9th ctools coding sprint

Progressing with the code base

Implementing new features and tools

Have fun!

Agenda

- Today: Introduction & Definition of Projects
  - Learning ctools & GammaLib development in an hour
  - Discussion of projects
  - Team formation
- Tuesday – Thursday: Coding
  - Dinner on Tuesday? 20h?
- Friday: Project Status & Outlook

Learning ctools and GammaLib development in an hour

What you should know

- How to write C++ and/or Python code
- How to use Git
- Our GitLab development workflow
  (see http://cta.irap.omp.eu/ctools/develop/git/index.html)
- Our v1.0.0 reference paper (mathematics)
  (see http://www.aanda.org/articles/aa/abs/2016/09/aa28822-16/aa28822-16.html)
- Our coding conventions

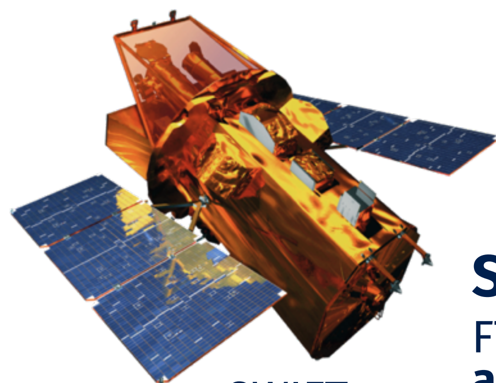
What you will learn now

- GammaLib and ctools introduction & overview
- Adding a new ctool to ctools
- Adding a new cscript to ctools

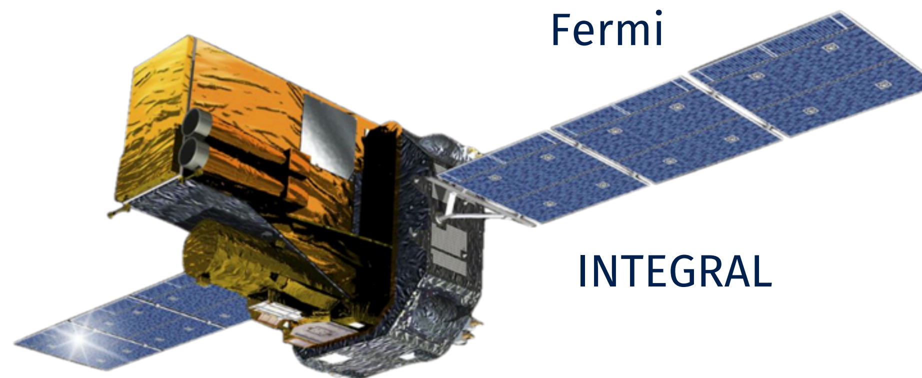
Where do the ctools come from?

Chandra

Fermi



SWIFT

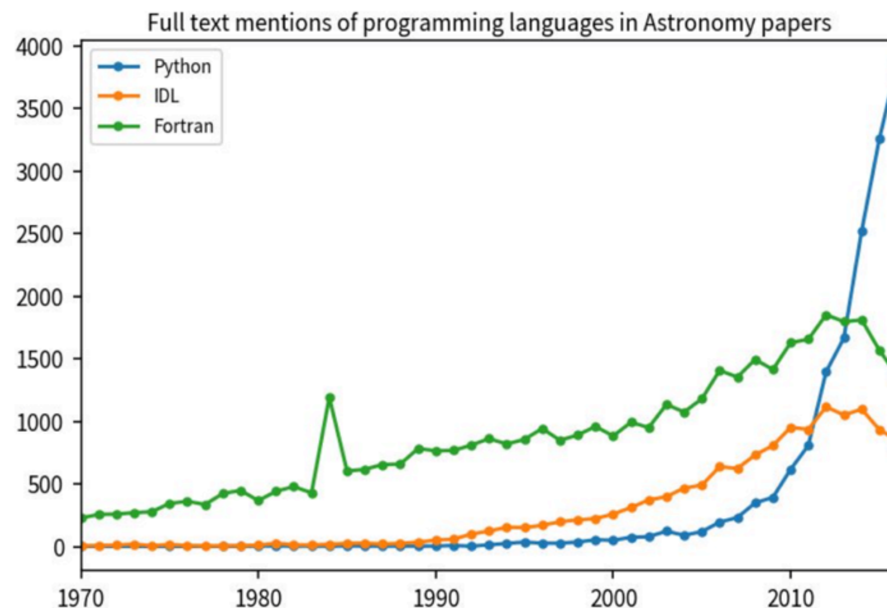


INTEGRAL

Standards

FTOOLS (like) analysis has become a **standard in high-energy astronomy**. Thousands of astronomers are today **familiar** with this standard. FTOOLS allows to setup User transparent **workflows**, producing well defined intermediate products for checks and verifications.

Where do the ctools come from?



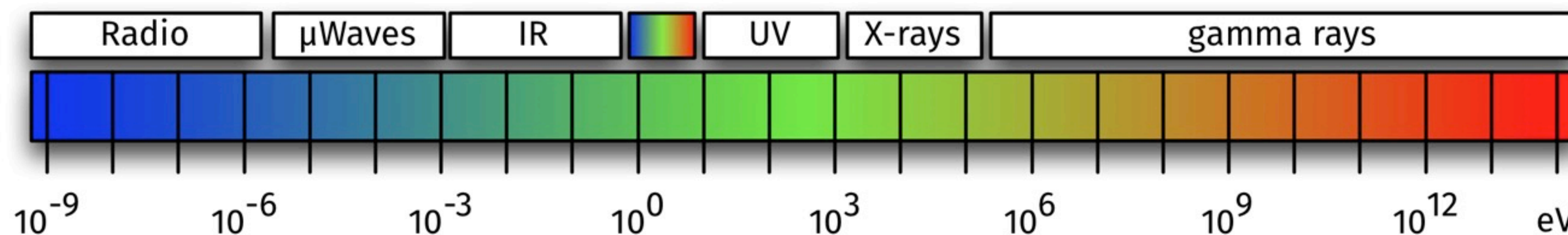
Credits: Thomas Robitaille

ctools are **intentionally** very similar to the Fermi/LAT Science Tools

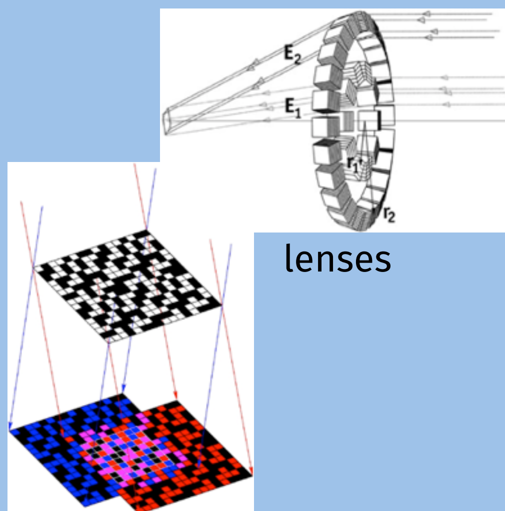
- Fermi/LAT Science Tools proven successful
- Basically **no learning** curve for **Fermi/LAT users**
- **Low learning curve** for users of other **HE observatories**
- But admittedly **some learning curve** for people from the **VHE community**

… and we follow the current user preferences by intrinsically integrating the ctools into **Python**

Observing gamma-rays



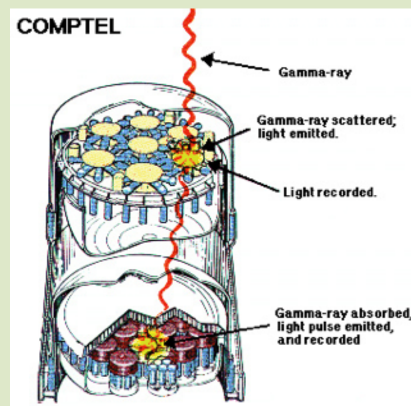
photoelectric effect



lenses

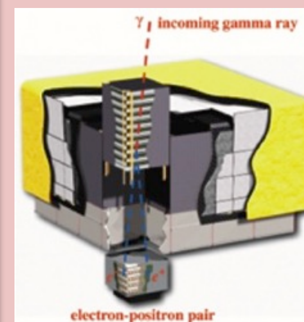
coded masks

Compton scattering



Compton telescopes

pair creation



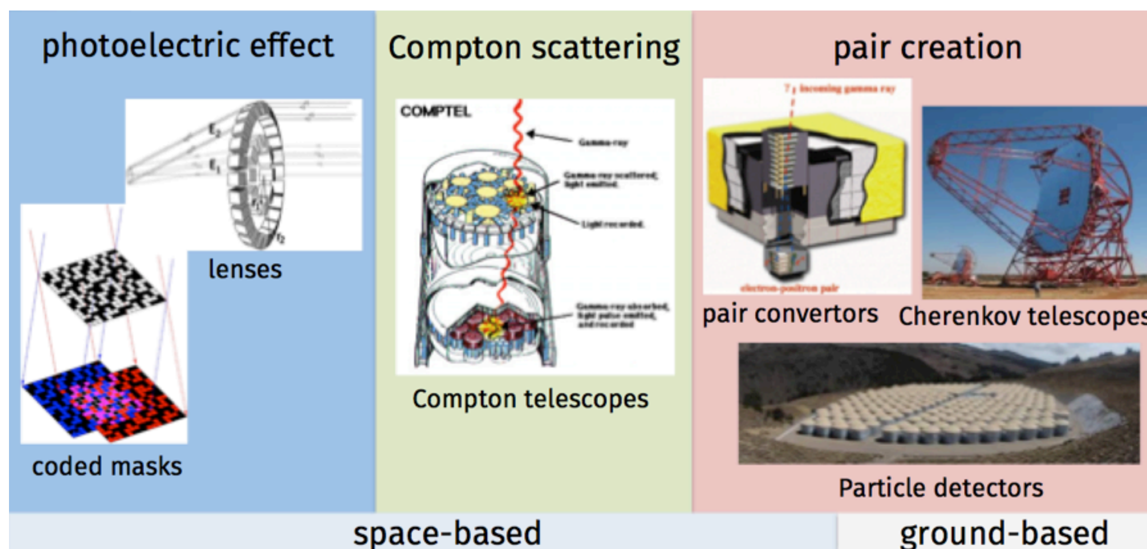
pair convertors

Cherenkov telescopes

Particle detectors

space-based | ground-based

What is GammaLib?



All gamma-ray telescopes measure individual photons as events. Existing high-energy analysis frameworks share a number of **common features** (FITS files, likelihood fitting, modular design) => It should be possible to **handle events from all gamma-ray telescopes with a single software framework** => **GammaLib**

CTA/IACT specific

is a IACT/CTA frontend to

generic

ctools paradigms

- The User is King
  - User driven code development
  - Put as less constraints as possible on the User, e.g. no reference platform
  - Make installation easy (not every User is a software geek)
  - Think multi-wavelength & Virtual Observatory
  - Provide ample documentation and tutorials
- Avoid dependencies
  - Lessons learned from past projects all point out that dependencies create a maintenance problem
  - Dependencies complicate also the software installation ("dependency hell")
  - Only implement what's needed (slim code base)
- Assure code quality
  - Enforce coding standards & code reviews
  - Multi-platform continuous integration & delivery
  - Code quality monitoring & quality gates

What you should do

- Make sure you read and follow the coding conventions
  - C++98 ISO (many servers still don't support C++11)
  - Python 2.3 – 3.6 (may evolve since few systems nowadays with Python 2.5-, but some with Python 2.6, and Python 2.7 supported at least until 2020)
  - Indent with 4 spaces, no tabs
  - Do not exceed 80 characters in one line
  - …
- Use coherent class and method names (always same method for same function, e.g. clear(), size(), is_empty(), append(), insert() …)
- Only use Python standard library modules
- Document your code (code documentation with Doxygen)
- Document how to use your code (user documentation with Sphinx)
- Write unit tests (you may do this even before writing your code)
- If in doubt, ask for code review (can be nicely done using GitLab by everyone)

Using GammaLib as C++ library

```
/* __ Includes ___________________________________________________________ */
#include "GammaLib.hpp"


/***********************************************************************//**
 * @brief Create sky map
 *
 * This code illustrates the creation of a sky map.
 ***************************************************************************/
int main(void) {

    // Create a SNR shell model. The shell is centred on RA=0.3 deg and
    // Dec=0.1 deg. The shell has an inner radius of 0.5 deg and a width
    // of 0.1 deg.
    GSkyDir centre;
    centre.radec_deg(0.3, 0.1);
    GModelSpatialRadialShell model(centre, 0.5, 0.1);

    // Create an empty sky map in celestial coordinates using a cartesian
    // projection. The image is centred on RA=Dec=0, has a bin size of
    // 0.01 degrees and 300 pixels in RA and Dec
    double ra(0.0);
    double dec(0.0);
    double binsz(0.01);
    int    npix(300);
    GSkyMap image("CAR", "CEL", ra, dec, -binsz, binsz, npix, npix, 1);

    // Fill the sky map with the model image
    GEnergy energy; // Dummy (not relevant as model is not energy dependent)
    GTime   time;   // Dummy (not relevant as model is not time dependent)
    for (int index = 0; index < image.npix(); ++index) {
        GSkyDir dir   = image.inx2dir(index); // sky coordinate of pixel
        double  theta = centre.dist(dir);     // distance in radians
        image(index)  = model.eval(theta, energy, time);
    }

    // Save the image to a FITS file
    image.save("my_sky_map.fits", true);

    // Exit
    return 0;
}
```

C++ program that uses GammaLib as C++ library.

ctools are such C++ programs.

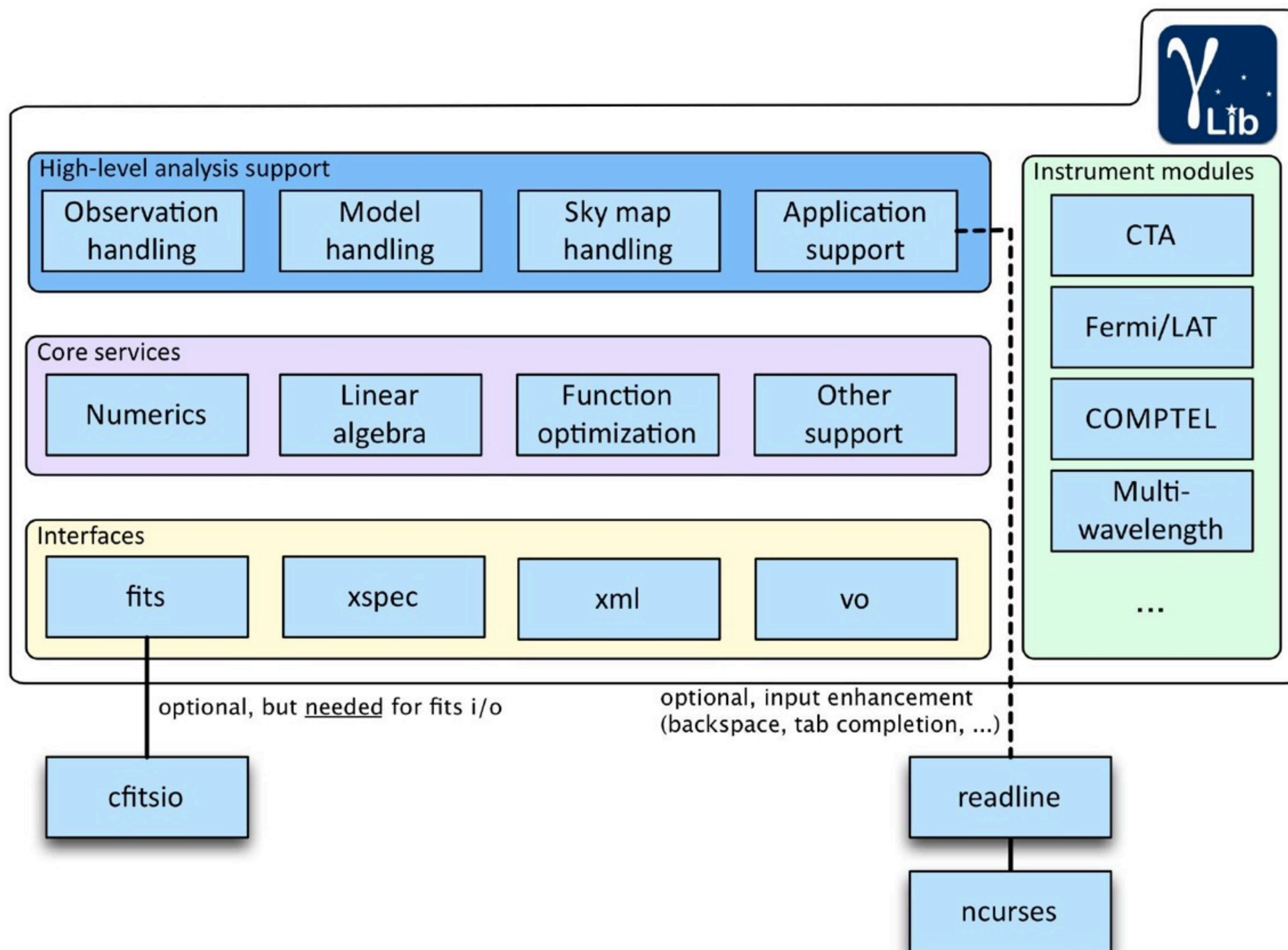
Using GammaLib as Python module

```
import gammalib


# ========================================== #
# Create a model container filled with models #
# ========================================== #
def create_models():
    """
    Create a model container filled with models
    """
    # Create model container
    models = gammalib.GModels()

    # Create a power law model for the Crab
    crabdir = gammalib.GSkyDir()
    crabdir.radec_deg(83.6331, 22.0145)
    spatial  = gammalib.GModelSpatialPointSource(crabdir)
    energy   = gammalib.GEnergy(100.0, "MeV")
    spectral = gammalib.GModelSpectralPlaw(5.7e-16, -2.48, energy)
    model    = gammalib.GModelSky(spatial, spectral)
    models.append(model)

    # Return models
    return models


# ======================== #
# Main routine entry point #
# ======================== #
if __name__ == '__main__':
    """
    Example script for creating and handling model.
    """
    # Dump header
    print("")
    print("******************************")
    print("* Example for model handling *")
    print("******************************")
    print("")

    # Create XML document
    models = create_models()
    print(models)
```

Python script that uses GammaLib as Python module.

cscripts are such Python scripts.

What is in GammaLib?



What you need as developer

- ANSI C++ compiler (e.g. g++, clang)
- Git
- make, automake, autoconf, libtools
- cfitsio
- readline & ncurses (nice to have for tab completion)
- Python (including Python.h header file)
- SWIG
- Example installation on Mac OS X (after Xcode install):

```
brew install automake
brew install libtool
brew install cfitsio
brew install swig
```

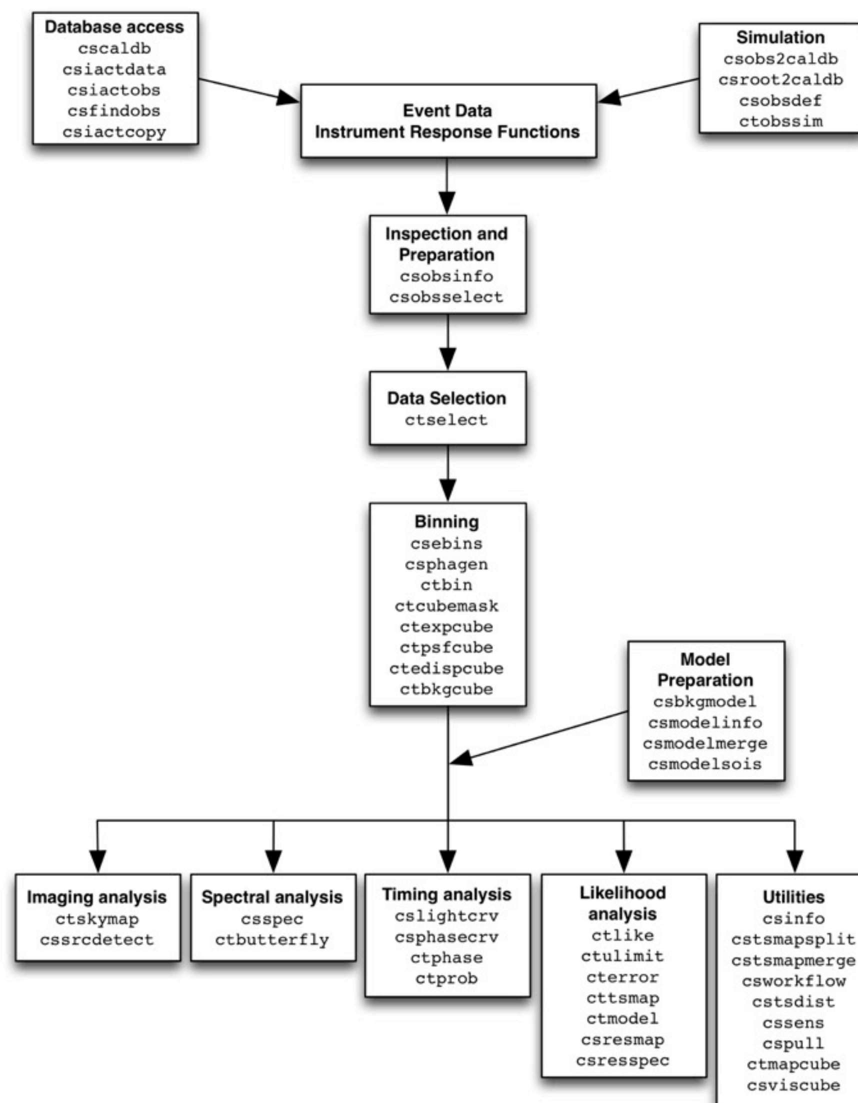
Using ctools/cscripts from command line

```
$ ctobssim
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]
End time (MET in s) [1800.0]
Lower energy limit (TeV) [0.1]
Upper energy limit (TeV) [100.0]
Calibration database [prod2]
Instrument response function [South_0.5h]
Input model XML file [$CTOOLS/share/models/crab.xml]
Output event data file or observation definition XML file [events.fits]

$ ctlike
Input event list, counts cube or observation definition XML file [events.fits]
Calibration database [prod2]
Instrument response function [South_0.5h]
Input model XML file [$CTOOLS/share/models/crab.xml]
Output model XML file [crab_results.xml]

$ csspec
Input event list, counts cube, or observation definition XML file [events.fits]
Calibration database [prod2]
Instrument response function [South_0.5h]
Input model XML file [$CTOOLS/share/models/crab.xml]
Source name [Crab]
Binning algorithm (LIN|LOG|FILE) [LOG]
Lower energy limit (TeV) [0.1]
Upper energy limit (TeV) [100.0]
Number of energy bins (0=unbinned) [20]
Output spectrum file [spectrum.fits]
```

Using ctools/cscripts as Python modules

```
import gammalib
import ctools
import cscripts


# ================================== #
# Simulation and analysis pipeline #
# ================================== #
def run_pipeline(obs, ra=83.63, dec=22.01, rad=3.0,
                 emin=0.1, emax=100.0,
                 tmin=0.0, tmax=0.0,
                 debug=False):
    """
    Simulation and binned analysis pipeline
    """
    # Simulate events
    sim = ctools.ctobssim(obs)
    sim['debug'] = debug
    sim.run()

    # Select events
    select = ctools.ctselect(sim.obs())
    select['ra']    = ra
    select['dec']   = dec
    select['rad']   = rad
    select['emin']  = emin
    select['emax']  = emax
    select['tmin']  = tmin
    select['tmax']  = tmax
    select['debug'] = debug
    select.run()

    # Perform maximum likelihood fitting
    like = ctools.ctlike(select.obs())
    like['debug'] = True # Switch this always on for results in console
    like.run()

    # Return
    return
```

What is in ctools?



Tutorial: adding a new ctool

Let's create a new tool to simulate pointings. Here are the steps:

1. Make sure that a Redmine issue exists for the new tool (#1632)
2. Find a good name for the tool (e.g. ctpntsim)
3. Create a new feature branch (e.g. 1632-add-ctpntsim)
4. Use code generator
   ```
   $ dev/codegen.py
   ```
   ← You need GammaLib installed and working for that
5. Reconfigure
   ```
   $ autoreconf
   $ ./configure
   $ make
   $ make check
   ```
6. Implement your code
7. Update ChangeLog and NEWS file
8. Commit the code
9. Push the code into your GitLab fork
10. Ask for code review or code integration

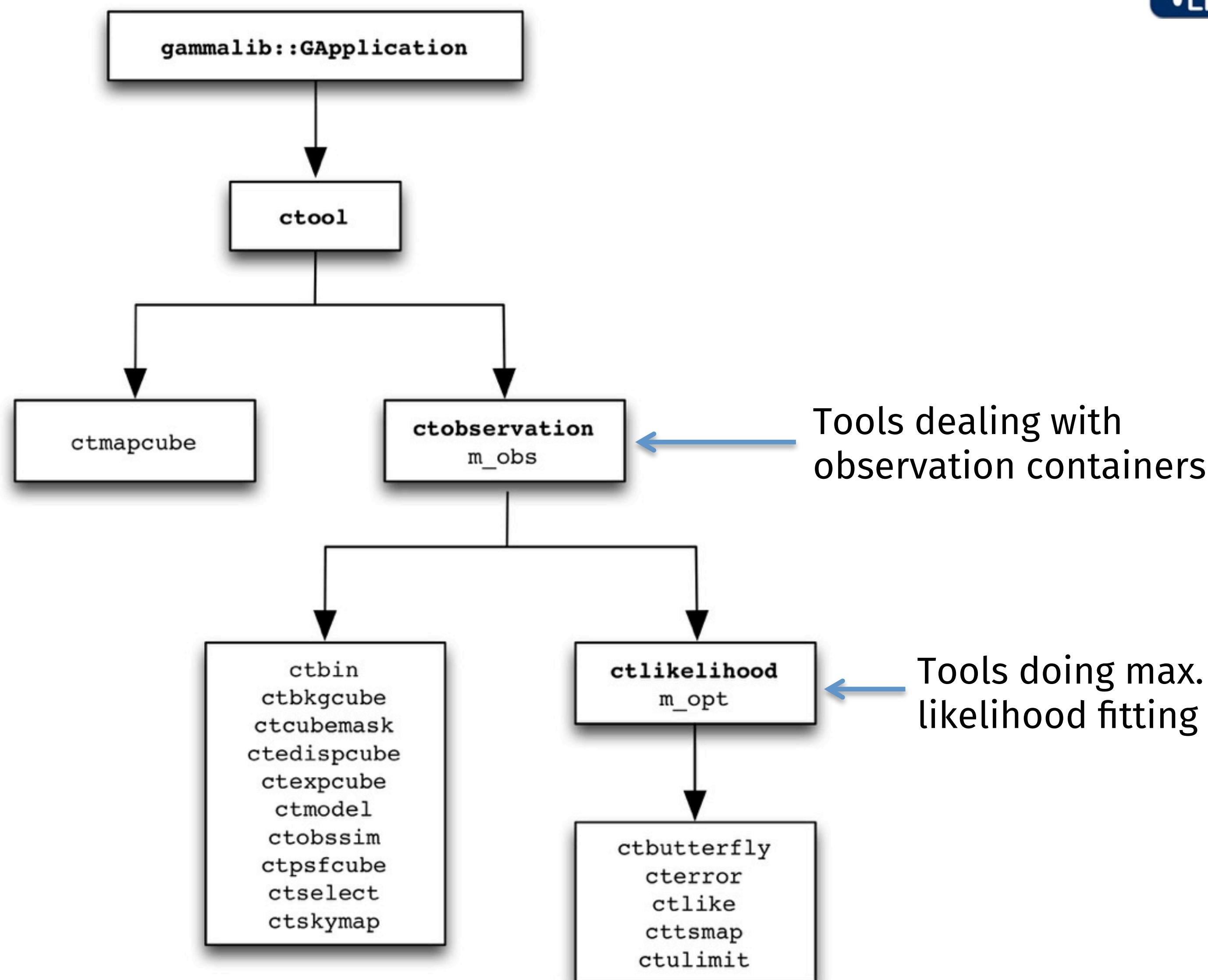
ctools code generator

```
ctools code generator
=====================

[1] Add ctool
[2] Add cscript
[q] Quit
Enter your choice: 1

Add ctool
---------
Please enter a ctool name (e.g. "ctpntsim"): ctpntsim
From which baseclass should the ctool derive?
[1] ctool
[2] ctobservation
[3] ctlikelihood
Enter your choice: 1
Please say what the tool is for (e.g. "Pointing simulation"): Pointing simulation
Please enter your name (e.g. "Joe Public"): Joe Public
Please enter your e-mail (e.g. "joe.public@dot.com"): joe.public@dot.com
Please enter your affiliation (e.g. "ESA"): ESA

All right. Have now:
ctool name .......: "ctpntsim"
Base class .......: "ctool"
Tools descriptor .: "Pointing simulation"
Your name ........: "Joe Public"
Your e-mail ......: "joe.public@dot.com"
Your affiliation .: "ESA"
Is this correct? (y/n): y

[1] Add ctool
[2] Add cscript
[q] Quit
Enter your choice: q
MacBook-Pro-de-Jurgen:ctools jurgen$
```

Add a ctool

Answer a few questions

ctool base classes



What the code generator did

```
MacBook-Pro-de-Jurgen:ctools jurgen$ git status
On branch add-codegen
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   configure.ac
        modified:   doc/source/users/reference_manual/index.rst
        modified:   pyext/Makefile.am
        modified:   pyext/ctools/tools.i
        modified:   src/Makefile.am
        modified:   test/test_python_ctools.py

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        doc/source/users/reference_manual/ctpntsim.rst
        pyext/ctpntsim.i
        src/ctpntsim/
        test/test_ctpntsim.py

no changes added to commit (use "git add" and/or "git commit -a")
MacBook-Pro-de-Jurgen:ctools jurgen$
```

Modified build system files (including unit test and reference documentation)

Added subfolder with minimal C++ code, Python interface, unit test script, and reference documentation. These are the files you need now to adapt.

Example: Python interface

```
/***************************************************************************
 *                  ctpntsim - Pointing simulation tool                    *
 * ----------------------------------------------------------------------- *
 *  copyright (C) 2018 by Joe Public                                       *
 * ----------------------------------------------------------------------- *
 *                                                                         *
 *  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 ctpntsim.i
 * @brief Pointing simulation tool definition
 * @author Joe Public
 */
%{
/* Put headers and other declarations here that are needed for compilation */
#include "ctpntsim.hpp"
%}


/***********************************************************************//**
 * @class ctpntsim
 *
 * @brief Pointing simulation tool
 ***************************************************************************/
class ctpntsim : public ctobservation {
public:
    // Constructors and destructors
    ctpntsim(void);
    explicit ctpntsim(const GObservations& obs);
    ctpntsim(int argc, char *argv[]);
    ctpntsim(const ctpntsim& app);
    virtual ~ctpntsim(void);

    // Methods
    void clear(void);
    void run(void);
    void save(void);
};


/***********************************************************************//**
 * @brief Pointing simulation tool Python extension
 ***************************************************************************/
%extend ctpntsim {
    ctpntsim copy() {
        return (*self);
    }
}
```

In this example the class was derived from the ctobservation base class

Class implements required methods. The tool works (but does nothing so far)

Example: Par file

```
#############################################################################
#                                                                           #
#                         CTA Science Analysis Tools                        #
#                                                                           #
# ------------------------------------------------------------------------- #
#                                                                           #
#  File:      ctpntsim.par                                                  #
#                                                                           #
#  Author:    Joe Public                                                    #
#             joe.public@dot.com                                            #
#             ESA                                                           #
#                                                                           #
#  Purpose:   Parameter file for the ctpntsim tool                          #
#                                                                           #
#############################################################################
#
# General parameters
#===================
# Your parameters go here

#
# Standard parameters
#====================
chatter,  i, h, 2,0,4, "Chattiness of output"
clobber,  b, h, yes,,, "Overwrite existing output files with new output files?"
debug,    b, h, no,,, "Activate debugging mode?"
mode,     s, h, ql,,, "Mode of automatic parameters"
logfile,  f, h, ctpntsim.log,,, "Log filename"
```

Only standard parameters exist

Example: code implementation

```
/***************************************************************************//**
 * @brief Run pointing simulation tool
 ***************************************************************************/
void ctpntsim::run(void)
{
    // If we're in debug mode then all output is also dumped on the screen
    if (logDebug()) {
        log.cout(true);
    }

    // Get task parameters
    get_parameters();

    // Write input observation container into logger
    log_observations(NORMAL, m_obs, "Input observation");

    // TODO: Your code goes here

    // Return
    return;
}


/***************************************************************************//**
 * @brief Save something
 *
 * Saves something.
 ***************************************************************************/
void ctpntsim::save(void)
{

    log_header1(TERSE, "Save something");

    // TODO: Your code goes here

    // Return
    return;
}
```

Main method

Need to implement in this method the querying of all User parameters

Here the real stuff happens

If something needs to be saved, do it here

Update reference documentation

```
.. _ctpntsim:

ctpntsim
==============================

ToDo: Describe in a one liner what the tool is doing.


Synopsis
--------

ToDo: Desribe in detail what the tool is doing, what it takes on input and
what it produces on output. Please do not write more than 80 characters per
line since this file is also used to produce a help text for the terminal.


General parameters
------------------

ToDo: Add here your parameters.


Standard parameters
-------------------

``(chatter = 2) [integer]``
    Verbosity of the executable:
     ``chatter = 0``: no information will be logged

     ``chatter = 1``: only errors will be logged

     ``chatter = 2``: errors and actions will be logged

     ``chatter = 3``: report about the task execution

     ``chatter = 4``: detailed report about the task execution

``(clobber = yes) [boolean]``
    Specifies whether an existing output file should be overwritten.

``(debug = no) [boolean]``
    Enables debug mode. In debug mode the executable will dump any log file outp

``(mode = ql) [string]``
    Mode of automatic parameters (default is ``ql``, i.e. "query and learn").

``(logfile = ctbin.log) [string]``
    Name of log file.


Related tools or scripts
------------------------

None
```

Try to write text in block that fits in 80 characters (also used for text shown when tool is executed with –help option).

Describe meaning of each parameter

Update index of reference documentation

```
Below you find links to the command line reference for all available tools and scripts.

ctools
------

.. toctree::
   :maxdepth: 1

   ctpntsim --- ToDo: Describe what tool is doing <ctpntsim>
   ctbin --- Generates counts cube <ctbin>
   ctbkgcube --- Generates background cube <ctbkgcube>
   ctbutterfly --- Compute butterfly <ctbutterfly>
   ctcubemask --- Filter counts cube <ctcubemask>
   ctedispcube --- Generates energy dispersion cube <ctedispcube>
   cterror --- Calculates likelihood profile errors <cterror>
   ctexpcube --- Generates exposure cube <ctexpcube>
   ctlike --- Performs maximum likelihood fitting <ctlike>
   ctmapcube --- Generates a map cube <ctmapcube>
   ctmodel --- Computes model counts cube <ctmodel>
   ctobssim --- Simulate observations <ctobssim>
   ctphase --- Computes the phase of each event <ctphase>
   ctprob --- Computes event probability for a given model <ctprob>
   ctpsfcube --- Generates point spread function cube <ctpsfcube>
   ctselect --- Selects event data <ctselect>
   ctskymap --- Generates sky map <ctskymap>
   cttsmap --- Generates Test Statistic map <cttsmap>
   ctulimit --- Calculates upper limit <ctulimit>
```

Describe what the tool is doing and move entry to alphabetically correct place

Tutorial: adding a new cscript

Let's create a new script to simulate pointings. Here are the steps:

1. Make sure that a Redmine issue exists for the new tool (#1632)
2. Find a good name for the tool (e.g. cspntsim)
3. Create a new feature branch (e.g. 1632-add-cspntsim)
4. Use code generator
   ```
   $ dev/codegen.py
   ```
5. Reconfigure
   ```
   $ autoreconf
   $ ./configure
   $ make check
   ```
6. Implement your code
7. Update ChangeLog and NEWS file
8. Commit the code
9. Push the code into your GitLab fork
10. Ask for code review or code integration

ctools code generator

```
ctools code generator
=====================

[1] Add ctool
[2] Add cscript
[q] Quit
Enter your choice: 2

Add cscript
-----------
Please enter a cscript name (e.g. "cspntsim"): cspntsim
From which baseclass should the cscript derive?
[1] cscript
[2] csobservation
[3] cslikelihood
Enter your choice: 1
Please say what the script is for (e.g. "Pointing simulation"): Pointing simulation
Please enter your name (e.g. "Joe Public"): Joe Public
Please enter your e-mail (e.g. "joe.public@dot.com"): joe.public@dot.com
Please enter your affiliation (e.g. "ESA"): ESA

All right. Have now:
cscript name .....: "cspntsim"
Base class .......: "cscript"
Script descriptor : "Pointing simulation"
Your name ........: "Joe Public"
Your e-mail ......: "joe.public@dot.com"
Your affiliation .: "ESA"
Is this correct? (y/n): y

[1] Add ctool
[2] Add cscript
[q] Quit
Enter your choice: q
MacBook-Pro-de-Jurgen:ctools jurgen$
```

Add a cscript

cscripts also have three possible base classes

What the code generator did

```
MacBook-Pro-de-Jurgen:ctools jurgen$ git status
On branch add-codegen
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   cscripts/Makefile.am
        modified:   cscripts/__init__.py.in
        modified:   doc/source/users/reference_manual/index.rst
        modified:   test/test_python_cscripts.py

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        cscripts/cspntsim.par
        cscripts/cspntsim.py
        doc/source/users/reference_manual/cspntsim.rst
        test/ctools_test_output.dmp
        test/test_cspntsim.py

no changes added to commit (use "git add" and/or "git commit -a")
MacBook-Pro-de-Jurgen:ctools jurgen$
```

Modified build system files (including unit test and reference documentation)

Added minimal Python code, par file, unit test script, and reference documentation. These are the files you need now to adapt.

cspntsim.py code adaption (1)

```
# ================== #
# cspntsim class #
# ================== #
class cspntsim(ctools.cscript):
    """
    Pointing simulation script
    """

    # Constructor
    def __init__(self, *argv):
        """
        Constructor

        Parameters
        ----------
        argv : list of str
            List of IRAF command line parameter strings of the form
            ``parameter=3``.
        """
        # Initialise application by calling the base class constructor
        self._init_cscript(self.__class__.__name__, ctools.__version__, argv)

        # Return
        return

    # State methods for pickling
    def __getstate__(self):
        """
        Extend ctools.cscript __getstate__ method

        Returns
        -------
        state : dict
            Pickled instance
        """
        # Set pickled dictionary
        state = {'base' : ctools.cscript.__getstate__(self)}

        # Return pickled dictionary
        return state

    def __setstate__(self, state):
        """
        Extend ctools.cscript __setstate__ method

        Parameters
        ----------
        state : dict
            Pickled instance
        """
        # Set state
        ctools.cscript.__setstate__(self, state['base'])

        # Return
        return
```

Properly align header

Add any private class members here

Add private class members to pickling dictionary

Initialise private class members from pickling dictionary

cspntsim.py code adaption (2)

```
# Private methods
def _get_parameters(self):
    """
    Get parameters from parfile
    """
    # TODO: Add code to query all relevant parameters

    #  Write input parameters into logger
    self._log_parameters(gammalib.TERSE)

    # Return
    return


# Public methods
def run(self):
    """
    Run the script
    """
    # Switch screen logging on in debug mode
    if self._logDebug():
        self._log.cout(True)

    # Get parameters
    self._get_parameters()

    # TODO: Your code goes here

    # Return
    return

def save(self):
    """
    Save something
    """
    # Write header
    self._log_header1(gammalib.TERSE, 'Save something')

    # TODO: Your code goes here

    # Return
    return
```

Query all relevant User parameters

Put the script code here.
Factorize the code by using private methods.

Save any results here (typically a FITS file)

NOW WE CAN START

CODING!

Backup tutorial

Adding a class to GammaLib
Unit testing

Tutorial: adding a class to GammaLib

Let's create a new spectral model (say an EBL model). Here are the steps:

1. Make sure that a Redmine issue exists for the new model
2. Find a good name for the class (e.g. GModelSpectralEBL)
3. Create a new feature branch (e.g. 2157-add-ebl-model)
4. Start by copying an existing class

```
$ cp src/model/GModelSpectralPlaw.cpp src/model/GModelSpectralEBL.cpp
$ cp include/GModelSpectralPlaw.hpp include/GModelSpectralEBL.hpp
$ cp pyext/GModelSpectralPlaw.i pyext/GModelSpectralEBL.i
```

5. Adapt code
6. Add new files to build system
7. Update ChangeLog and NEWS file
8. Commit the code
9. Push the code into your GitLab fork
10. Ask for code review or code integration

.hpp code adaption (1)

GModelSpectralEBL.hpp

GModelSpectralEBL.hpp 〉 class GModelSpectralEBL

Replace GModelSpectralPlaw

GModelSpectralEBL

```
/***************************************************************************
 *              GModelSpectralEBL.hpp - Spectral EBL model class           *
 * ----------------------------------------------------------------------- *
 *  copyright (C) 2016 by Joe Public                                       *
 * ----------------------------------------------------------------------- *
 *                                                                         *
 *  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 GModelSpectralEBL.hpp
 * @brief EBL spectral model class interface definition
 * @author Joe Public
 */

#ifndef GMODELSPECTRALEBL_HPP
#define GMODELSPECTRALEBL_HPP

/* __ Includes ___________________________________________________________ */
#include <string>
#include "GModelSpectral.hpp"
#include "GModelPar.hpp"
#include "GEnergy.hpp"

/* __ Forward declarations _______________________________________________ */
class GRan;
class GTime;
class GXmlElement;
```

- mass replacement
- your name and current year
- your name and purpose of file
- Adapt to class name (multiple include protection)
- Things for which you need the interface definition
- Things that need to be known to exist

.hpp code adaption (2)

```
class GModelSpectralEBL : public GModelSpectral {

public:
    // Constructors and destructors
    GModelSpectralEBL(void);
    GModelSpectralEBL(const double&  prefactor,
                      const double&  index,
                      const GEnergy& pivot);
    explicit GModelSpectralEBL(const GXmlElement& xml);
    GModelSpectralEBL(const GModelSpectralEBL& model);
    virtual ~GModelSpectralEBL(void);

    // Operators
    virtual GModelSpectralEBL& operator=(const GModelSpectralEBL& model);

    // Implemented pure virtual methods
    virtual void               clear(void);
    virtual GModelSpectralEBL* clone(void) const;
    virtual std::string        classname(void) const;
    virtual std::string        type(void) const;
    virtual double             eval(const GEnergy& srcEng,
                                    const GTime&   srcTime = GTime(),
                                    const bool&    gradients = false) const;
    virtual double             flux(const GEnergy& emin,
                                    const GEnergy& emax) const;
    virtual double             eflux(const GEnergy& emin,
                                     const GEnergy& emax) const;
    virtual GEnergy            mc(const GEnergy& emin,
                                  const GEnergy& emax,
                                  const GTime&   time,
                                  GRan&          ran) const;
    virtual void               read(const GXmlElement& xml);
    virtual void               write(GXmlElement& xml) const;
    virtual std::string        print(const GChatter& chatter = NORMAL) const;

    // Other methods
    double  prefactor(void) const;
    void    prefactor(const double& prefactor);

protected:
    // Protected methods
    void init_members(void);
    void copy_members(const GModelSpectralEBL& model);
    void free_members(void);

    // Protected members
    std::string m_type;              //!< Model type
    GModelPar   m_norm;              //!< Normalization factor
    GModelPar   m_index;             //!< Spectral index
    GModelPar   m_pivot;             //!< Pivot energy
};
```

use explicit to avoid automatic type conversion

Implement all pure virtual methods from the base class

add any other methods as needed

Standard methods for initialisation, copying and freeing members

Add members as needed (use mutable for pre-computed and cashed values)

.hpp code adaption (3)

```
/***************************************************************************//**
 * @brief Return class name
 *
 * @return String containing the class name ("GModelSpectralEBL").
 ***************************************************************************/
inline
std::string GModelSpectralEBL::classname(void) const
{
    return ("GModelSpectralEBL");
}


/***************************************************************************//**
 * @brief Return model type
 *
 * @return Model type.
 *
 * Returns the type of the spectral EBL model.
 ***************************************************************************/
inline
std::string GModelSpectralEBL::type(void) const
{
    return (m_type);
}


/***************************************************************************//**
 * @brief Return pre factor
 *
 * @return Pre factor (ph/cm2/s/MeV).
 *
 * Returns the pre factor.
 ***************************************************************************/
inline
double GModelSpectralEBL::prefactor(void) const
{
    return (m_norm.value());
}


/***************************************************************************//**
 * @brief Set pre factor
 *
 * @param[in] prefactor Pre factor (ph/cm2/s/MeV).
 *
 * Sets the pre factor.
 ***************************************************************************/
inline
void GModelSpectralEBL::prefactor(const double& prefactor)
{
    m_norm.value(prefactor);
    return;
}
```

add “one-liners” as inline methods

Every method has a single-line brief description

Provide detailed description of method, including formulae, references, etc. (everything a user of this method may need)

Document return value and units

Document input parameters and units

Doxygen code documentation

double GModelSpectralEBL::prefactor (void) const inline

Return pre factor.

Returns

Pre factor (ph/cm2/s/MeV).

Returns the pre factor.

Definition at line 147 of file GModelSpectralEBL.hpp.

References m_norm, and GOptimizerPar::value().

void GModelSpectralEBL::prefactor (const double & prefactor) inline

Set pre factor.

Parameters

`[in]` **prefactor** Pre factor (ph/cm2/s/MeV).

Sets the pre factor.

Definition at line 161 of file GModelSpectralEBL.hpp.

References m_index, m_norm, m_pivot, GEnergy::MeV(), and GOptimizerPar::value().

.cpp code adaption (1)

```
/***************************************************************************
 *          GModelSpectralEBL.cpp - Spectral EBL model class             *
 * ----------------------------------------------------------------------- *
 *  copyright (C) 2016 by Joe Public                                       *
 * ----------------------------------------------------------------------- *
 *                                                                         *
 *  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 GModelSpectralEBL.cpp
 * @brief EBL spectral model class implementation
 * @author Joe Public
 */

/* __ Includes ___________________________________________________________ */
#ifdef HAVE_CONFIG_H
#include <config.h>
#endif
#include <cmath>
#include "GException.hpp"
#include "GTools.hpp"
#include "GRan.hpp"
#include "GModelSpectralEBL.hpp"
#include "GModelSpectralRegistry.hpp"

/* __ Constants __________________________________________________________ */

/* __ Globals ____________________________________________________________ */
const GModelSpectralEBL      g_spectral_ebl_seed;
const GModelSpectralRegistry g_spectral_ebl_registry(&g_spectral_ebl_seed);

/* __ Method name definitions ____________________________________________ */
#define G_MC        "GModelSpectralEBL::mc(GEnergy&, GEnergy&, GTime&, GRan&)"
#define G_READ                        "GModelSpectralEBL::read(GXmlElement&)"
#define G_WRITE                      "GModelSpectralEBL::write(GXmlElement&)"
```

Make sure that package configuration is available (conditional compiling)

Things that are used within the file

Register the model (will handle detection in XML files)

In case a method throws an exception

.cpp code adaption (2)

```
/*==========================================================================
 =                                                                        =
 =                        Constructors/destructors                        =
 =                                                                        =
 ==========================================================================*/

/***********************************************************************//**
 * @brief Void constructor
 ***************************************************************************/
GModelSpectralEBL::GModelSpectralEBL(void) : GModelSpectral()
{
    // Initialise private members for clean destruction
    init_members();

    // Return
    return;
}


/***********************************************************************//**
 * @brief Copy constructor
 *
 * @param[in] model Spectral EBL model.
 ***************************************************************************/
GModelSpectralEBL::GModelSpectralEBL(const GModelSpectralEBL& model) :
                   GModelSpectral(model)
{
    // Initialise members
    init_members();

    // Copy members
    copy_members(model);

    // Return
    return;
}


/***********************************************************************//**
 * @brief Destructor
 ***************************************************************************/
GModelSpectralEBL::~GModelSpectralEBL(void)
{
    // Free members
    free_members();

    // Return
    return;
}
```

Invoke base class constructor

Initialise all class members

Invoke base class constructor

Initialise all class members

Copy all class members

Free all class members

.cpp code adaption (3)

```
/*==========================================================================
 =                                                                        =
 =                              Operators                                 =
 =                                                                        =
 ==========================================================================*/

/***********************************************************************//**
 * @brief Assignment operator
 *
 * @param[in] model Spectral EBL model.
 * @return Spectral EBL model.
 ***************************************************************************/
GModelSpectralEBL& GModelSpectralEBL::operator=(const GModelSpectralEBL& model)
{
    // Execute only if object is not identical
    if (this != &model) {

        // Copy base class members
        this->GModelSpectral::operator=(model);

        // Free members
        free_members();

        // Initialise members
        init_members();

        // Copy members
        copy_members(model);

    } // endif: object was not identical

    // Return
    return *this;
}


/*==========================================================================
 =                                                                        =
 =                            Public methods                              =
 =                                                                        =
 ==========================================================================*/

/***********************************************************************//**
 * @brief Clear spectral EBL model
 ***************************************************************************/
void GModelSpectralEBL::clear(void)
{
    // Free class members (base and derived classes, derived class first)
    free_members();
    this->GModelSpectral::free_members();

    // Initialise members
    this->GModelSpectral::init_members();
    init_members();

    // Return
    return;
}
```

Don't copy identity

Assign base class members

Free all class members

Initialise all class members

Copy all class members

Free all members, including all base classes (base classes last)

Initialise all members, including all base classes (base classes first)

.cpp code adaption (4)

```
/***************************************************************************//**
 * @brief Initialise class members
 ***************************************************************************/
void GModelSpectralEBL::init_members(void)
{
    // Initialise model type
    m_type = "EBL";

    // Initialise EBL normalisation
    m_norm.clear();
    m_norm.name("Prefactor");
    m_norm.unit("ph/cm2/s/MeV");
    m_norm.scale(1.0);
    m_norm.value(1.0);          // default: 1.0
    m_norm.min(0.0);            // min:     0.0
    m_norm.free();
    m_norm.gradient(0.0);
    m_norm.has_grad(true);

    // Initialise EBL index
    m_index.clear();
    m_index.name("Index");
    m_index.scale(1.0);
    m_index.value(-2.0);        // default: -2.0
    m_index.range(-10.0,+10.0); // range:   [-10,+10]
    m_index.free();
    m_index.gradient(0.0);
    m_index.has_grad(true);

    // Initialise pivot energy
    m_pivot.clear();
    m_pivot.name("PivotEnergy");
    m_pivot.unit("MeV");
    m_pivot.scale(1.0);
    m_pivot.value(100.0);       // default: 100
    m_pivot.fix();
    m_pivot.gradient(0.0);
    m_pivot.has_grad(true);

    // Set parameter pointer(s)
    m_pars.clear();
    m_pars.push_back(&m_norm);
    m_pars.push_back(&m_index);
    m_pars.push_back(&m_pivot);

    // Return
    return;
}
```

The type of your model in the XML file

One block for each parameter

True if parameter has an analytical gradient, false otherwise

Push all parameters into the parameter stack (allows iterating over parameters)

.cpp code adaption (5)

```
/***************************************************************************//**
 * @brief Evaluate function
 *
 * @param[in] srcEng True photon energy.
 * @param[in] srcTime True photon arrival time.
 * @param[in] gradients Compute gradients?
 * @return Spectral EBL model value (ph/cm2/s/MeV).
 *
 * Evaluates
 *
 * \f[
 *    S_{\rm E}(E | t) = HERE YOUR NEW FANCY FORMULA IS NEEDED
 * \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.
 *
 * If the @p gradients flag is true the method will also compute the
 * partial derivatives of the model with respect to the parameters using ...
 ***************************************************************************/
double GModelSpectralEBL::eval(const GEnergy& srcEng,
                               const GTime&   srcTime,
                               const bool&    gradients) const
{
    // Compute function value
    // DUMMY CODE. HERE YOUR NEW FANCY CODE IS NEEDED !!!
    double value  = 0.0;

    // Optionally compute gradients
    if (gradients) {

        // Compute partial derivatives of the parameter values
        // DUMMY CODE. HERE YOUR NEW FANCY CODE IS NEEDED !!!
        double g_norm  = 0.0;
        double g_index = 0.0;
        double g_pivot = 0.0;

        // Set gradients
        m_norm.factor_gradient(g_norm);
        m_index.factor_gradient(g_index);
        m_pivot.factor_gradient(g_pivot);

    } // endif: gradient computation was requested

    // Compile option: Check for NaN/Inf
    #if defined(G_NAN_CHECK)
    if (gammalib::is_notanumber(value) || gammalib::is_infinite(value)) {
        std::cout << "*** ERROR: GModelSpectralEBL::eval";
        std::cout << "(srcEng=" << srcEng << ", srcTime=" << srcTime << "):";
        std::cout << " NaN/Inf encountered (value=" << value << ")" << std::endl;
    }
    #endif

    // Return
    return value;
}
```

The only function you really needed to adapt to the model (and possibly optimise)

Formula in LaTeX

Here the real magic happens

And here even more magic happens. Hopefully you can work out the analytical parameter gradients (with respect to the parameter value, i.e. parameter gradients times parameter scaling)

Compile option to catch NaNs / Infs

.i code adaption (1)

.i files are interface files processed by SWIG to create the gammalib Python module. SWIG generates a xxx_wrap.cpp and a xxx.py file for each module xxx. The GModelSpactralEBL.i file is part of the "model" module.

```
/***************************************************************************
 *             GModelSpectralEBL.i - Spectral EBL model class              *
 * ----------------------------------------------------------------------- *
 *  copyright (C) 2016 by Joe Public                                       *
 * ----------------------------------------------------------------------- *
 *                                                                         *
 *  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 GModelSpectralEBL.i
 * @brief EBL spectral model class interface definition
 * @author Joe Public
 */
%{
/* Put headers and other declarations here that are needed for compilation */
#include "GModelSpectralEBL.hpp"
%}
```

Special SWIG directive

.i code adaption (2)

```
/***********************************************************************//**
 * @class GModelSpectralEBL
 *
 * @brief EBL spectral model class
 ***************************************************************************/
class GModelSpectralEBL : public GModelSpectral {
public:
    // Constructors and destructors
    GModelSpectralEBL(void);
    GModelSpectralEBL(const double&  prefactor,
                      const double&  index,
                      const GEnergy& pivot);
    explicit GModelSpectralEBL(const GXmlElement& xml);
    GModelSpectralEBL(const GModelSpectralEBL& model);
    virtual ~GModelSpectralEBL(void);

    // Implemented pure virtual methods
    virtual void               clear(void);
    virtual GModelSpectralEBL* clone(void) const;
    virtual std::string        classname(void) const;
    virtual std::string        type(void) const;
    virtual double             eval(const GEnergy& srcEng,
                                    const GTime&   srcTime = GTime(),
                                    const bool&    gradients = false) const;
    virtual double             flux(const GEnergy& emin,
                                    const GEnergy& emax) const;
    virtual double             eflux(const GEnergy& emin,
                                     const GEnergy& emax) const;
    virtual GEnergy            mc(const GEnergy& emin,
                                  const GEnergy& emax,
                                  const GTime&   time,
                                  GRan&          ran) const;
    virtual void               read(const GXmlElement& xml);
    virtual void               write(GXmlElement& xml) const;

    // Other methods
    double  prefactor(void) const;
    void    prefactor(const double& prefactor);
};


/***********************************************************************//**
 * @brief GModelSpectralEBL class extension
 ***************************************************************************/
%extend GModelSpectralEBL {
    GModelSpectralEBL copy() {
        return (*self);
    }
    bool __eq__(const GModelSpectralEBL& model) const {
        return ((*self) == model);
    }
};
```

Almost identical to the public class definition in the .hpp file

No = operator

No print() method

Extensions can make the class more “Pythonic” (also extension written in Python possible)

Add new files to build system

Update the following files and type automake & ./configure

include/Gammalib.hpp

```
/* __ Model handling ________________________
#include "GModelPar.hpp"
#include "GModels.hpp"
#include "GModel.hpp"
#include "GModelRegistry.hpp"
#include "GModelSky.hpp"
#include "GModelData.hpp"
#include "GModelSpatial.hpp"
#include "GModelSpatialRegistry.hpp"
#include "GModelSpatialPointSource.hpp"
#include "GModelSpatialRadial.hpp"
#include "GModelSpatialRadialDisk.hpp"
#include "GModelSpatialRadialGauss.hpp"
#include "GModelSpatialRadialShell.hpp"
#include "GModelSpatialElliptical.hpp"
#include "GModelSpatialEllipticalDisk.hpp"
#include "GModelSpatialEllipticalGauss.hpp"
#include "GModelSpatialDiffuse.hpp"
#include "GModelSpatialDiffuseConst.hpp"
#include "GModelSpatialDiffuseCube.hpp"
#include "GModelSpatialDiffuseMap.hpp"
#include "GModelSpectral.hpp"
#include "GModelSpectralRegistry.hpp"
#include "GModelSpectralBrokenPlaw.hpp"
#include "GModelSpectralConst.hpp"
#include "GModelSpectralExpPlaw.hpp"
#include "GModelSpectralExpInvPlaw.hpp"
#include "GModelSpectralSuperExpPlaw.hpp"
#include "GModelSpectralFunc.hpp"
#include "GModelSpectralGauss.hpp"
#include "GModelSpectralLogParabola.hpp"
#include "GModelSpectralNodes.hpp"
#include "GModelSpectralPlaw.hpp"
#include "GModelSpectralPlawPhotonFlux.hpp"
#include "GModelSpectralPlawEnergyFlux.hpp"
#include "GModelSpectralEBL.hpp"
#include "GModelTemporal.hpp"
#include "GModelTemporalRegistry.hpp"
#include "GModelTemporalConst.hpp"
```

include/Makefile.am

```
GSource.hpp \
GModelPar.hpp \
GModels.hpp \
GModel.hpp \
GModelRegistry.hpp \
GModelSky.hpp \
GModelData.hpp \
GModelSpatial.hpp \
GModelSpatialRegistry.hpp \
GModelSpatialPointSource.hpp \
GModelSpatialRadial.hpp \
GModelSpatialRadialDisk.hpp \
GModelSpatialRadialGauss.hpp \
GModelSpatialRadialShell.hpp \
GModelSpatialElliptical.hpp \
GModelSpatialEllipticalDisk.hpp \
GModelSpatialEllipticalGauss.hpp \
GModelSpatialDiffuse.hpp \
GModelSpatialDiffuseConst.hpp \
GModelSpatialDiffuseCube.hpp \
GModelSpatialDiffuseMap.hpp \
GModelSpectral.hpp \
GModelSpectralRegistry.hpp \
GModelSpectralBrokenPlaw.hpp \
GModelSpectralConst.hpp \
GModelSpectralExpInvPlaw.hpp \
GModelSpectralExpPlaw.hpp \
GModelSpectralSuperExpPlaw.hpp \
GModelSpectralFunc.hpp \
GModelSpectralGauss.hpp \
GModelSpectralLogParabola.hpp \
GModelSpectralNodes.hpp \
GModelSpectralPlaw.hpp \
GModelSpectralPlawPhotonFlux.hpp \
GModelSpectralPlawEnergyFlux.hpp \
GModelSpectralEBL.hpp \
GModelTemporal.hpp \
GModelTemporalRegistry.hpp \
GModelTemporalConst.hpp
```

pyext/gammalib/model.i

```
/* __ Model handling ________________________
%include "GModelPar.i"
%include "GModels.i"
%include "GModel.i"
%include "GModelRegistry.i"
%include "GModelSky.i"
%include "GModelData.i"
%include "GModelSpatial.i"
%include "GModelSpatialRegistry.i"
%include "GModelSpatialPointSource.i"
%include "GModelSpatialRadial.i"
%include "GModelSpatialRadialDisk.i"
%include "GModelSpatialRadialGauss.i"
%include "GModelSpatialRadialShell.i"
%include "GModelSpatialElliptical.i"
%include "GModelSpatialEllipticalDisk.i"
%include "GModelSpatialEllipticalGauss.i"
%include "GModelSpatialDiffuse.i"
%include "GModelSpatialDiffuseConst.i"
%include "GModelSpatialDiffuseCube.i"
%include "GModelSpatialDiffuseMap.i"
%include "GModelSpectral.i"
%include "GModelSpectralRegistry.i"
%include "GModelSpectralBrokenPlaw.i"
%include "GModelSpectralConst.i"
%include "GModelSpectralExpInvPlaw.i"
%include "GModelSpectralExpPlaw.i"
%include "GModelSpectralSuperExpPlaw.i"
%include "GModelSpectralFunc.i"
%include "GModelSpectralGauss.i"
%include "GModelSpectralLogParabola.i"
%include "GModelSpectralNodes.i"
%include "GModelSpectralPlaw.i"
%include "GModelSpectralPlawPhotonFlux.i"
%include "GModelSpectralPlawEnergyFlux.i"
%include "GModelSpectralEBL.i"
%include "GModelTemporal.i"
%include "GModelTemporalRegistry.i"
%include "GModelTemporalConst.i"
```

src/model/Makefile.am

```
# Define sources for this directory
sources = GModelPar.cpp \
          GModels.cpp \
          GModel.cpp \
          GModelRegistry.cpp \
          GModelSky.cpp \
          GModelData.cpp \
          GModelSpatial.cpp \
          GModelSpatialRegistry.cpp \
          GModelSpatialPointSource.cpp \
          GModelSpatialRadial.cpp \
          GModelSpatialRadialDisk.cpp \
          GModelSpatialRadialGauss.cpp \
          GModelSpatialRadialShell.cpp \
          GModelSpatialElliptical.cpp \
          GModelSpatialEllipticalDisk.cpp \
          GModelSpatialEllipticalGauss.cpp \
          GModelSpatialDiffuse.cpp \
          GModelSpatialDiffuseConst.cpp \
          GModelSpatialDiffuseCube.cpp \
          GModelSpatialDiffuseMap.cpp \
          GModelSpectral.cpp \
          GModelSpectralRegistry.cpp \
          GModelSpectralBrokenPlaw.cpp \
          GModelSpectralConst.cpp \
          GModelSpectralExpInvPlaw.cpp \
          GModelSpectralExpPlaw.cpp \
          GModelSpectralSuperExpPlaw.cpp \
          GModelSpectralFunc.cpp \
          GModelSpectralGauss.cpp \
          GModelSpectralLogParabola.cpp \
          GModelSpectralNodes.cpp \
          GModelSpectralPlaw.cpp \
          GModelSpectralPlawPhotonFlux.cpp \
          GModelSpectralPlawEnergyFlux.cpp \
          GModelSpectralEBL.cpp \
          GModelTemporal.cpp \
          GModelTemporalRegistry.cpp \
          GModelTemporalConst.cpp \
          GException_model.cpp
```

Testing in GammaLib and ctools

GammaLib includes classes for implementing tests

- **GTestCase:** implements one test
- **GTestSuite:** implements a collection of tests (for example all tests of the methods of a given class)
- **GTestSuites:** implements a container of test suites (for example all tests for a given module, all ctools tests, etc.)

Setting up a test suite in C++

```
/***************************************************************************//**
 * @brief Set parameters and tests
 ***************************************************************************/
void TestGSupport::set(void){

    // Set test name
    name("GSupport");

    // Append tests
    append(static_cast<pfunction>(&TestGSupport::test_tools), "Test GTools");

    // Return
    return;
}


/***************************************************************************//**
 * @brief Test GTools functions
 ***************************************************************************/
void TestGSupport::test_tools(void)
{
    // Illustration of test cases
    test_assert(gammalib::strip_whitespace("  World  ") == "World");
    test_value(gammalib::plaw_energy_flux(2.0, 3.0, 2.5, -1.0), 2.5);
    test_try("Void constructor");
    try {
        GNodeArray array;
        test_try_success();
    }
    catch (std::exception &e) {
        test_try_failure(e);
    }

    // Return
    return;
}


/***************************************************************************//**
 * @brief Main test entry point
 ***************************************************************************/
int main(void)
{
    // Allocate test suite container
    GTestSuites testsuites("Support module");

    // Create and append test suite
    TestGSupport test;
    testsuites.append(test);

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

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

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

Set all test methods

One test method (for example one per class)

Test assertions, values and exceptions

Allocate and append test suite to container (TestGSupport class derives from GTestSuite base class)

Run tests

Write test report into XML file (JUnit format)

Setting up a test suite in Python

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

        # Return
        return

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

        # Append tests
        self.append(self.test_node_array, 'Test GNodeArray')
        self.append(self.test_url_file,   'Test GUrlFile')
        self.append(self.test_url_string, 'Test GUrlString')
        self.append(self.test_filename,   'Test GFilename')
        self.append(self.test_csv,        'Test GCsv')

        # Return
        return

    # Test GNodeArray class
    def test_node_array(self):
        """
        Test GNodeArray class.
        """
        # Set-up vector and data array. Test all vector elements.
        vector = gammalib.GVector(20)
        data   = gammalib.GVector(20)
        for i in range(20):
            vector[i] = 10.0 + i * 5.0
            data[i]   = math.sin(0.15 * (vector[i] - 10.0))
            self.test_value(data[i], math.sin(0.15 * i * 5.0))

        # Set-up node array
        array = gammalib.GNodeArray()
        array.nodes(vector)
```

Python test class derives from GPythonTestSuite

Set all test methods

Test method

Test case