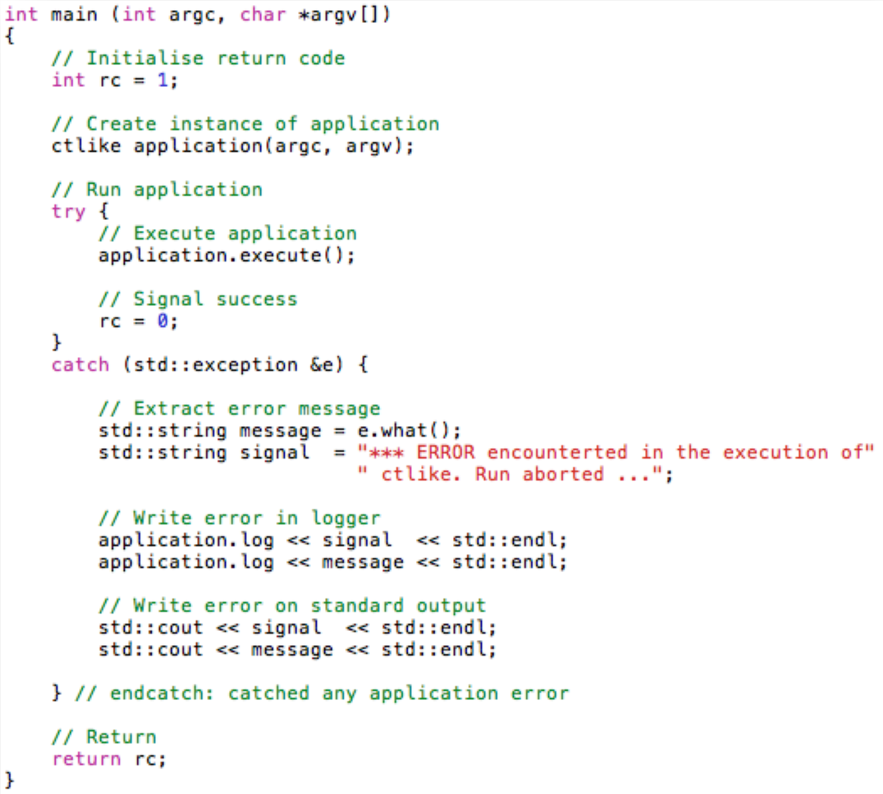
```
class GEvent : public GBase {

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



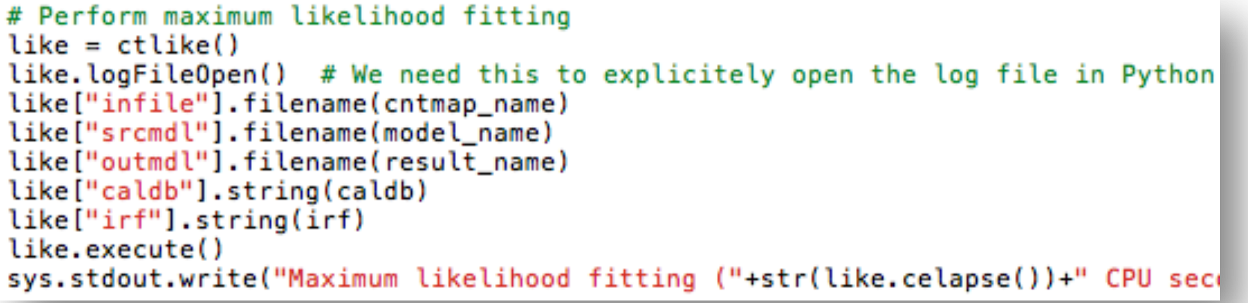
```
class ctlike : public GApplication  {
public:
    // Constructors and destructors
    ctlike(void);
    explicit ctlike(GObservations obs);
    ctlike(int argc, char *argv[]);
    ctlike(const ctlike& app);
    virtual ~ctlike(void);

    // 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;
}
```

… 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
```

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

Running a *ctool* executable

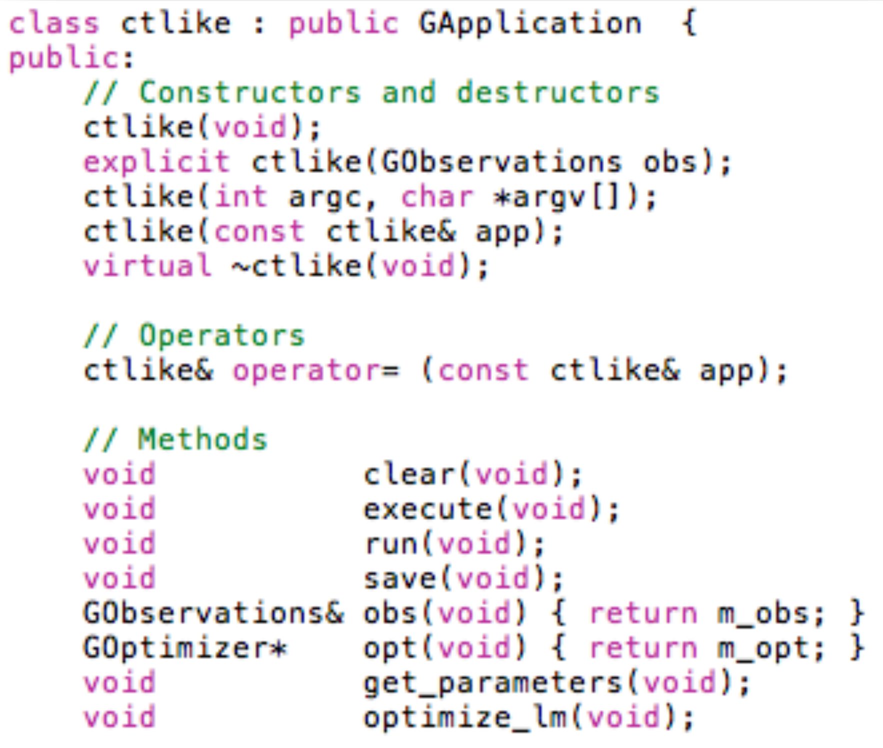
CTA event list simulator

```
$ ctobssim
Model [$CTOOLS/share/models/crab.xml]
Calibration database [aar]
Instrument response function [DESY20140105_50h]
RA of pointing (degrees) (0-360) [83.63]
Dec of pointing (degrees) (-90-90) [22.01]
Radius of FOV (degrees) (0-180) [5.0]
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

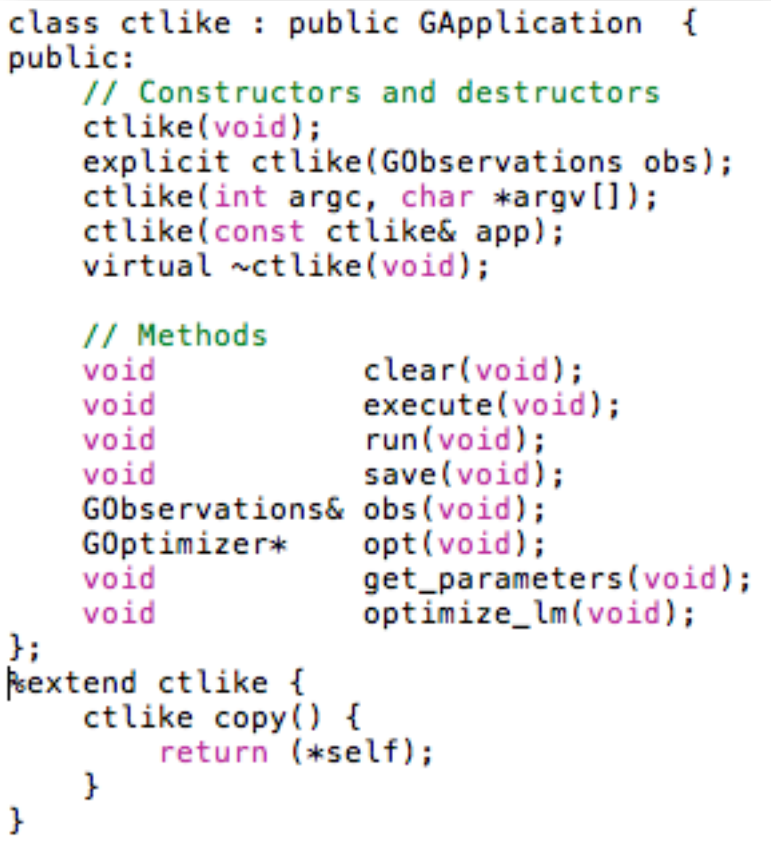
http://www.swig.org/

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);
```

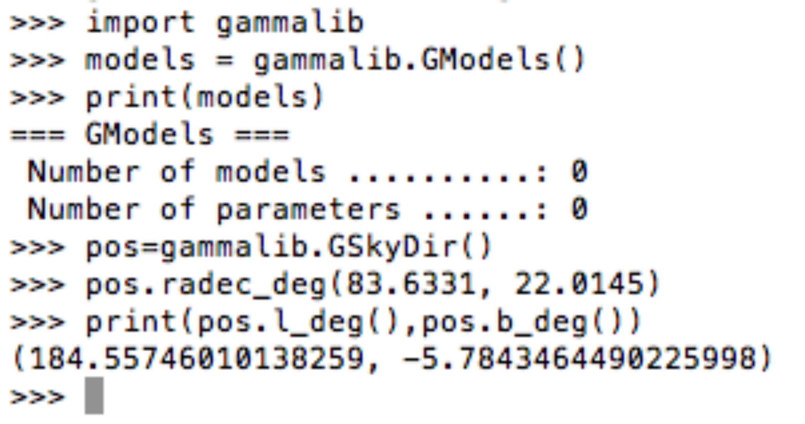
ctlike.i



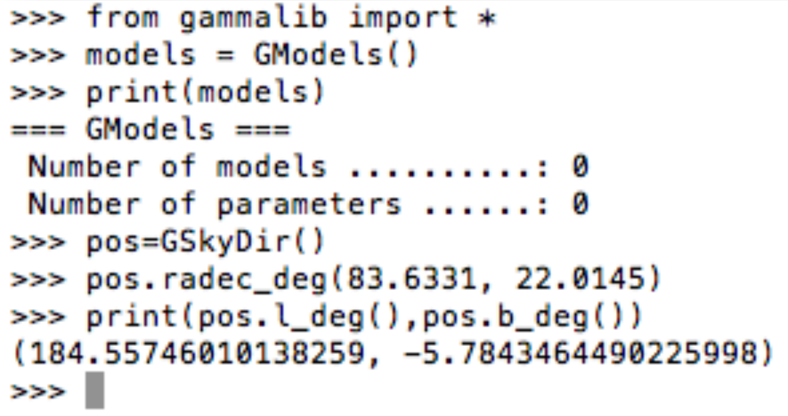
```
class ctlike : public GApplication  {
public:
    // Constructors and destructors
    ctlike(void);
    explicit ctlike(GObservations obs);
    ctlike(int argc, char *argv[]);
    ctlike(const ctlike& app);
    virtual ~ctlike(void);

    // 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



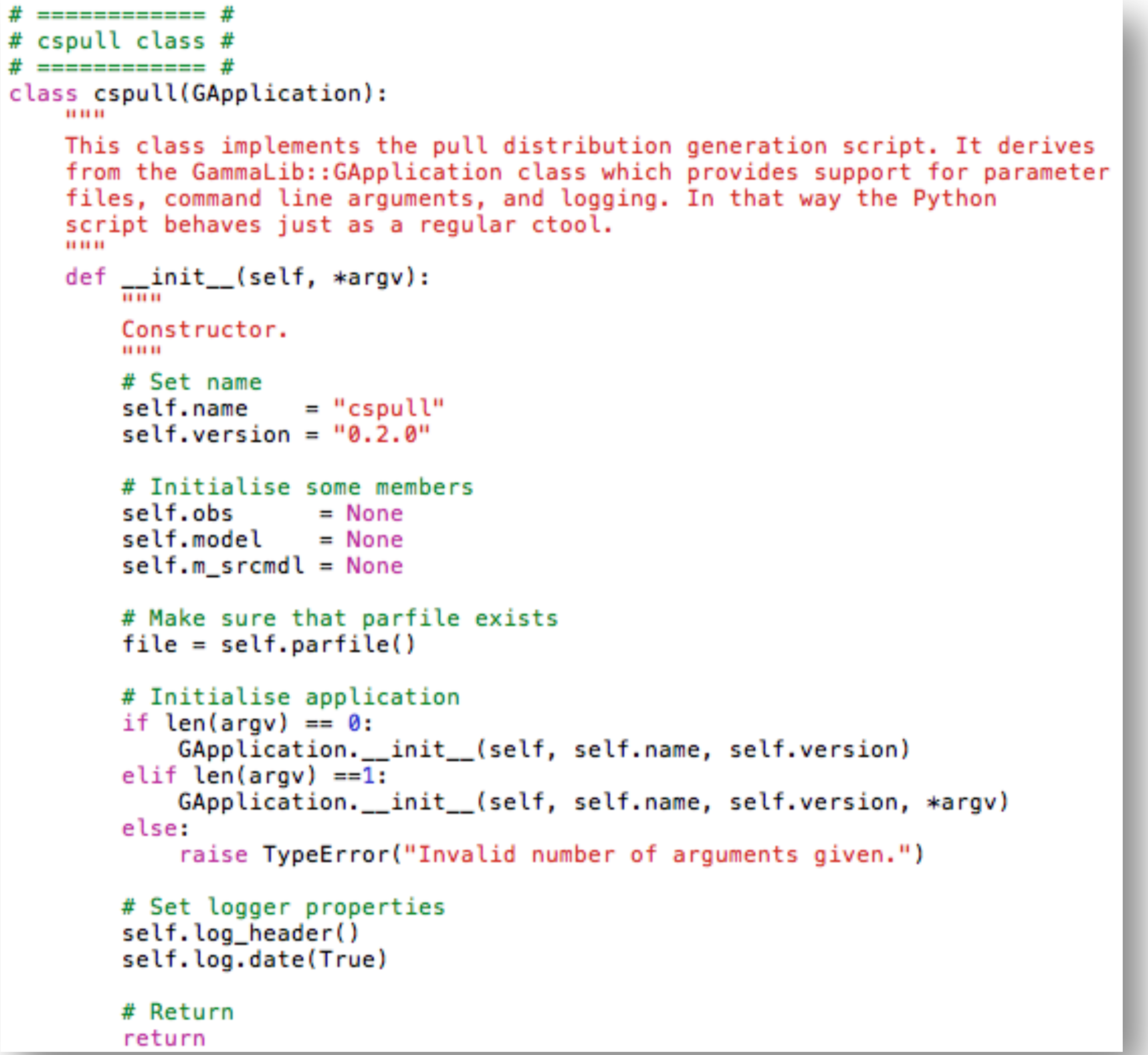
```
>>> import gammalib
>>> models = gammalib.GModels()
>>> print(models)
=== GModels ===
 Number of models ..........: 0
 Number of parameters ......: 0
>>> pos=gammalib.GSkyDir()
>>> pos.radec_deg(83.6331, 22.0145)
>>> print(pos.l_deg(),pos.b_deg())
(184.55746010138259, -5.7843464490225998)
>>> 
```



```
>>> 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)
>>> 
```

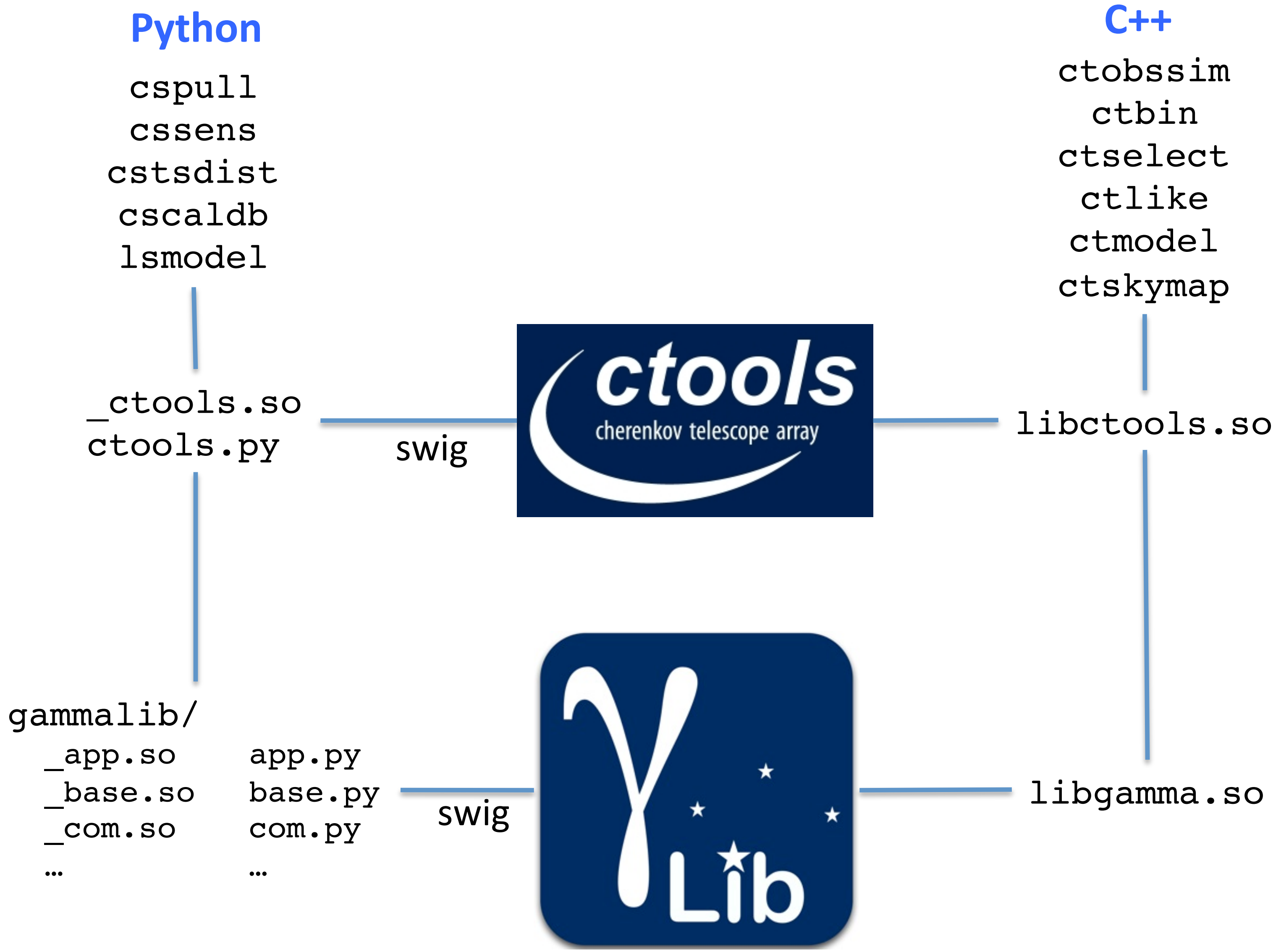
… same story for *ctools*

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



Python

cspull
cssens
cstsdist
cscaldb
lsmodel

_ctools.so
ctools.py

swig

gammalib/
_app.so
_base.so
_com.so
…

app.py
base.py
com.py
…

swig

C++

ctobssim
ctbin
ctselect
ctlike
ctmodel
ctskymap

libctools.so

libgamma.so

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.

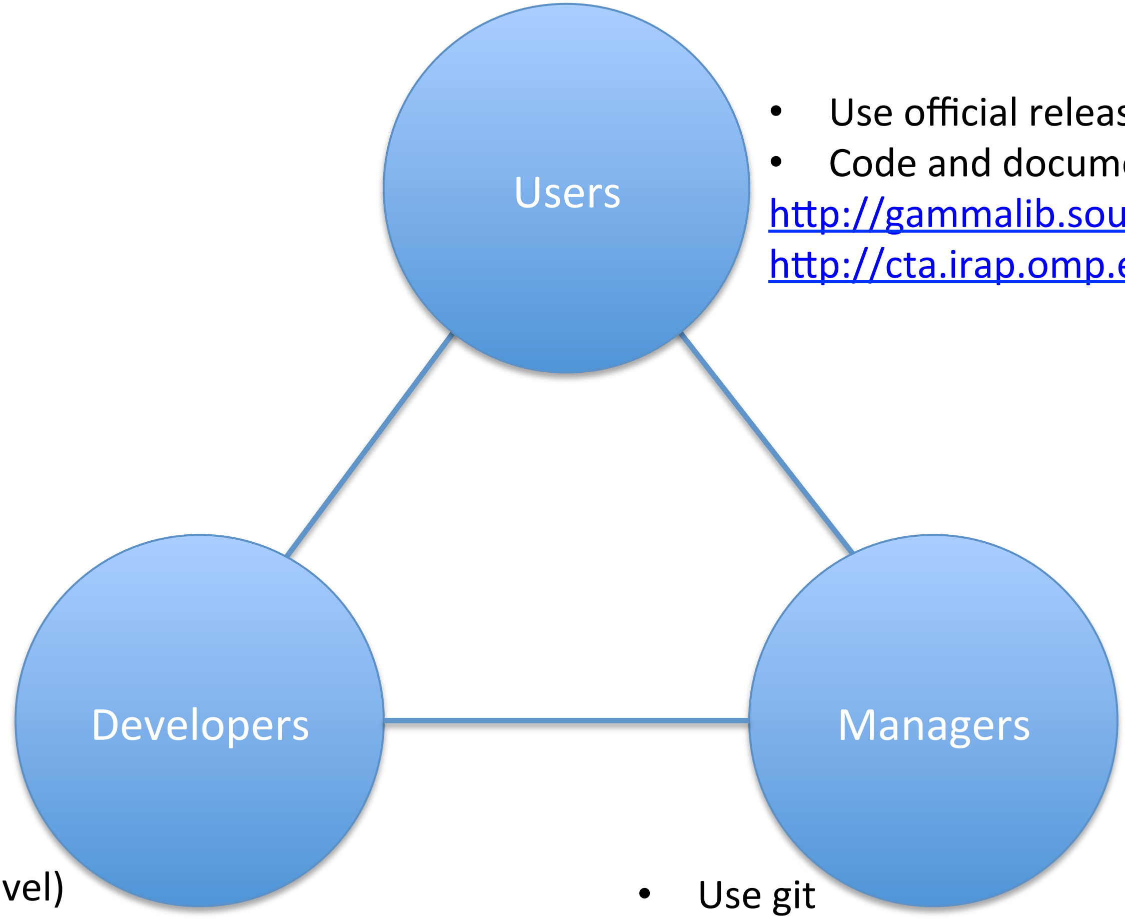
… I want to test a new idea (e.g. create a ring background generator)?
Create a new cscript that uses the gammalib Python module.

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.

2. Coding for GammaLib and ctools

Communities

@gammalib



- Use official releases
- Code and documentation at

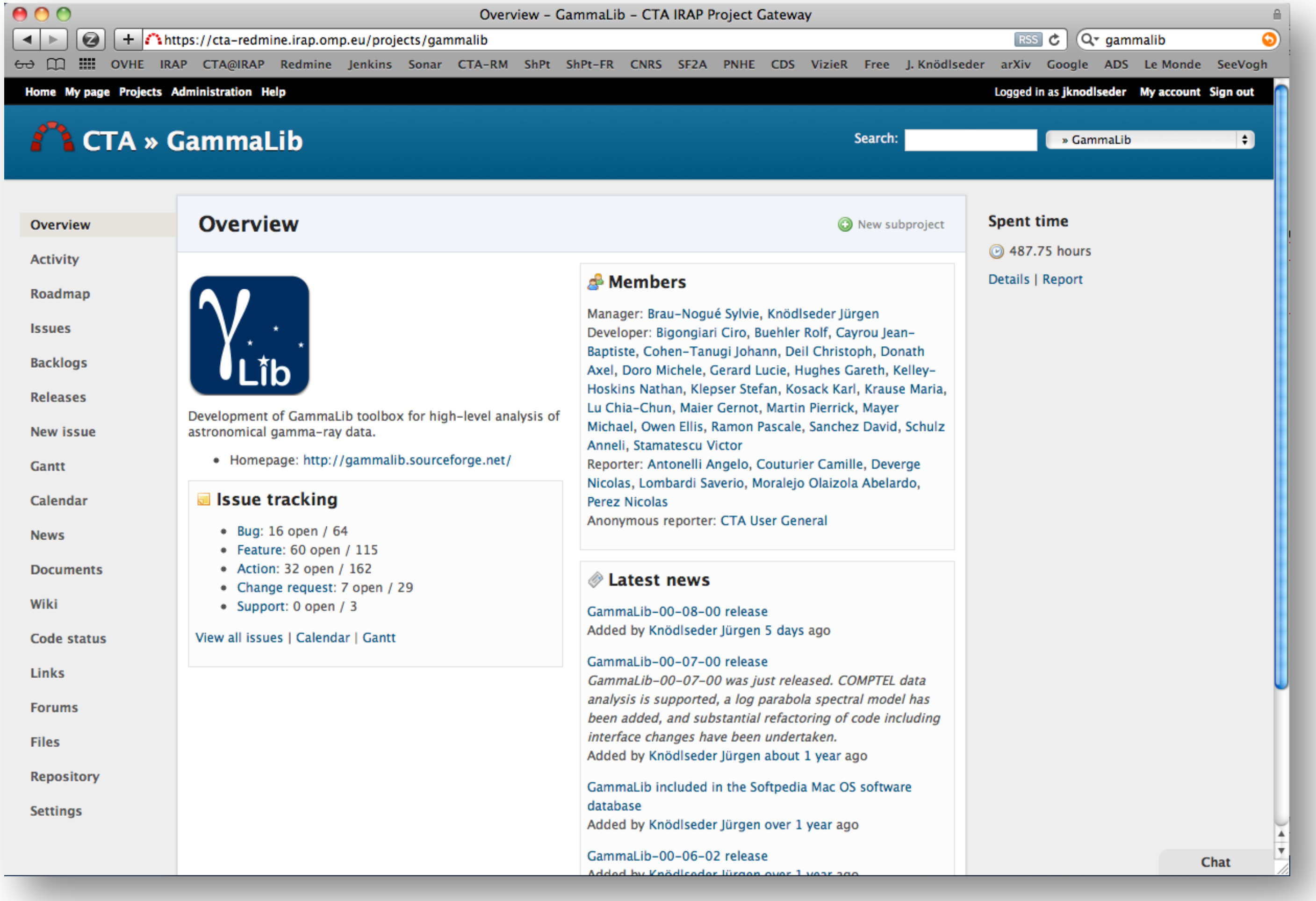
http://gammalib.sourceforge.net/
http://cta.irap.omp.eu/ctools/index.html

- 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
New subproject
Chat

Overview

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
Added by Knödlseder Jürgen over 1 year ago

Spent time

487.75 hours

Details | Report

If you want a new feature, find a bug, or request a change: use it!
Don't worry too much whether you file things under ctools or GammaLib (will clean up if needed)

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 <== always start from here
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

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).

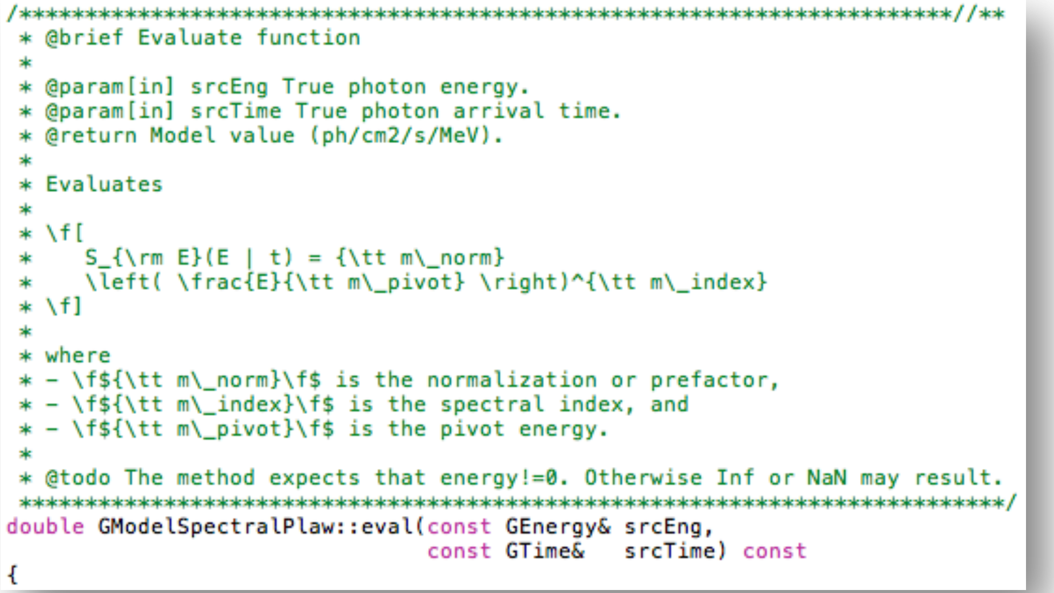
Why can't I push to master, release, devel or integration?
These branches are protected from pushing. Only the integration manager is allowed to push to them. See it from the good side: this puts a lower work burden on your side, and prevents you from destroying the repository.

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/



```
/************************************************************************//**
 * @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
{
```

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/

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*

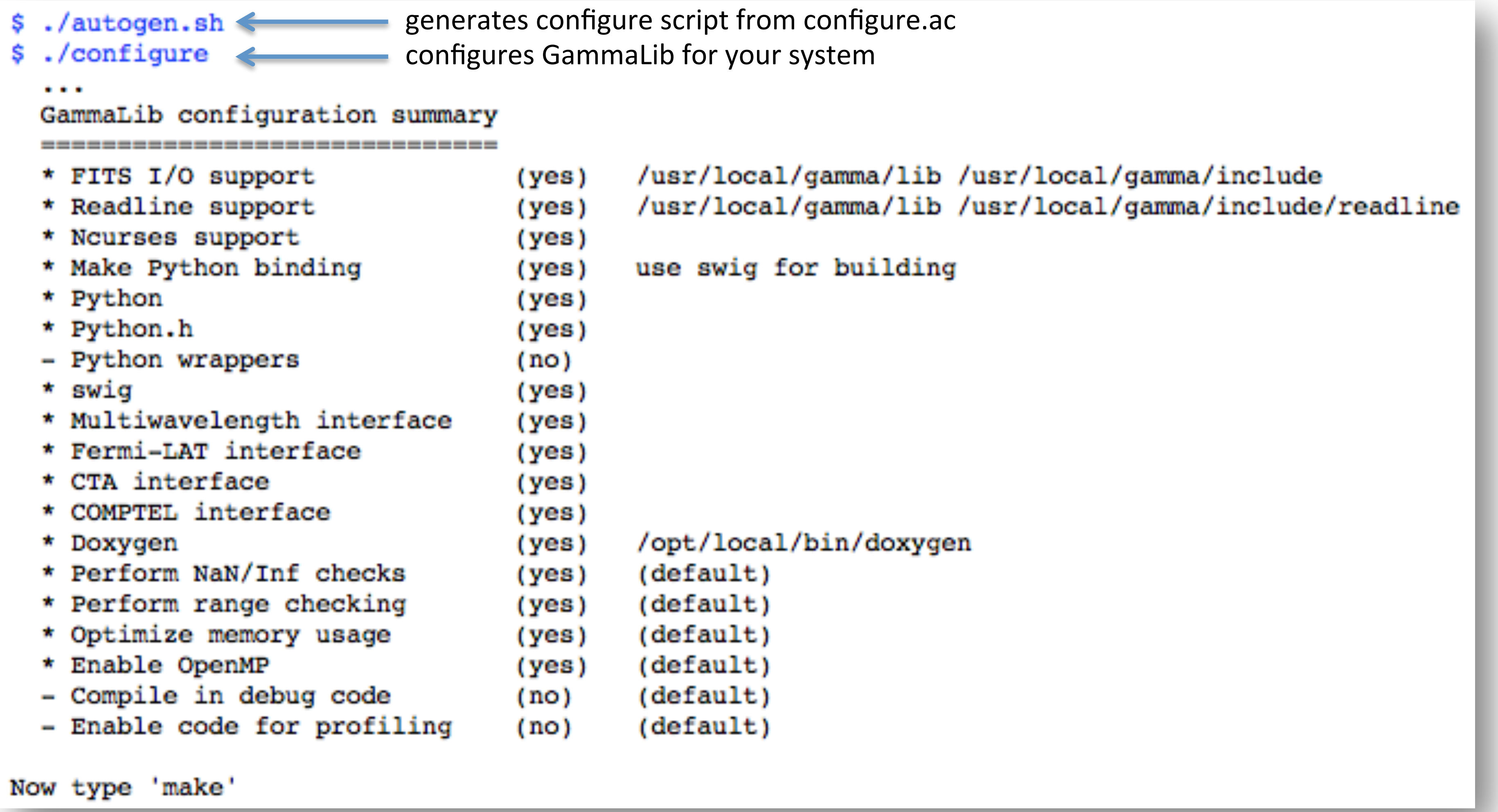
https://cta-git.irap.omp.eu/ctools

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



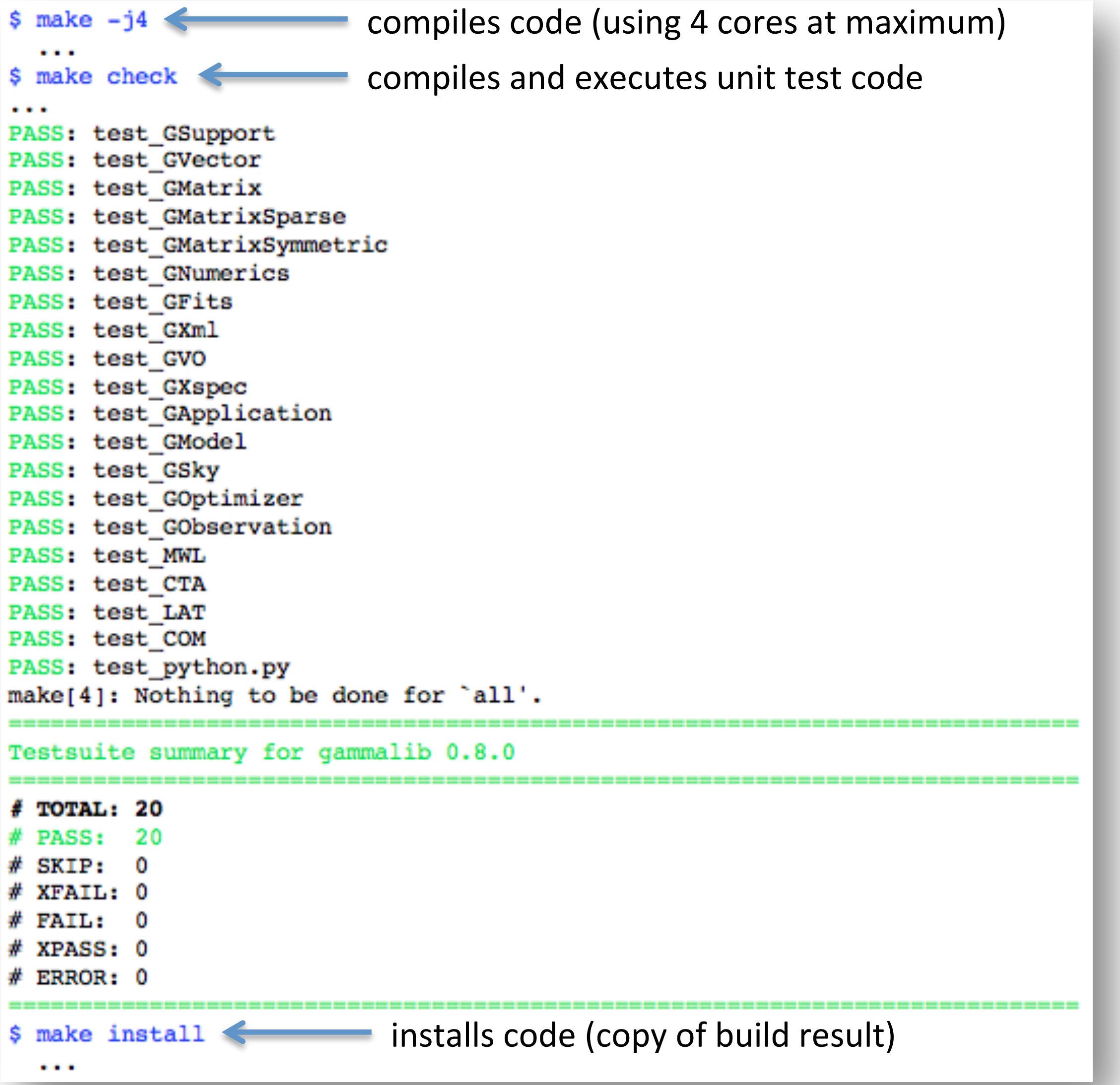
```
$ ./autogen.sh
$ ./configure
  ...
  GammaLib configuration summary
  ==============================
  * FITS I/O support              (yes)   /usr/local/gamma/lib /usr/local/gamma/include
  * Readline support              (yes)   /usr/local/gamma/lib /usr/local/gamma/include/readline
  * Ncurses 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).

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

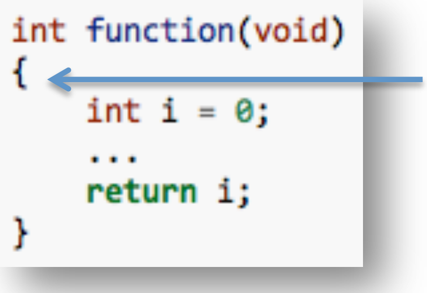
(apply to GammaLib and ctools; **will be enforced**)

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

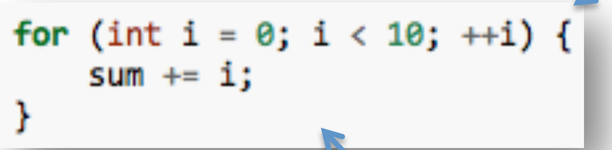


```
int function(void)
{
    int i = 0;
    ...
    return i;
}
```

Curly opening bracket at new line

Block format

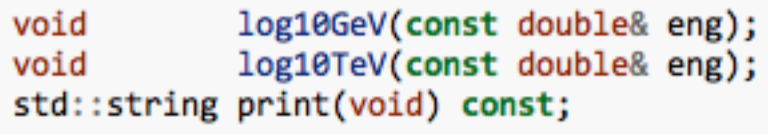
Curly opening bracket at end



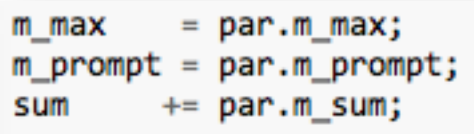
```
for (int i = 0; i < 10; ++i) {
    sum += i;
}
```

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

Code alignment

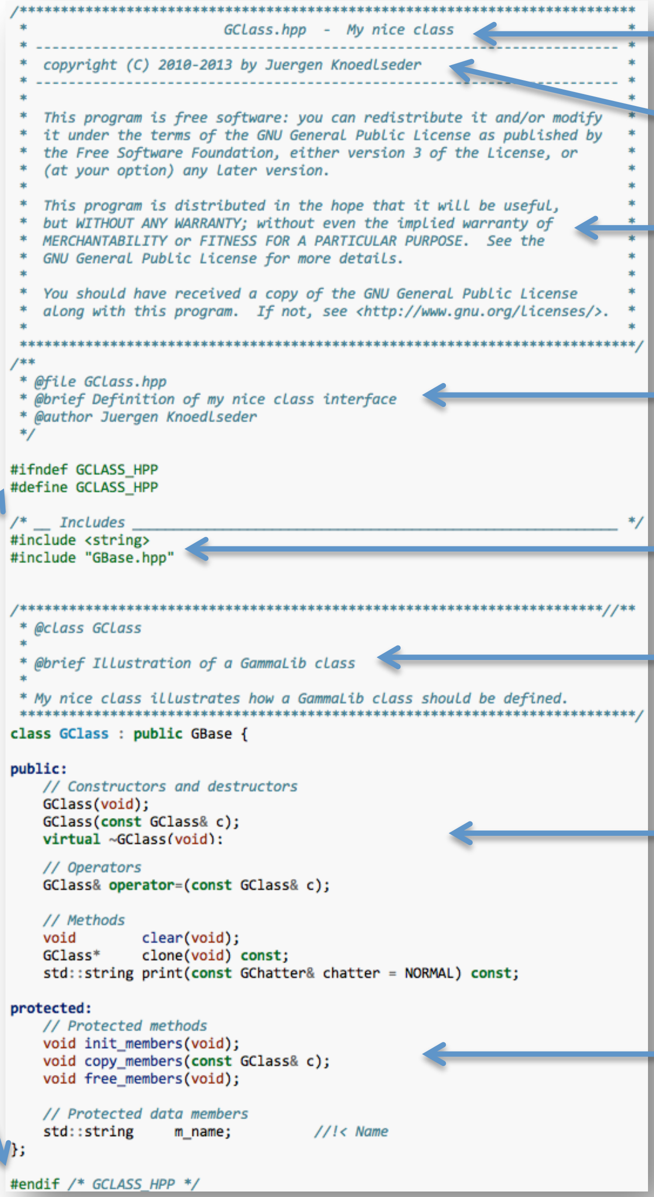


```
void        log10GeV(const double& eng);
void        log10TeV(const double& eng);
std::string print(void) const;
```



```
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 “ “)

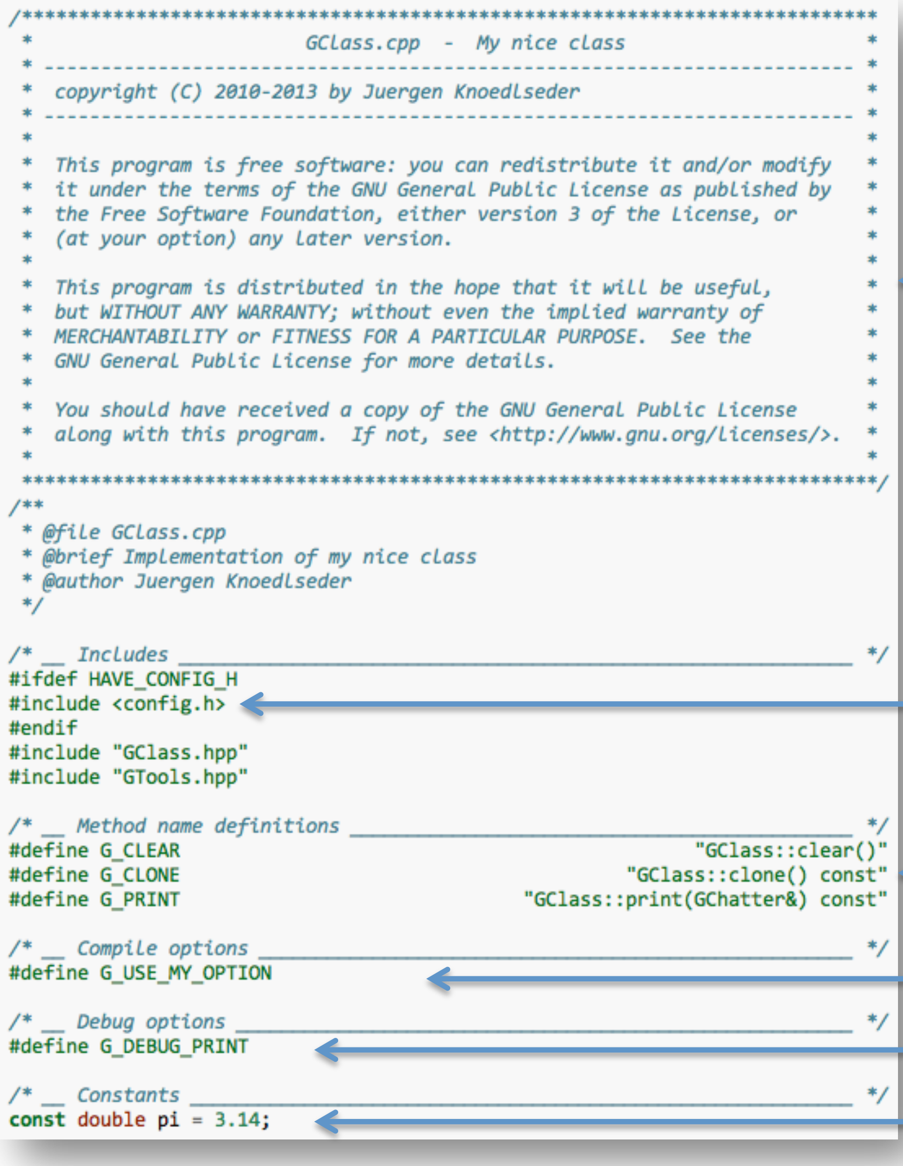
Class description (Doxygen syntax)

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.

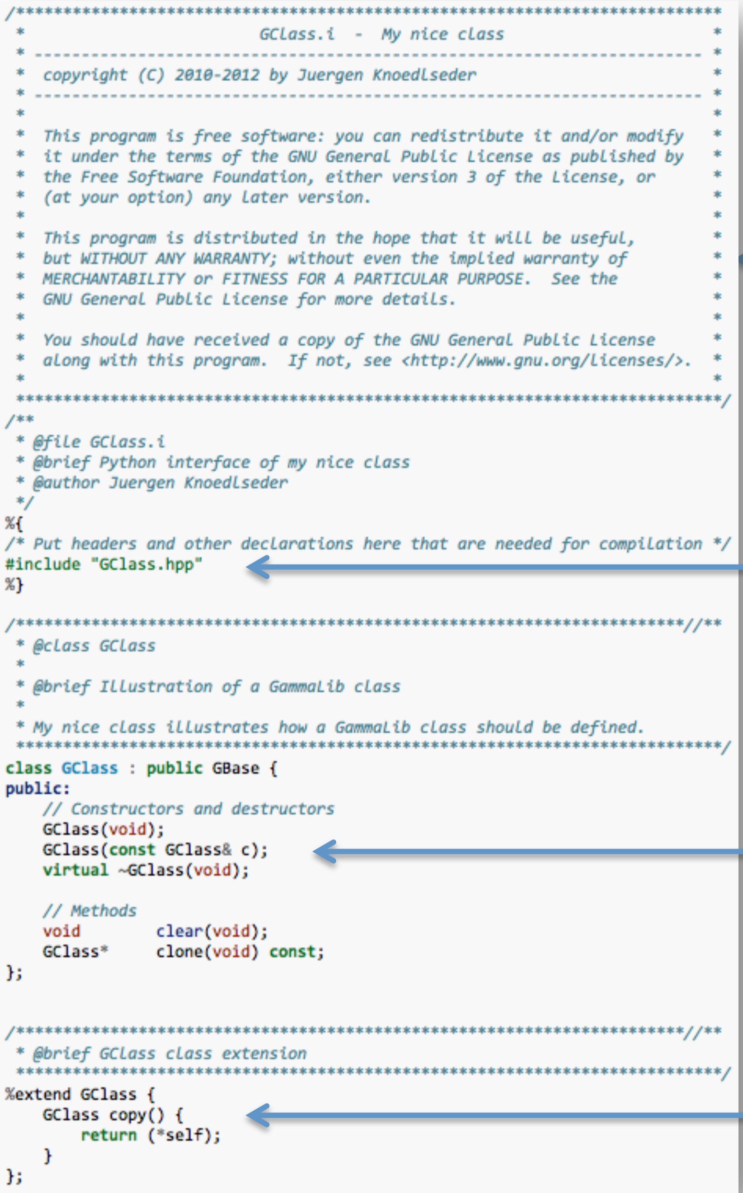
Method names used in exceptions

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)

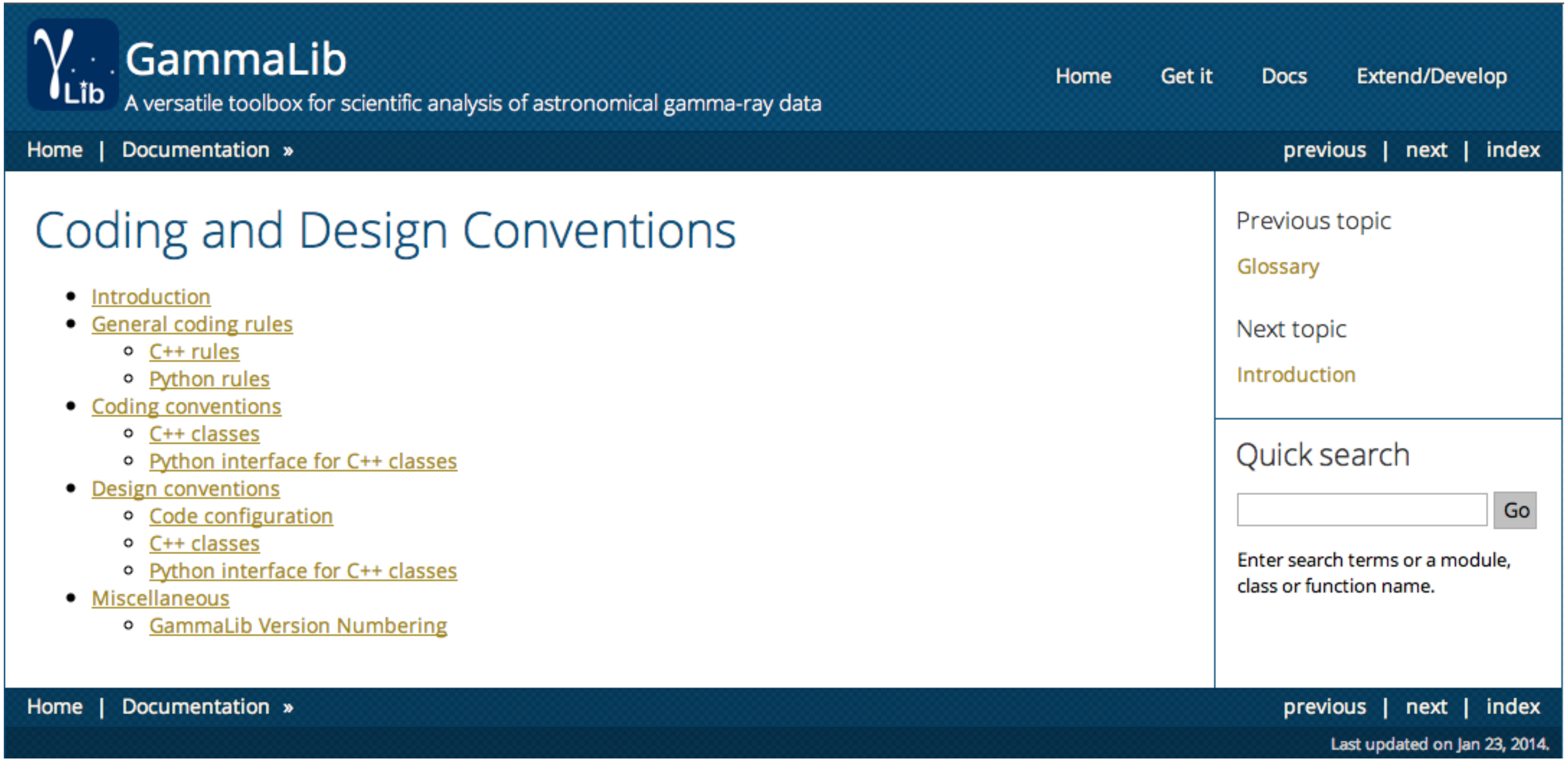
Basically a copy of the public class definition from
the .hpp file
without:

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

Extensions to the class only available in Python

More reading

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



GammaLib
A versatile toolbox for scientific analysis of astronomical gamma-ray data
Home Get it Docs Extend/Develop
Home | Documentation »
previous | next | index

Coding and Design Conventions

- Introduction
- General coding rules
  - C++ rules
  - Python rules
- Coding conventions
  - C++ classes
  - Python interface for C++ classes
- Design conventions
  - Code configuration
  - C++ classes
  - Python interface for C++ classes
- Miscellaneous
  - GammaLib Version Numbering

Previous topic

Glossary

Next topic

Introduction

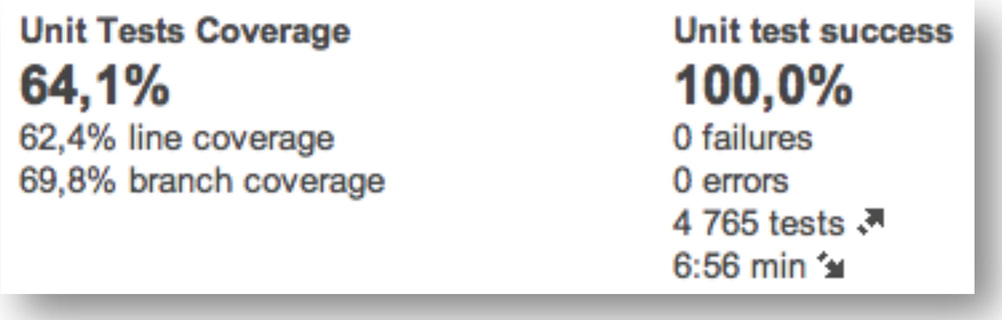
Quick search

Go

Enter search terms or a module, class or function name.

Last updated on Jan 23, 2014.

Unit testing

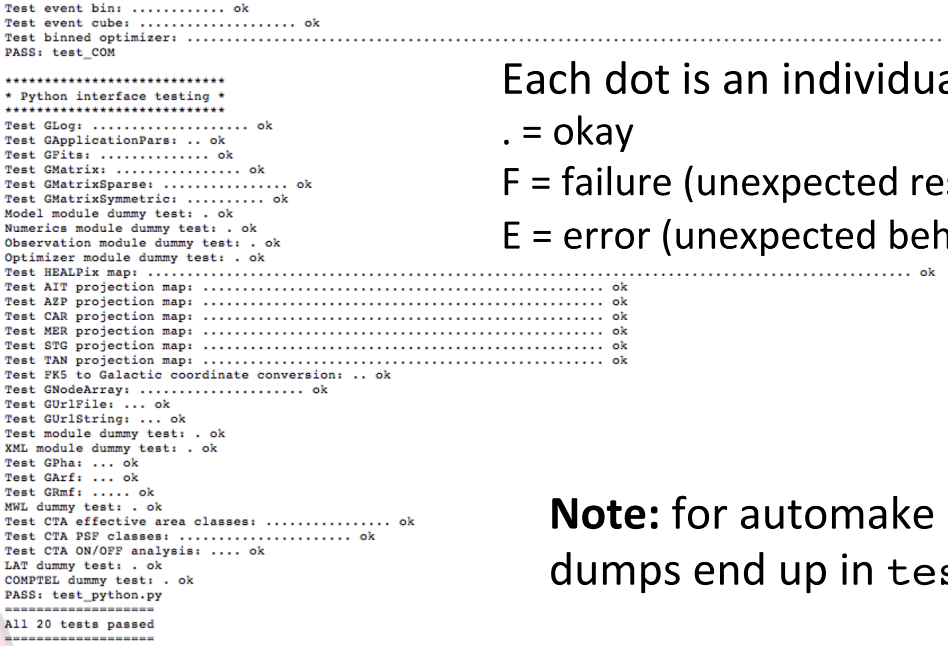


Unit Tests Coverage
64,1%
62,4% line coverage
69,8% branch coverage

Unit test success
100,0%
0 failures
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`



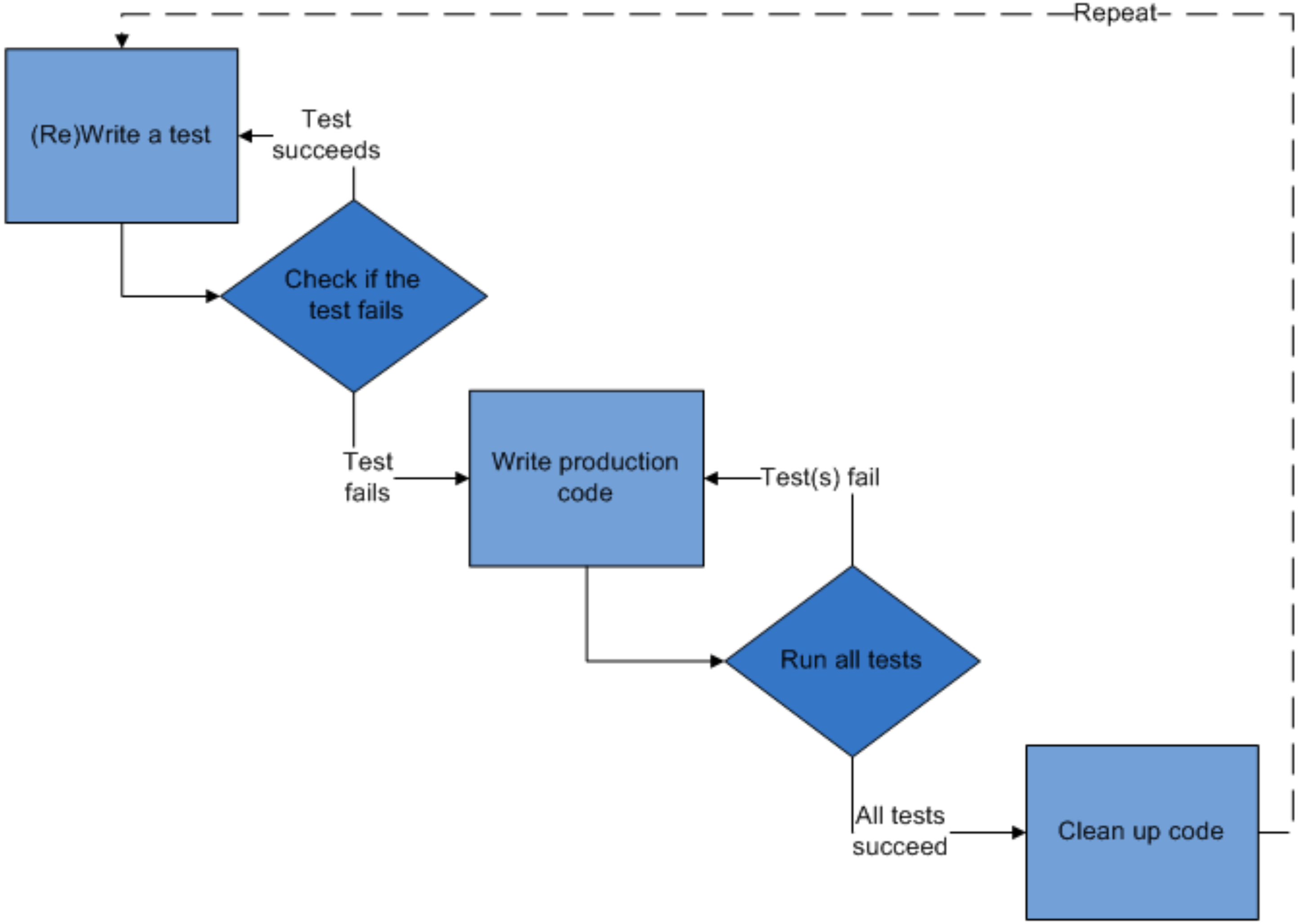
```
Test event bin: ............ ok
Test event cube: .................... ok
Test binned optimizer: .................................................................................................
PASS: test_COM

****************************
* 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

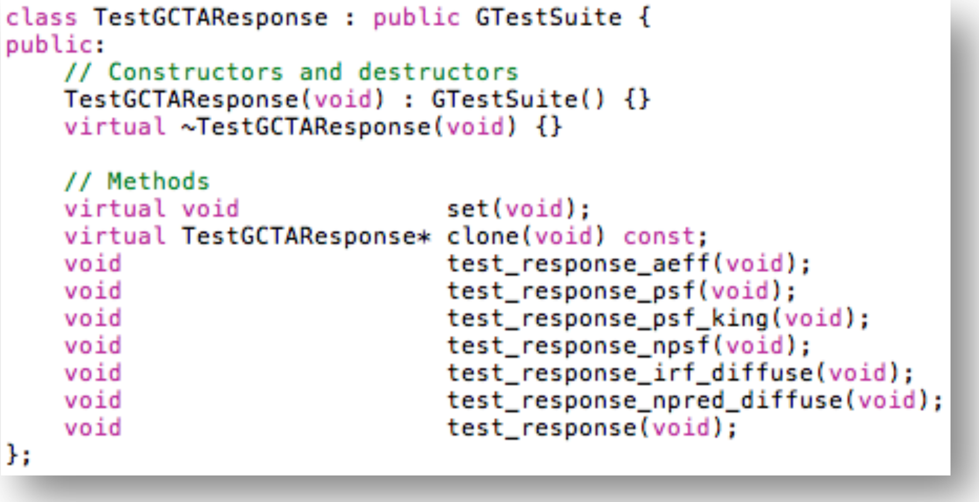
You should give it a try …



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

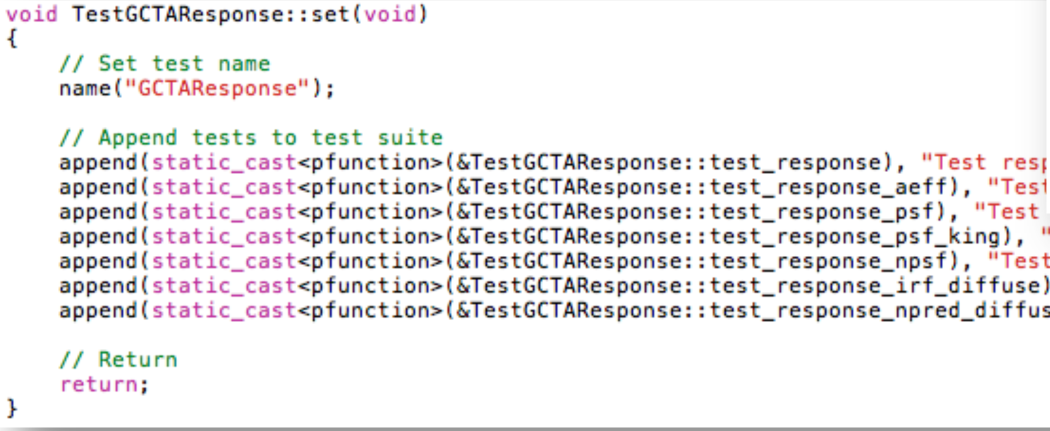
(see `inst/test/test_CTA.hpp` and `inst/test/test_CTA.cpp`)

Create a class that derived from GTestSuite



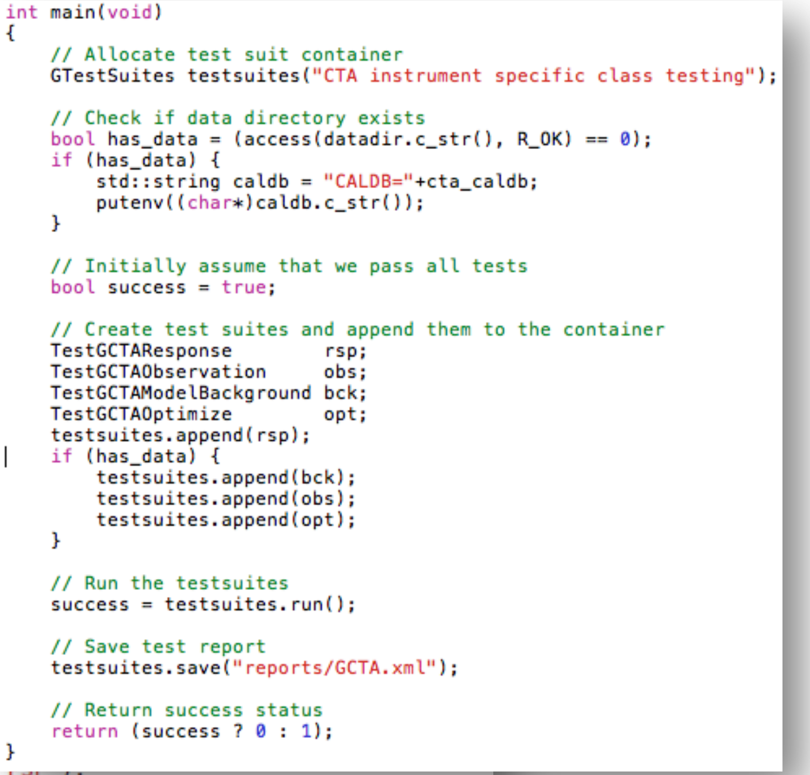
```
class TestGCTAResponse : public GTestSuite {
public:
    // Constructors and destructors
    TestGCTAResponse(void) : GTestSuite() {}
    virtual ~TestGCTAResponse(void) {}

    // 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 PSF");
    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;
}
```

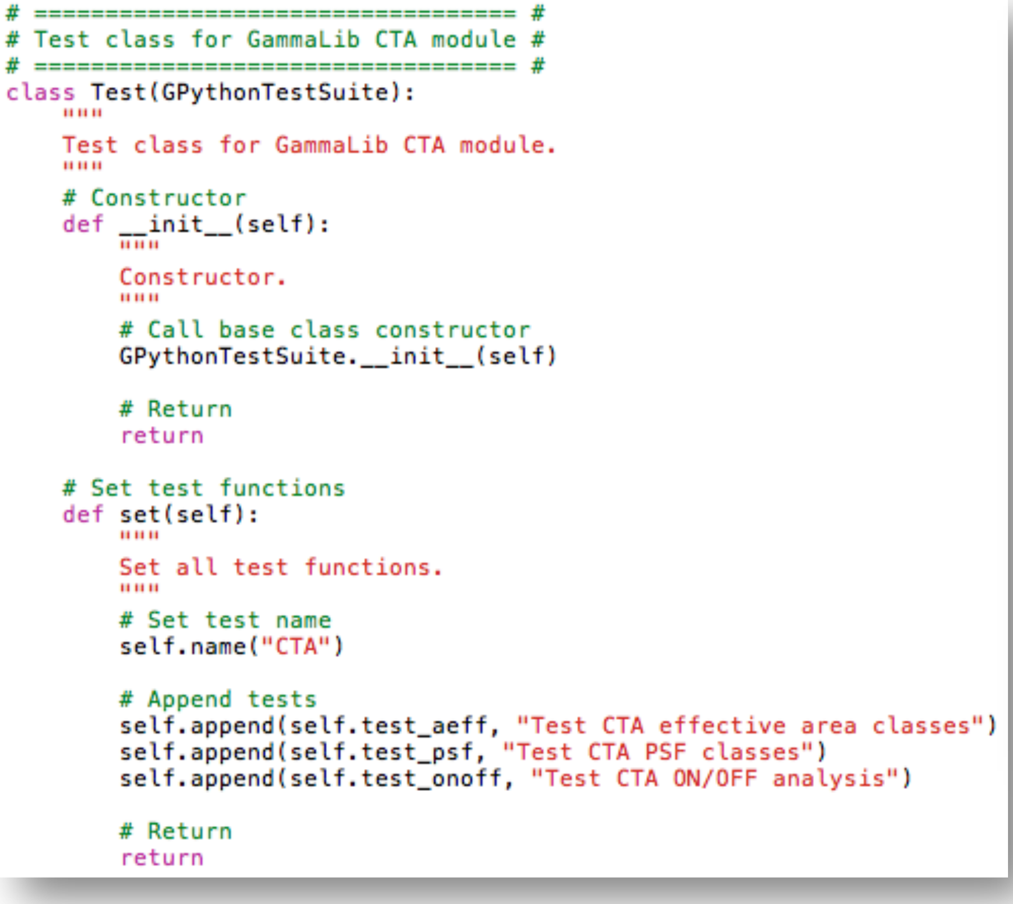
Append GTestSuite to container, run tests and save results



```
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);
}
```

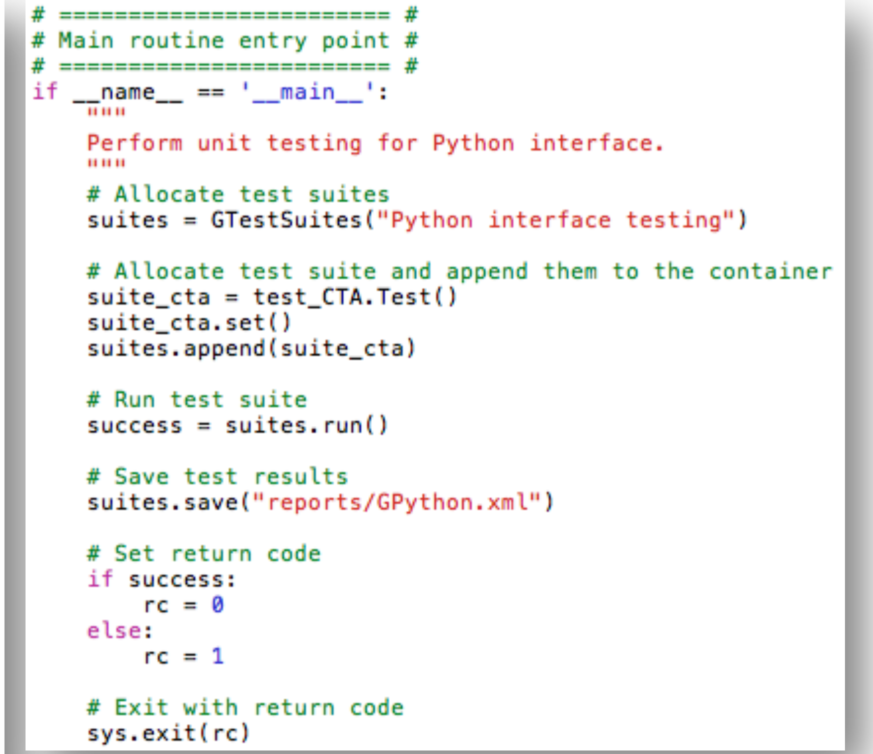
And in Python?

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



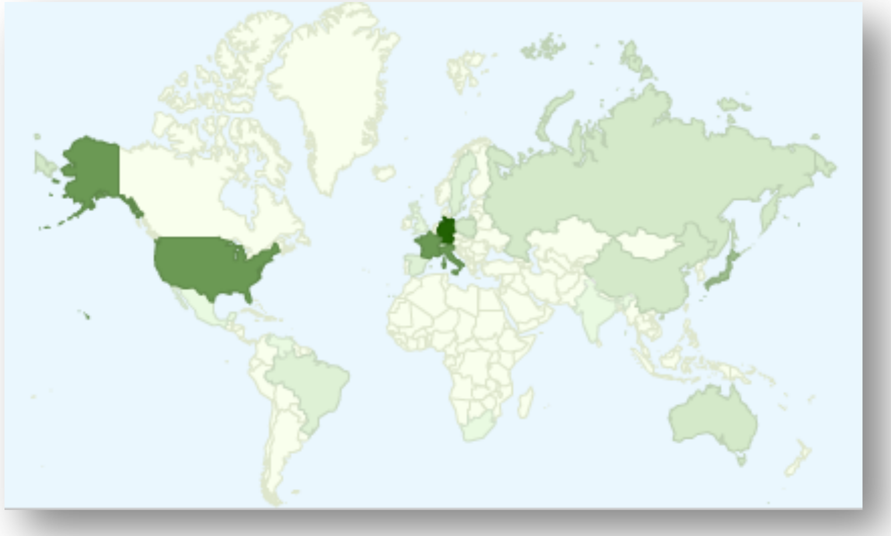
```
# ======================== #
# Main routine entry point #
# ======================== #
if __name__ == '__main__':
    """
    Perform unit testing for Python interface.
    """
    # Allocate test suites
    suites = GTestSuites("Python interface testing")

    # 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)
```

3. Current status

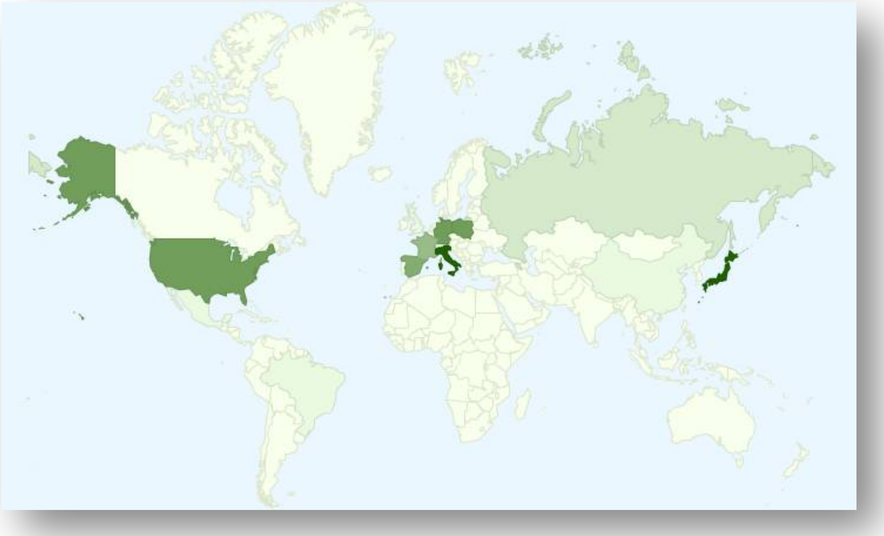
GammaLib world map

SourceForge downloads

2013

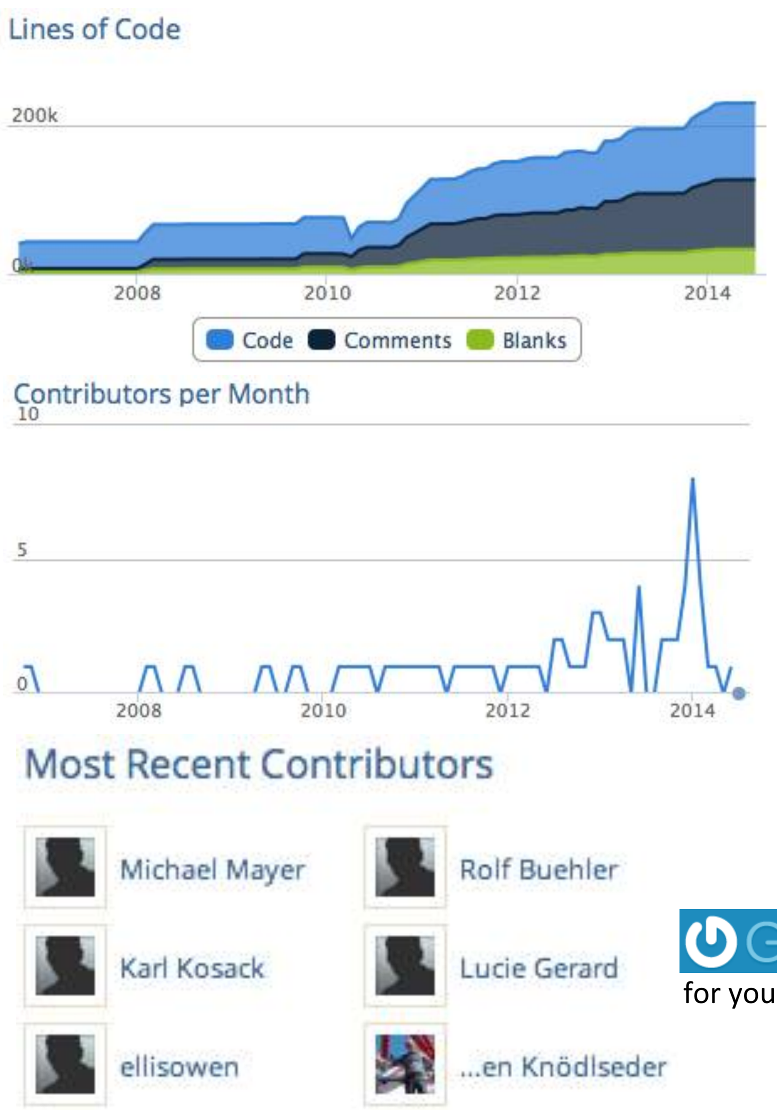


2014



GammaLib & ctools statistics

https://www.ohloh.net/



Lines of Code

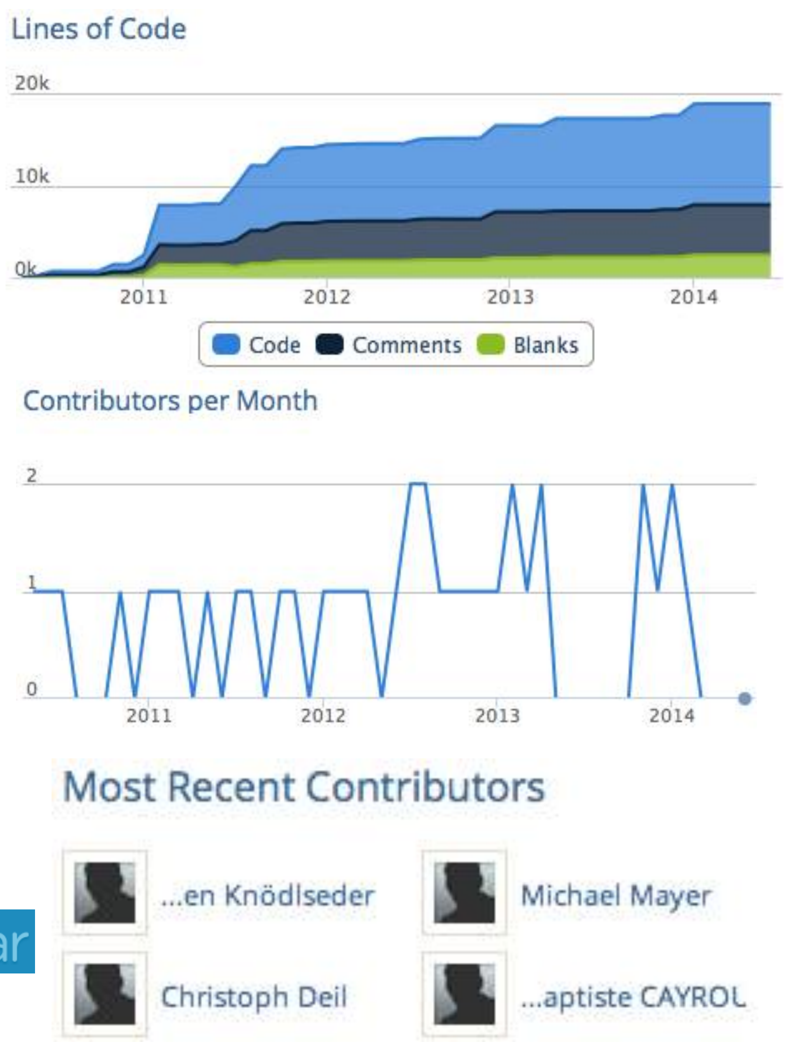
Contributors per Month

Most Recent Contributors

Michael Mayer	Rolf Buehler
Karl Kosack	Lucie Gerard
ellisowen	...en Knödlseder

Gravatar

for your picture!



Lines of Code

Contributors per Month

Most Recent Contributors

...en Knödlseder	Michael Mayer
Christoph Deil	...aptiste CAYROL

Current status

GammaLib

- Release 00-08-01
- 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 00-07-01
- Provides observation simulation, event selection, binning, model fitting, sky mapping
- Supports multi-instrument fitting

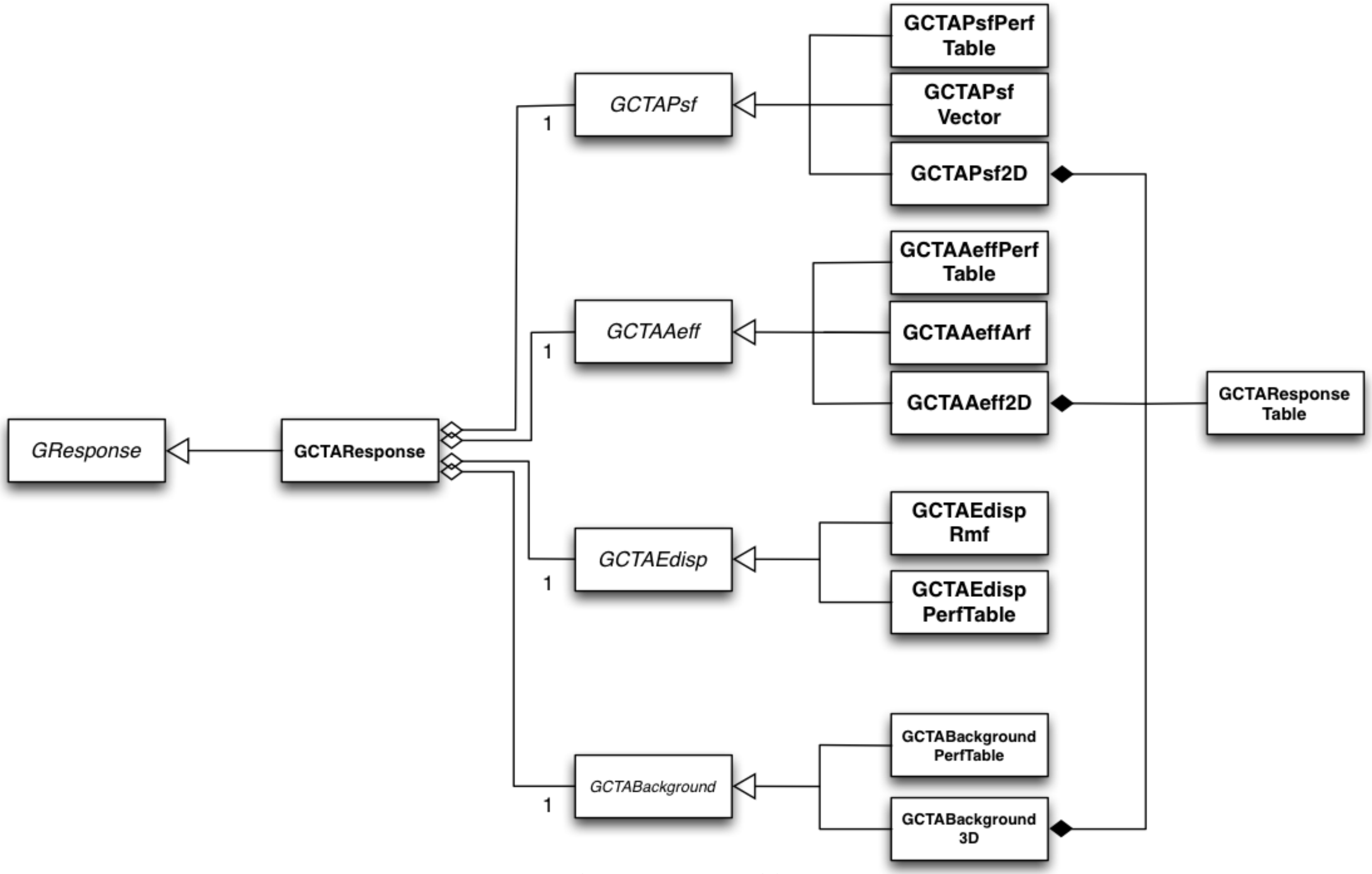
Current IRF handling

$$R_\gamma(\alpha',\delta',E'|\alpha,\delta,E,\vec{a}) = \underbrace{A_\gamma(\alpha,\delta,E,\vec{a})}_{\text{effective area (cm2)}} \times \underbrace{PSF(\alpha',\delta'|\alpha,\delta,E,\vec{a})}_{\text{point spread function}} \times \underbrace{D(E'|\alpha,\delta,E,\vec{a})}_{\text{energy dispersion}}$$

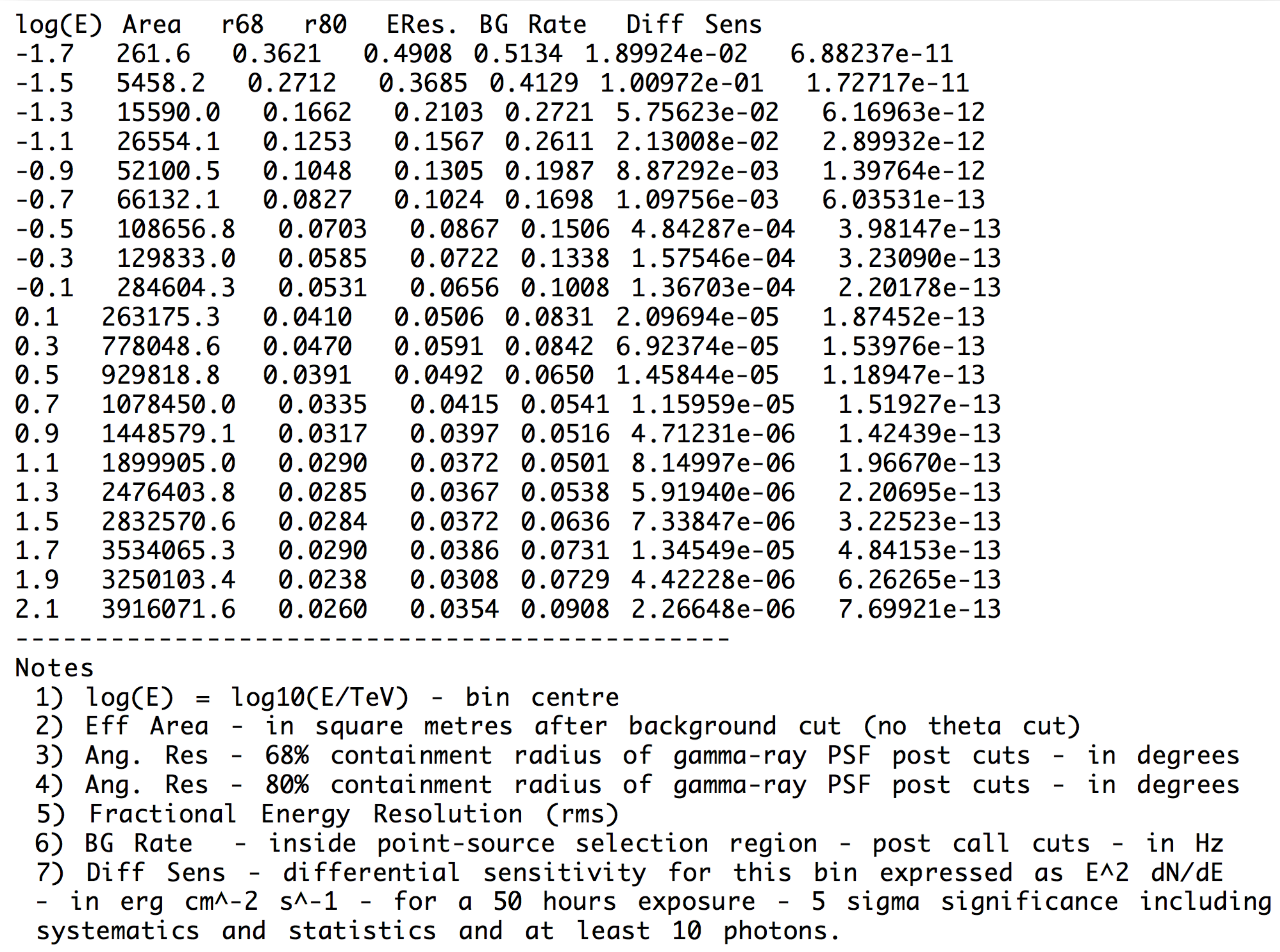
Full area, no angle cuts

$$1 = \int d\alpha'\, d\delta'\, PSF(\alpha',\delta'|\alpha,\delta,E,\vec{a})$$

$$1 = \int dE'\, D(E'|\alpha,\delta,E,\vec{a})$$



Performance tables (historic)



```
log(E) Area    r68   r80   ERes.  BG Rate    Diff  Sens
-1.7    261.6    0.3621   0.4908  0.5134  1.89924e-02   6.88237e-11
-1.5    5458.2    0.2712   0.3685  0.4129  1.00972e-01   1.72717e-11
-1.3    15590.0    0.1662   0.2103  0.2721  5.75623e-02   6.16963e-12
-1.1    26554.1    0.1253   0.1567  0.2611  2.13008e-02   2.89932e-12
-0.9    52100.5    0.1048   0.1305  0.1987  8.87292e-03   1.39764e-12
-0.7    66132.1    0.0827   0.1024  0.1698  1.09756e-03   6.03531e-13
-0.5    108656.8    0.0703   0.0867  0.1506  4.84287e-04   3.98147e-13
-0.3    129833.0    0.0585   0.0722  0.1338  1.57546e-04   3.23090e-13
-0.1    284604.3    0.0531   0.0656  0.1008  1.36703e-04   2.20178e-13
0.1   263175.3    0.0410   0.0506  0.0831  2.09694e-05   1.87452e-13
0.3   778048.6    0.0470   0.0591  0.0842  6.92374e-05   1.53976e-13
0.5   929818.8    0.0391   0.0492  0.0650  1.45844e-05   1.18947e-13
0.7   1078450.0    0.0335   0.0415  0.0541  1.15959e-05   1.51927e-13
0.9   1448579.1    0.0317   0.0397  0.0516  4.71231e-06   1.42439e-13
1.1   1899905.0    0.0290   0.0372  0.0501  8.14997e-06   1.96670e-13
1.3   2476403.8    0.0285   0.0367  0.0538  5.91940e-06   2.20695e-13
1.5   2832570.6    0.0284   0.0372  0.0636  7.33847e-06   3.22523e-13
1.7   3534065.3    0.0290   0.0386  0.0731  1.34549e-05   4.84153e-13
1.9   3250103.4    0.0238   0.0308  0.0729  4.42228e-06   6.26265e-13
2.1   3916071.6    0.0260   0.0354  0.0908  2.26648e-06   7.69921e-13
--------------------------------------------
Notes
  1) log(E) = log10(E/TeV) - bin centre
  2) Eff Area - in square metres after background cut (no theta cut)
  3) Ang. Res - 68% containment radius of gamma-ray PSF post cuts - in degrees
  4) Ang. Res - 80% containment radius of gamma-ray PSF post cuts - in degrees
  5) Fractional Energy Resolution (rms)
  6) BG Rate  - inside point-source selection region - post call cuts - in Hz
  7) Diff Sens - differential sensitivity for this bin expressed as E^2 dN/dE
  - in erg cm^-2 s^-1 - for a 50 hours exposure - 5 sigma significance including
  systematics and statistics and at least 10 photons.
```

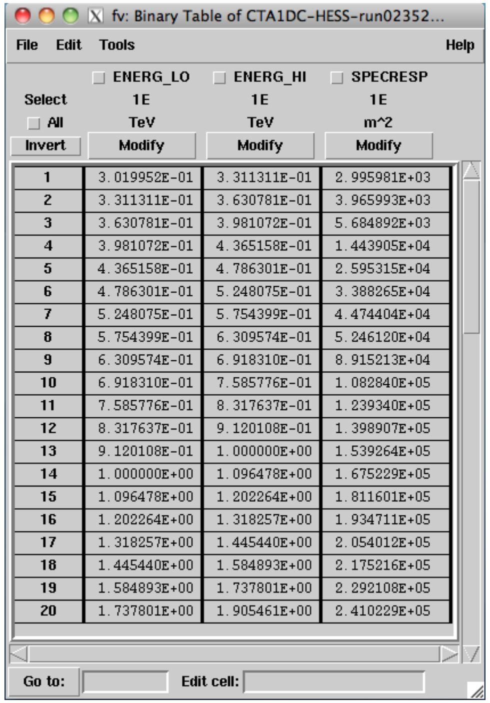
```
GCTAAeffPerfTable
GCTAPsfPerfTable
GCTAEdispPerfTable
GCTABackgroundPerfTable
```

Only on-axis information

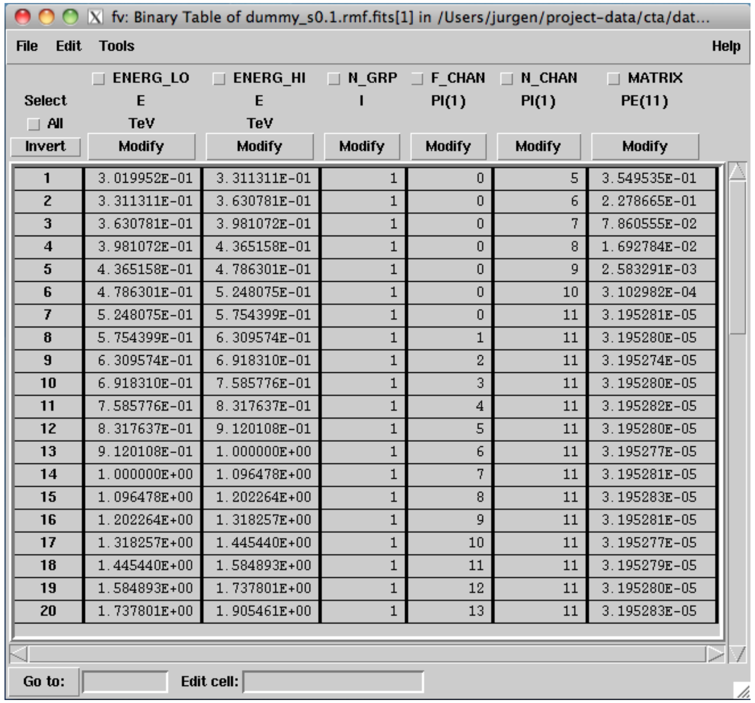
A_{eff} and B_{rate} off-axis dependence modelled using $B(\theta) \propto \exp\left(-\frac{1}{2}\frac{\theta^4}{\sigma^2}\right)$

Gaussians assumed for PSF and energy dispersion

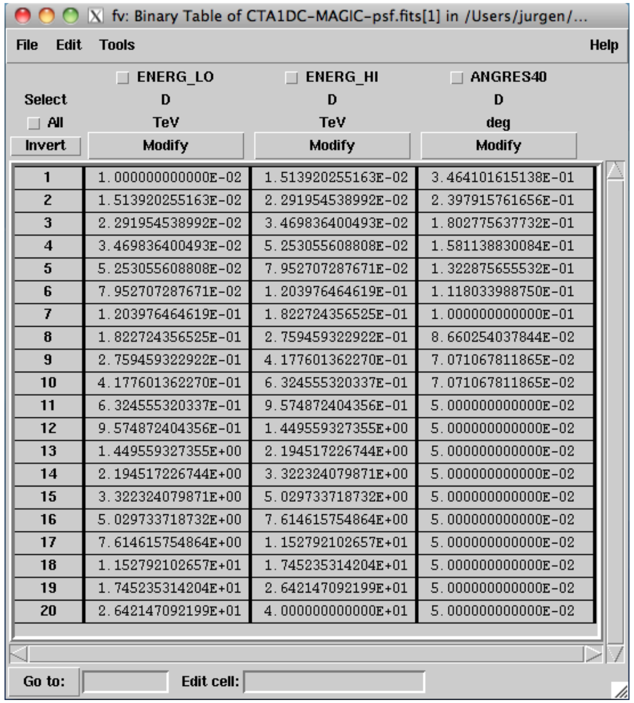
ARF, RMF, PSF vectors (1DC)


fv: Binary Table of CTA1DC-HESS-run02352...

	□ ENERG_LO	□ ENERG_HI	□ SPECRESP
Select	1E	1E	1E
□ All	TeV	TeV	m^2
Invert	Modify	Modify	Modify
1	3.019952E-01	3.311311E-01	2.995981E+03
2	3.311311E-01	3.630781E-01	3.965993E+03
3	3.630781E-01	3.981072E-01	5.684892E+03
4	3.981072E-01	4.365158E-01	1.443905E+04
5	4.365158E-01	4.786301E-01	2.595315E+04
6	4.786301E-01	5.248075E-01	3.388265E+04
7	5.248075E-01	5.754399E-01	4.474404E+04
8	5.754399E-01	6.309574E-01	5.246120E+04
9	6.309574E-01	6.918310E-01	8.915213E+04
10	6.918310E-01	7.585776E-01	1.082840E+05
11	7.585776E-01	8.317637E-01	1.239340E+05
12	8.317637E-01	9.120108E-01	1.398907E+05
13	9.120108E-01	1.000000E+00	1.539264E+05
14	1.000000E+00	1.096478E+00	1.675229E+05
15	1.096478E+00	1.202264E+00	1.811601E+05
16	1.202264E+00	1.318257E+00	1.934711E+05
17	1.318257E+00	1.445440E+00	2.054012E+05
18	1.445440E+00	1.584893E+00	2.175216E+05
19	1.584893E+00	1.737801E+00	2.292108E+05
20	1.737801E+00	1.905461E+00	2.410229E+05


fv: Binary Table of dummy_s0.1.rmf.fits[1] in /Users/jurgen/project-data/cta/dat...

	□ ENERG_LO	□ ENERG_HI	□ N_GRP	□ F_CHAN	□ N_CHAN	□ MATRIX
Select	E	E	I	PI(1)	PI(1)	PE(11)
□ All	TeV	TeV				
Invert	Modify	Modify	Modify	Modify	Modify	Modify
1	3.019952E-01	3.311311E-01	1	0	5	3.549535E-01
2	3.311311E-01	3.630781E-01	1	0	6	2.278665E-01
3	3.630781E-01	3.981072E-01	1	0	7	7.860555E-02
4	3.981072E-01	4.365158E-01	1	0	8	1.692784E-02
5	4.365158E-01	4.786301E-01	1	0	9	2.583291E-03
6	4.786301E-01	5.248075E-01	1	0	10	3.102982E-04
7	5.248075E-01	5.754399E-01	1	0	11	3.195281E-05
8	5.754399E-01	6.309574E-01	1	1	11	3.195280E-05
9	6.309574E-01	6.918310E-01	1	2	11	3.195274E-05
10	6.918310E-01	7.585776E-01	1	3	11	3.195280E-05
11	7.585776E-01	8.317637E-01	1	4	11	3.195282E-05
12	8.317637E-01	9.120108E-01	1	5	11	3.195280E-05
13	9.120108E-01	1.000000E+00	1	6	11	3.195277E-05
14	1.000000E+00	1.096478E+00	1	7	11	3.195281E-05
15	1.096478E+00	1.202264E+00	1	8	11	3.195283E-05
16	1.202264E+00	1.318257E+00	1	9	11	3.195281E-05
17	1.318257E+00	1.445440E+00	1	10	11	3.195277E-05
18	1.445440E+00	1.584893E+00	1	11	11	3.195279E-05
19	1.584893E+00	1.737801E+00	1	12	11	3.195280E-05
20	1.737801E+00	1.905461E+00	1	13	11	3.195283E-05


fv: Binary Table of CTA1DC-MAGIC-psf.fits[1] in /Users/jurgen/...

	□ ENERG_LO	□ ENERG_HI	□ ANGRES40
Select	D	D	D
□ All	TeV	TeV	deg
Invert	Modify	Modify	Modify
1	1.000000000000E-02	1.513920255163E-02	3.464101615138E-01
2	1.513920255163E-02	2.291954538992E-02	2.397915761656E-01
3	2.291954538992E-02	3.469836400493E-02	1.802775637732E-01
4	3.469836400493E-02	5.253055608808E-02	1.581138830084E-01
5	5.253055608808E-02	7.952707287671E-02	1.322875655532E-01
6	7.952707287671E-02	1.203976464619E-01	1.118033988750E-01
7	1.203976464619E-01	1.822724356525E-01	1.000000000000E-01
8	1.822724356525E-01	2.759459322922E-01	8.660254037844E-02
9	2.759459322922E-01	4.177601362270E-01	7.071067811865E-02
10	4.177601362270E-01	6.324555320337E-01	7.071067811865E-02
11	6.324555320337E-01	9.574872404356E-01	5.000000000000E-02
12	9.574872404356E-01	1.449559327355E+00	5.000000000000E-02
13	1.449559327355E+00	2.194517226744E+00	5.000000000000E-02
14	2.194517226744E+00	3.322324079871E+00	5.000000000000E-02
15	3.322324079871E+00	5.029733718732E+00	5.000000000000E-02
16	5.029733718732E+00	7.614615754864E+00	5.000000000000E-02
17	7.614615754864E+00	1.152792102657E+01	5.000000000000E-02
18	1.152792102657E+01	1.745235314204E+01	5.000000000000E-02
19	1.745235314204E+01	2.642147092199E+01	5.000000000000E-02
20	2.642147092199E+01	4.000000000000E+01	5.000000000000E-02

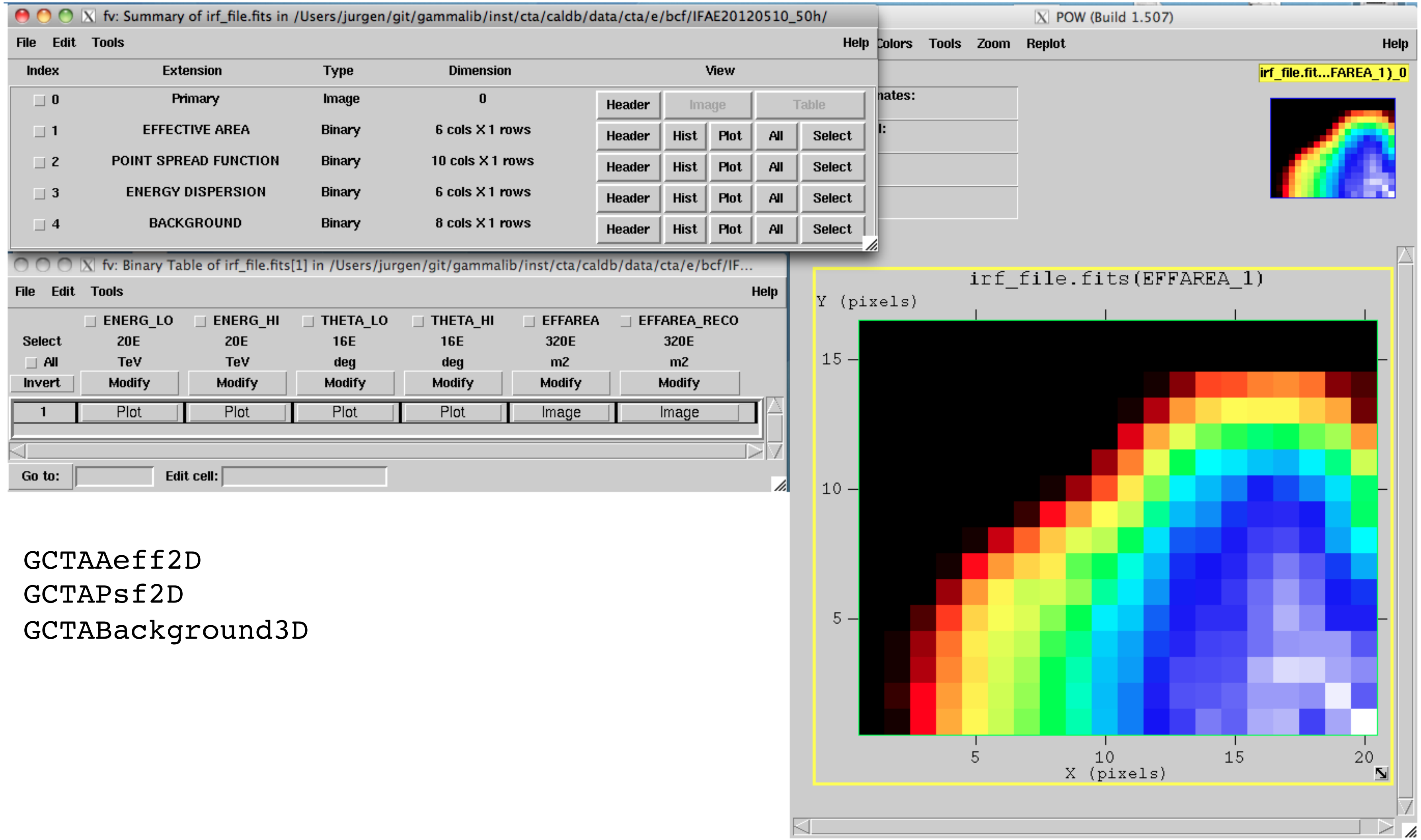
```
GCTAAeffArf
GCTAPsfVector
GCTAEdispRmf
```

Only on-axis information

A_{eff} and B_{rate} off-axis dependence modelled using $B(\theta) \propto \exp\left(-\frac{1}{2}\frac{\theta^4}{\sigma^2}\right)$

Gaussian assumed for PSF

Response cubes (new proposed standard)



fv: Summary of irf_file.fits in /Users/jurgen/git/gammalib/inst/cta/caldb/data/cta/e/bcf/IFAE20120510_50h/

Index	Extension	Type	Dimension	View
□ 0	Primary	Image	0	Header Image Table
□ 1	EFFECTIVE AREA	Binary	6 cols X 1 rows	Header Hist Plot All Select
□ 2	POINT SPREAD FUNCTION	Binary	10 cols X 1 rows	Header Hist Plot All Select
□ 3	ENERGY DISPERSION	Binary	6 cols X 1 rows	Header Hist Plot All Select
□ 4	BACKGROUND	Binary	8 cols X 1 rows	Header Hist Plot All Select

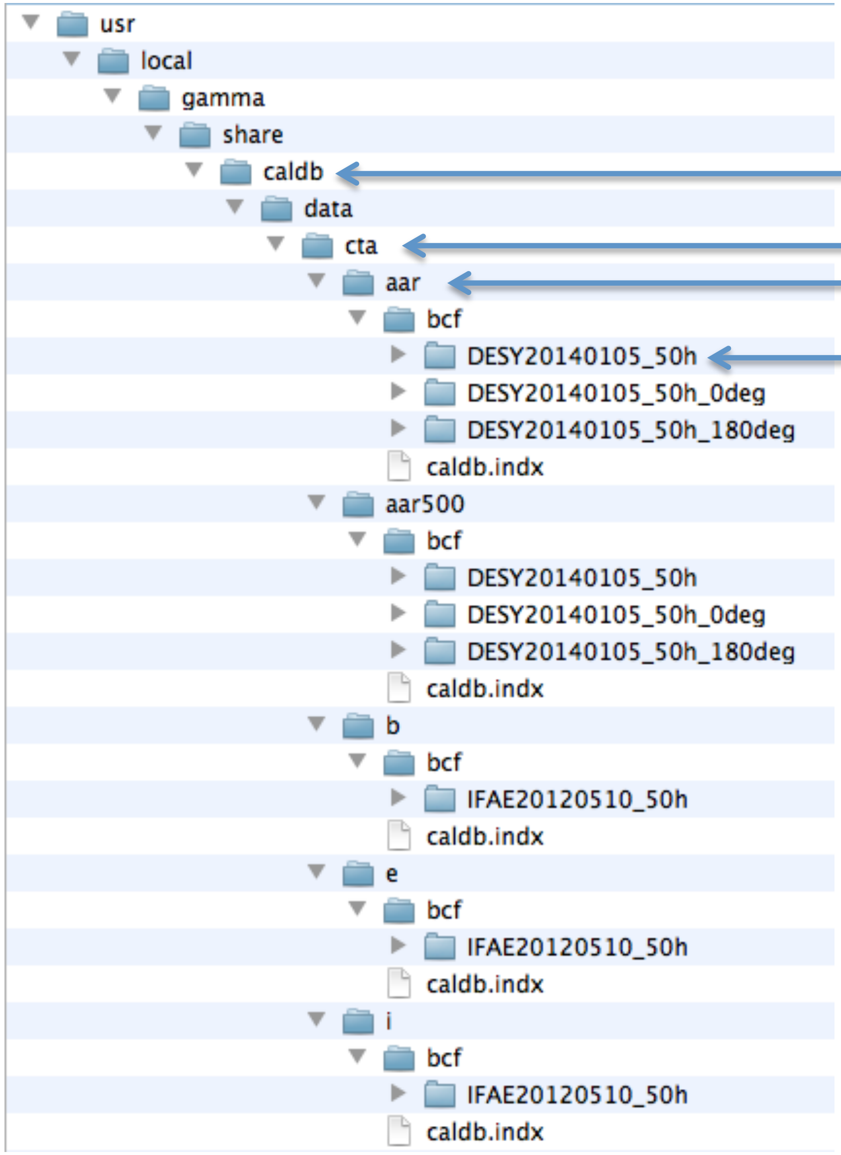
fv: Binary Table of irf_file.fits[1] in /Users/jurgen/git/gammalib/inst/cta/caldb/data/cta/e/bcf/IF...

Select	□ ENERG_LO	□ ENERG_HI	□ THETA_LO	□ THETA_HI	□ EFFAREA	□ EFFAREA_RECO
	20E	20E	16E	16E	320E	320E
□ All	TeV	TeV	deg	deg	m2	m2
Invert	Modify	Modify	Modify	Modify	Modify	Modify
1	Plot	Plot	Plot	Plot	Image	Image

Go to: Edit cell:

GCTAAeff2D
GCTAPsf2D
GCTABackground3D

Calibration database usage



```
usr
  local
    gamma
      share
        caldb
          data
            cta
              aar
                bcf
                  DESY20140105_50h
                  DESY20140105_50h_0deg
                  DESY20140105_50h_180deg
                caldb.indx
              aar500
                bcf
                  DESY20140105_50h
                  DESY20140105_50h_0deg
                  DESY20140105_50h_180deg
                caldb.indx
              b
                bcf
                  IFAE20120510_50h
                caldb.indx
              e
                bcf
                  IFAE20120510_50h
                caldb.indx
              i
                bcf
                  IFAE20120510_50h
                caldb.indx
```

Set calibration database root:

```
export CALDB=/usr/local/gamma/share/caldb
```

mission

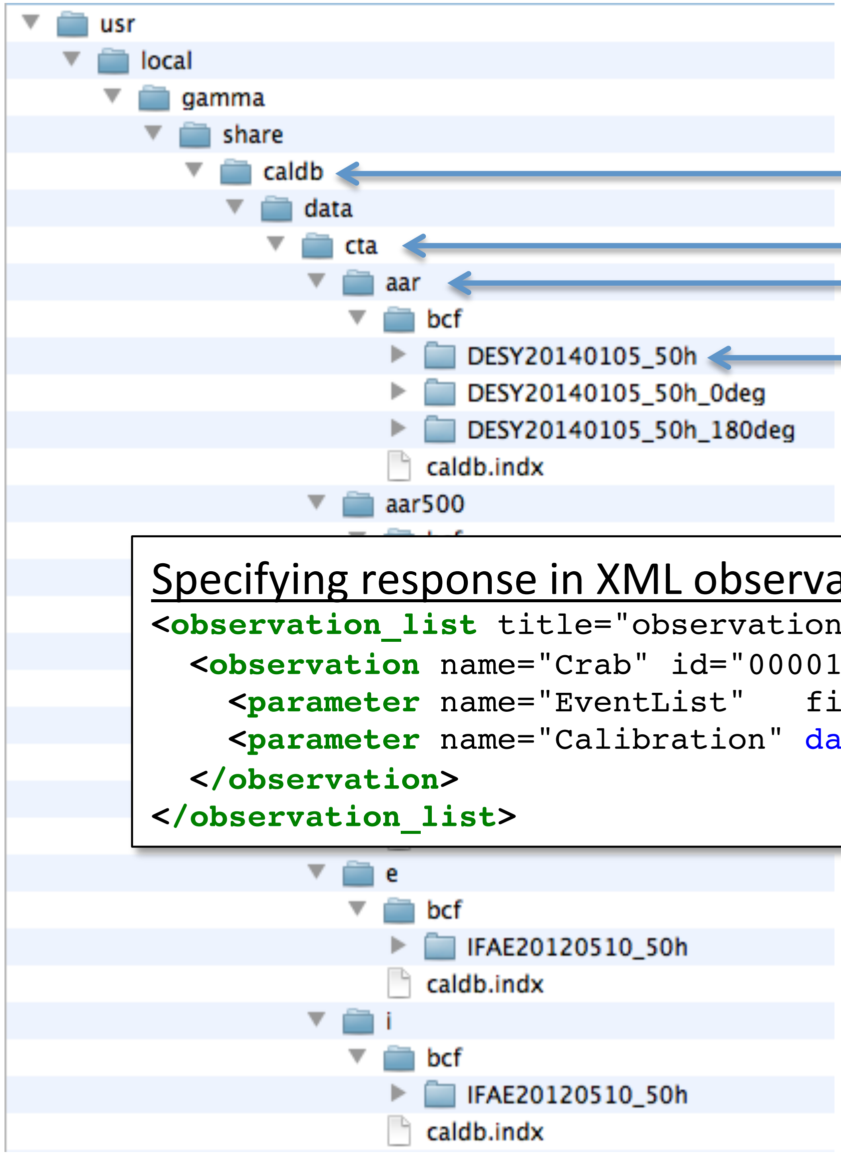
instrument

rspname

Specifying response as input parameters:

```
$ ctobssim
Model [$CTOOLS/share/models/crab.xml]
Calibration database [aar]
Instrument response function [DESY20140105_50h]
RA of pointing (degrees) (0-360) [83.63]
Dec of pointing (degrees) (-90-90) [22.01]
Radius of FOV (degrees) (0-180) [5.0]
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]
```

Calibration database usage



```
usr
  local
    gamma
      share
        caldb
          data
            cta
              aar
                bcf
                  DESY20140105_50h
                  DESY20140105_50h_0deg
                  DESY20140105_50h_180deg
                caldb.indx
              aar500
              e
                bcf
                  IFAE20120510_50h
                caldb.indx
              i
                bcf
                  IFAE20120510_50h
                caldb.indx
```

<u>Set calibration database root:</u>

```
export CALDB=/usr/local/gamma/share/caldb
```

mission

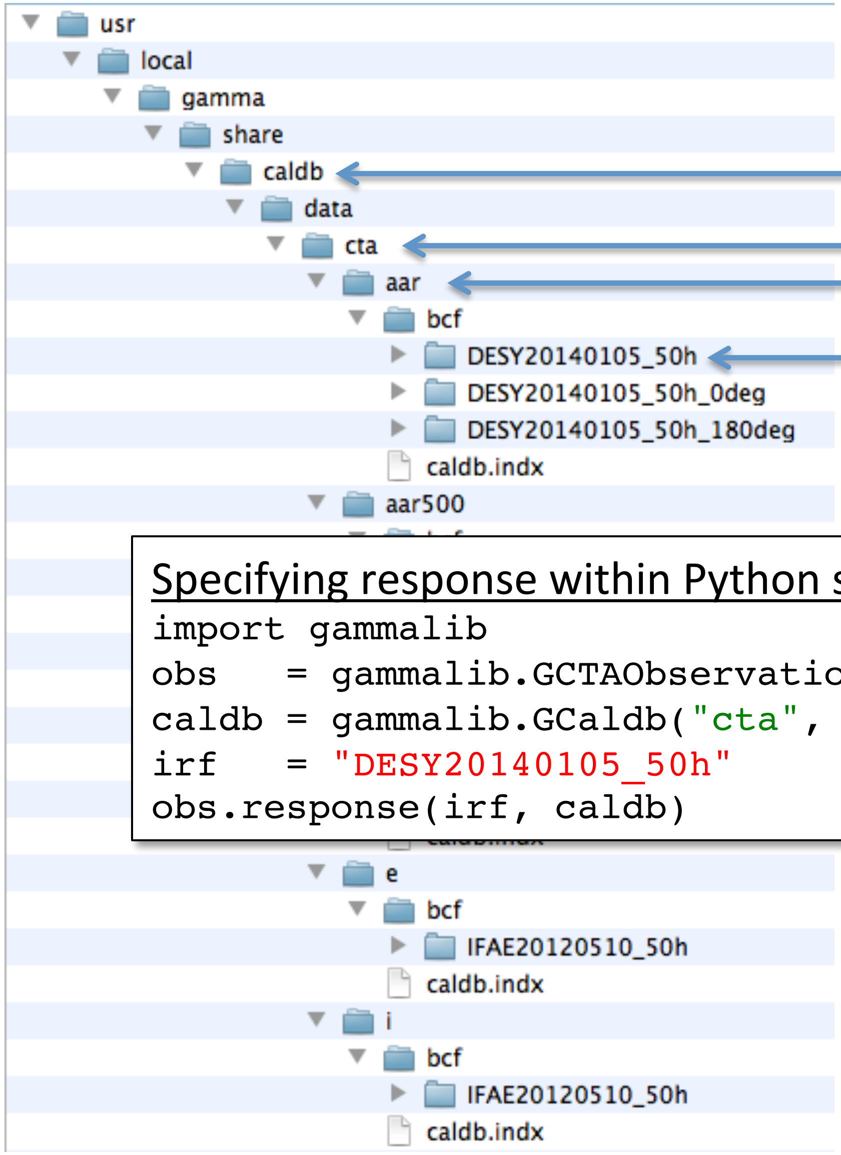
instrument

rspname

Specifying response in XML observation definition file:

```
<observation_list title="observation library">
  <observation name="Crab" id="00001" instrument="CTA">
    <parameter name="EventList"   file="events.fits"/>
    <parameter name="Calibration" database="aar" response="DESY20140105_50h"/>
  </observation>
</observation_list>
```

Calibration database usage



```
usr
  local
    gamma
      share
        caldb
          data
            cta
              aar
                bcf
                  DESY20140105_50h
                  DESY20140105_50h_0deg
                  DESY20140105_50h_180deg
                caldb.indx
              aar500
              e
                bcf
                  IFAE20120510_50h
                caldb.indx
              i
                bcf
                  IFAE20120510_50h
                caldb.indx
```

Set calibration database root:

```
export CALDB=/usr/local/gamma/share/caldb
```

Specifying response within Python script:

```
import gammalib
obs   = gammalib.GCTAObservation()
caldb = gammalib.GCaldb("cta", "aar")
irf   = "DESY20140105_50h"
obs.response(irf, caldb)
```

Calibration database summary

```
$ cscaldb debug=yes
2014-07-04T20:56:30: +============+
2014-07-04T20:56:30: | Parameters |
2014-07-04T20:56:30: +============+
2014-07-04T20:56:30:  chatter ...................: 2
2014-07-04T20:56:30:  clobber ...................: yes
2014-07-04T20:56:30:  debug .....................: yes
2014-07-04T20:56:30:  mode ......................: ql
2014-07-04T20:56:30:
2014-07-04T20:56:30: +==============+
2014-07-04T20:56:30: | Mission: cta |
2014-07-04T20:56:30: +==============+
2014-07-04T20:56:30: === Calibration: aar ===
2014-07-04T20:56:30: DESY20140105_50h
2014-07-04T20:56:30: DESY20140105_50h_0deg
2014-07-04T20:56:30: DESY20140105_50h_180deg
2014-07-04T20:56:30:
2014-07-04T20:56:30: === Calibration: aar500 ===
2014-07-04T20:56:30: DESY20140105_50h
2014-07-04T20:56:30: DESY20140105_50h_0deg
2014-07-04T20:56:30: DESY20140105_50h_180deg
2014-07-04T20:56:30:
2014-07-04T20:56:30: === Calibration: b ===
2014-07-04T20:56:30: IFAE20120510_50h
```

Calibration file usage

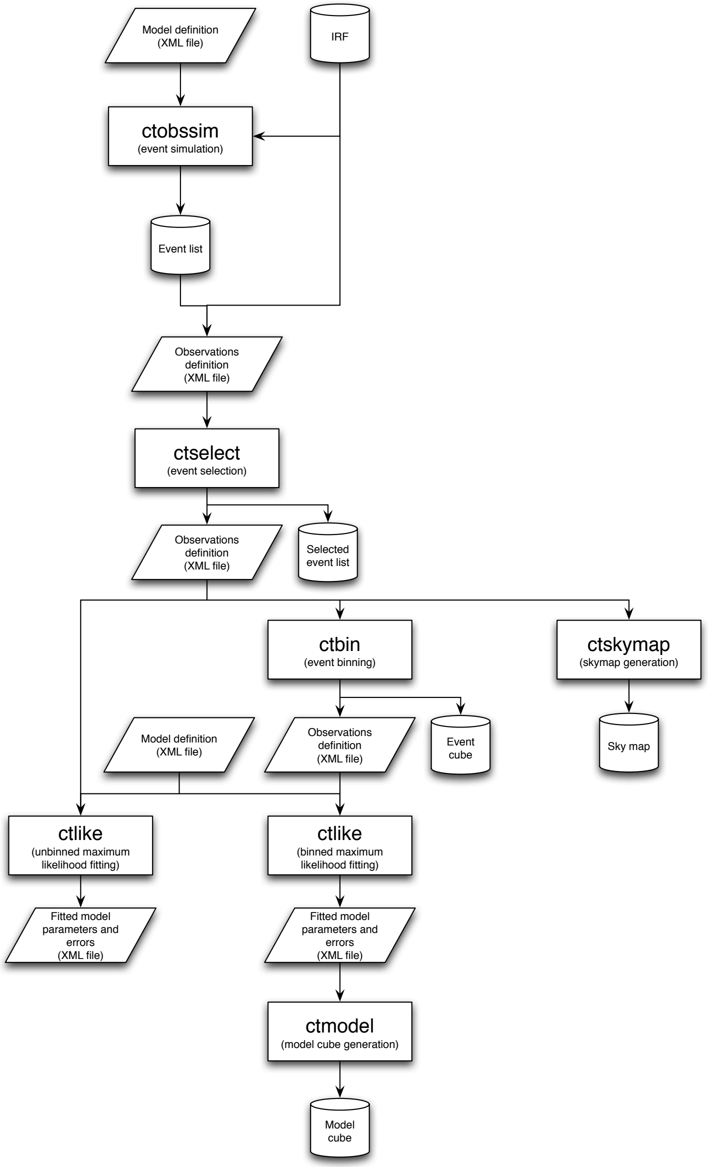
Specifying response in XML observation definition file:

```
<observation_list title="observation library">
  <observation name="Crab" id="00001" instrument="CTA">
    <parameter name="EventList"           file="events.fits"/>
    <parameter name="EffectiveArea"       file="$CALDB/data/cta/aar/bcf/DESY20140105_50h/irf.fits"/>
    <parameter name="PointSpreadFunction" file="$CALDB/data/cta/aar/bcf/DESY20140105_50h/irf.fits"/>
    <parameter name="EnergyDispersion"    file="$CALDB/data/cta/aar/bcf/DESY20140105_50h/irf.fits"/>
    <parameter name="Background"          file="$CALDB/data/cta/aar/bcf/DESY20140105_50h/irf.fits"/>
  </observation>
</observation_list>
```

Specifying response within Python script:

```
import gammalib
obs   = gammalib.GCTAObservation()
caldb = gammalib.GCaldb("$CALDB/data/cta/aar/bcf/DESY20140105_50h")
irf   = "irf.fits"
obs.response(irf, caldb)
```

Typical ctools workflows


Model definition
(XML file)
IRF
ctobssim
(event simulation)
Event list
Observations definition (XML file)
ctselect
(event selection)
Observations definition (XML file)
Selected event list
ctbin
(event binning)
ctskymap
(skymap generation)
Model definition
(XML file)
Observations definition (XML file)
Event cube
Sky map
ctlike
(unbinned maximum likelihood fitting)
ctlike
(binned maximum likelihood fitting)
Fitted model parameters and errors (XML file)
Fitted model parameters and errors (XML file)
ctmodel
(model cube generation)
Model cube

ctobssim

Simulation of CTA event list based on the IRF, a source & background model, and a given pointing direction.

- Does not yet consider visibility constraints.
- Can simulate multiple pointings / event lists from within Python (OpenMP support).

```
$ ctobssim
Model [$CTOOLS/share/models/crab.xml]
Calibration database [aar]
Instrument response function [DESY20140105_50h]
RA of pointing (degrees) (0-360) [83.63]
Dec of pointing (degrees) (-90-90) [22.01]
Radius of FOV (degrees) (0-180) [5.0]
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]
```

Proposed evolution:

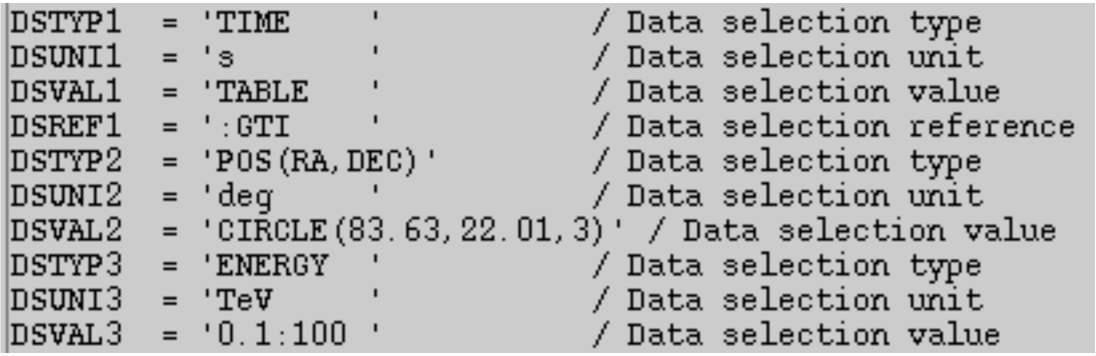
- Separate pointing definition from ctobssim (e.g. ctpntsim) to handle multiple pointings and to implement visibility constraints. Define pointing definition file (XML).

ctselect

Select CTA events from event file(s).

- Works on individual FITS files and observation definition XML files.

```
$ ctselect
Input event list or observation definition file [events.fits]
RA for ROI centre (degrees) (0-360) [83.63]
Dec for ROI centre (degrees) (-90-90) [22.01]
Radius of ROI (degrees) (0-180) [3.0]
Start time (CTA MET in seconds) (0) [0.0]
End time (CTA MET in seconds) (0) [0.0]
Lower energy limit (TeV) (0) [0.1]
Upper energy limit (TeV) (0) [100.0]
Output event list or observation definition file [selected_events.fits]
```



```
DSTYP1  = 'TIME    '           / Data selection type
DSUNI1  = 's       '           / Data selection unit
DSVAL1  = 'TABLE   '           / Data selection value
DSREF1  = ':GTI    '           / Data selection reference
DSTYP2  = 'POS(RA,DEC)'        / Data selection type
DSUNI2  = 'deg     '           / Data selection unit
DSVAL2  = 'CIRCLE(83.63,22.01,3)' / Data selection value
DSTYP3  = 'ENERGY  '           / Data selection type
DSUNI3  = 'TeV     '           / Data selection unit
DSVAL3  = '0.1:100 '           / Data selection value
```

Data selection keywords in FITS header

Proposed evolution:

None.

ctbin

Bin CTA events into 3D event cube (RA/GLON, DEC/GLAT, $\log_{10}$ energy).

- Works on individual FITS files and observation definition XML files.
- Coordinate system rotation (axisrot) not yet implemented.

```
$ ctbin
Input event list or observation definition file [events.fits]
First coordinate of image center in degrees (RA or galactic l) [83.63]
Second coordinate of image center in degrees (DEC or galactic b) [22.01]
Algorithm for defining energy bins (FILE|LIN|LOG) [LOG]
Start value for first energy bin in TeV [0.1]
Stop value for last energy bin in TeV [10]
Number of energy bins [10]
Projection method e.g. AIT|AZP|CAR|MER|STG|TAN (AIT|AZP|CAR|MER|STG|TAN) [TAN]
Coordinate system (CEL - celestial, GAL - galactic) (CEL|GAL) [GAL] CEL
Image scale (in degrees/pixel) [0.02]
Size of the X axis in pixels [200]
Size of the Y axis in pixels [200]
Output counts map or observation definition file [cntmap.fits]
```

Proposed evolution:

- Implement all WCS projections (GammaLib).
- Implement HealPix?
- Combination of several event lists into single cube? (or separate tool?)

ctlike

Fit parametric source & background model to events.

- Binned and unbinned maximum likelihood.
- Fits all GammaLib supported instrument data.
- Parallel computation of multiple observations (OpenMP).

```
$ ctlike
Event list, counts map or observation definition file [events.fits]
Calibration database [dummy] aar
Instrument response function [cta_dummy_irf] DESY20140105_50h
Source model [$CTOOLS/share/models/crab.xml]
Source model output file [crab_results.xml]
```

Proposed evolution:

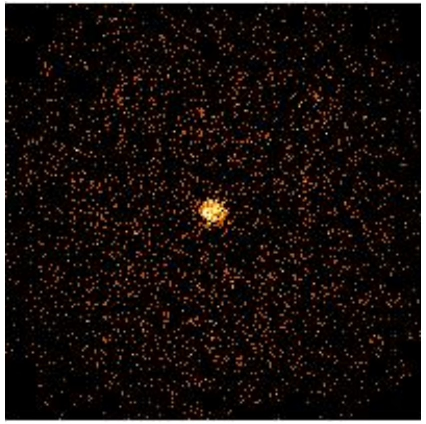
- Implement full Hessian computation for error estimation (GammaLib).
- Speed-up binned analysis (GammaLib).

ctmodel

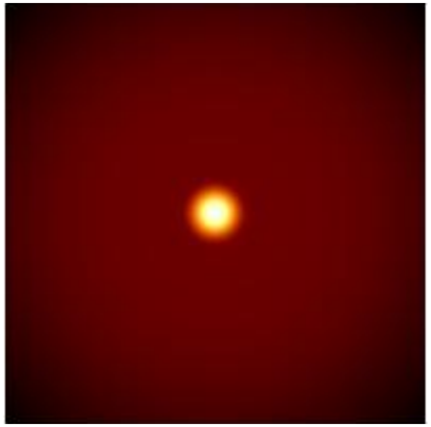
Create model counts map for a given model.

- Works on individual FITS files and observation definition XML files.
- Specifying counts map(s).

```
$ ctmodel
Input counts map or observation definition file [NONE] cntmap.fits
Output counts map or observation definition file [modmap.fits]
Calibration database [$CTOOLS/share/caldb/data/cta/dummy] aar
Instrument response function [cta_dummy_irf] DESY20140105_50h
Source model [$CTOOLS/share/models/crab.xml]
```



cntmap.fits



modmap.fits

Proposed evolution:

- None.

ctskymap

Bin CTA events into 2D sky map (RA/GLON, DEC/GLAT).

- Basically 2D version of ctbin.
- Works on individual FITS files only.
- Coordinate system rotation (axisrot) not yet implemented.

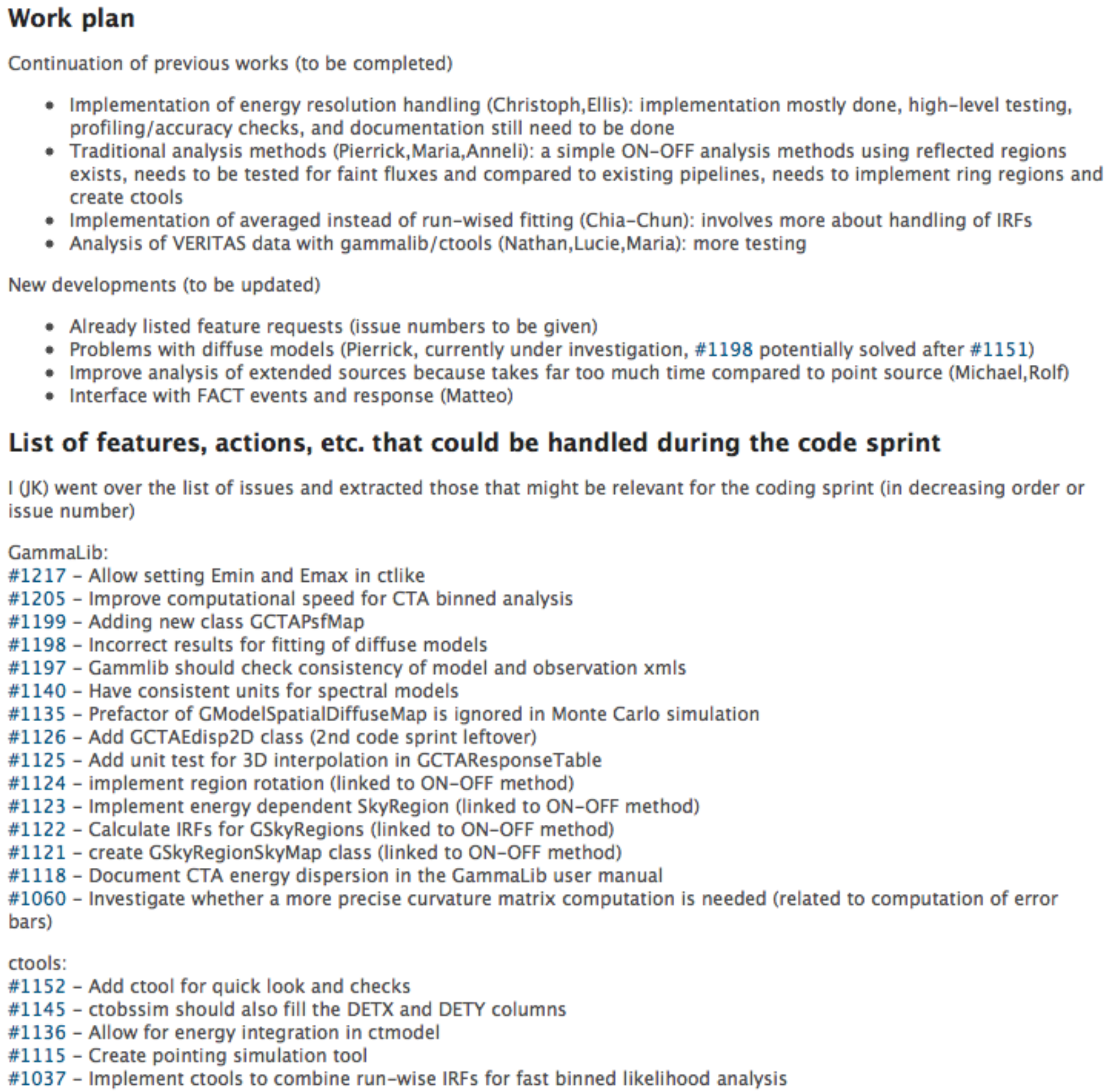
```
$ ctskymap
Output file name [skymap.fits]
Event data file name [events.fits]
First coordinate of image center in degrees (RA or galactic l) [83.63]
Second coordinate of image center in degrees (DEC or galactic b) [22.01]
Minimum energy in TeV [0.1]
Maximum energy in TeV [100.0]
Projection method e.g. AIT|AZP|CAR|MER|STG|TAN (AIT|AZP|CAR|MER|STG|TAN) [CAR]
Coordinate system (CEL - celestial, GAL - galactic) (CEL|GAL) [CEL]
Image scale (in degrees/pixel) [0.02]
Size of the X axis in pixels [200]
Size of the Y axis in pixels [200]
```

Proposed evolution:

- Combination of several event lists into single map.
- Implement imaging algorithms (e.g. ring background, template background, …)
- Implement all WCS projections (GammaLib).
- Implement HealPix?

4. Goals of this sprint

Goals of this sprint



Work plan

Continuation of previous works (to be completed)

- Implementation of energy resolution handling (Christoph,Ellis): implementation mostly done, high-level testing, profiling/accuracy checks, and documentation still need to be done
- Traditional analysis methods (Pierrick,Maria,Anneli): a simple ON-OFF analysis methods using reflected regions exists, needs to be tested for faint fluxes and compared to existing pipelines, needs to implement ring regions and create ctools
- Implementation of averaged instead of run-wised fitting (Chia-Chun): involves more about handling of IRFs
- Analysis of VERITAS data with gammalib/ctools (Nathan,Lucie,Maria): more testing

New developments (to be updated)

- Already listed feature requests (issue numbers to be given)
- Problems with diffuse models (Pierrick, currently under investigation, #1198 potentially solved after #1151)
- Improve analysis of extended sources because takes far too much time compared to point source (Michael,Rolf)
- Interface with FACT events and response (Matteo)

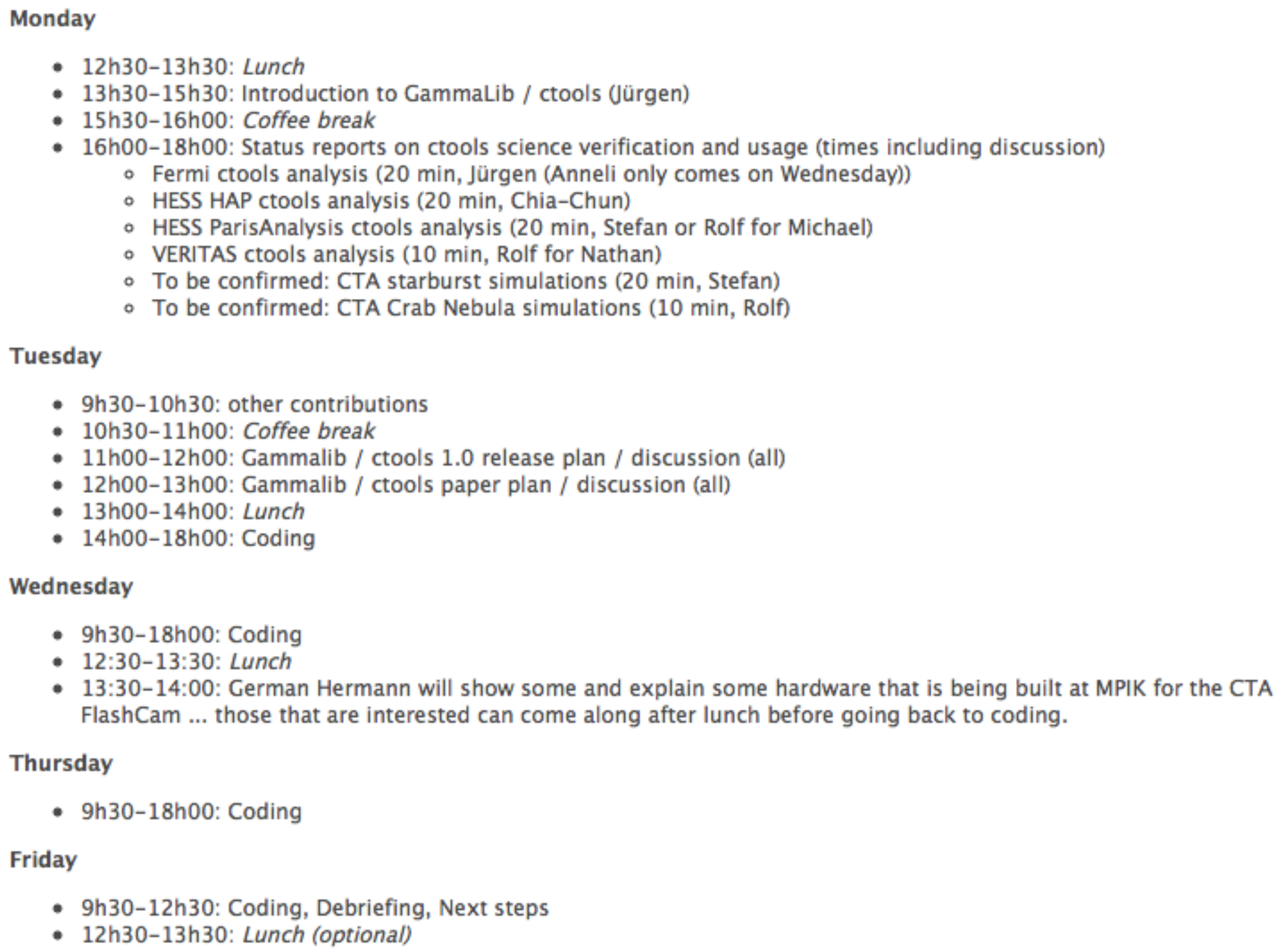
List of features, actions, etc. that could be handled during the code sprint

I (JK) went over the list of issues and extracted those that might be relevant for the coding sprint (in decreasing order or issue number)

GammaLib:
#1217 – Allow setting Emin and Emax in ctlike
#1205 – Improve computational speed for CTA binned analysis
#1199 – Adding new class GCTAPsfMap
#1198 – Incorrect results for fitting of diffuse models
#1197 – Gammlib should check consistency of model and observation xmls
#1140 – Have consistent units for spectral models
#1135 – Prefactor of GModelSpatialDiffuseMap is ignored in Monte Carlo simulation
#1126 – Add GCTAEdisp2D class (2nd code sprint leftover)
#1125 – Add unit test for 3D interpolation in GCTAResponseTable
#1124 – implement region rotation (linked to ON-OFF method)
#1123 – Implement energy dependent SkyRegion (linked to ON-OFF method)
#1122 – Calculate IRFs for GSkyRegions (linked to ON-OFF method)
#1121 – create GSkyRegionSkyMap class (linked to ON-OFF method)
#1118 – Document CTA energy dispersion in the GammaLib user manual
#1060 – Investigate whether a more precise curvature matrix computation is needed (related to computation of error bars)

ctools:
#1152 – Add ctool for quick look and checks
#1145 – ctobssim should also fill the DETX and DETY columns
#1136 – Allow for energy integration in ctmodel
#1115 – Create pointing simulation tool
#1037 – Implement ctools to combine run-wise IRFs for fast binned likelihood analysis

Agenda



Monday

- 12h30–13h30: *Lunch*
- 13h30–15h30: Introduction to GammaLib / ctools (Jürgen)
- 15h30–16h00: *Coffee break*
- 16h00–18h00: Status reports on ctools science verification and usage (times including discussion)
  - Fermi ctools analysis (20 min, Jürgen (Anneli only comes on Wednesday))
  - HESS HAP ctools analysis (20 min, Chia-Chun)
  - HESS ParisAnalysis ctools analysis (20 min, Stefan or Rolf for Michael)
  - VERITAS ctools analysis (10 min, Rolf for Nathan)
  - To be confirmed: CTA starburst simulations (20 min, Stefan)
  - To be confirmed: CTA Crab Nebula simulations (10 min, Rolf)

Tuesday

- 9h30–10h30: other contributions
- 10h30–11h00: *Coffee break*
- 11h00–12h00: Gammalib / ctools 1.0 release plan / discussion (all)
- 12h00–13h00: Gammalib / ctools paper plan / discussion (all)
- 13h00–14h00: *Lunch*
- 14h00–18h00: Coding

Wednesday

- 9h30–18h00: Coding
- 12:30–13:30: *Lunch*
- 13:30–14:00: German Hermann will show some and explain some hardware that is being built at MPIK for the CTA FlashCam ... those that are interested can come along after lunch before going back to coding.

Thursday

- 9h30–18h00: Coding

Friday

- 9h30–12h30: Coding, Debriefing, Next steps
- 12h30–13h30: *Lunch (optional)*