10th ctools coding sprint

1 – 5 February 2021

Learning ctools and GammaLib development in one hour

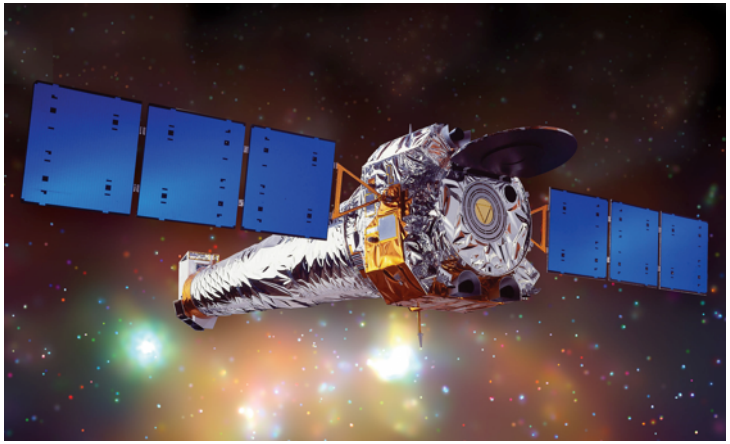
What you should know

- 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 https://www.aanda.org/articles/aa/pdf/2016/09/aa28822-16.pdf)
- Our H.E.S.S. paper (more tools & scripts)
  (see https://www.aanda.org/articles/aa/pdf/2019/12/aa36010-19.pdf)
- Our coding conventions
  (see http://cta.irap.omp.eu/gammalib/develop/coding/index.html and http://cta.irap.omp.eu/ctools/develop/coding/index.html)

What you will learn in the next hour

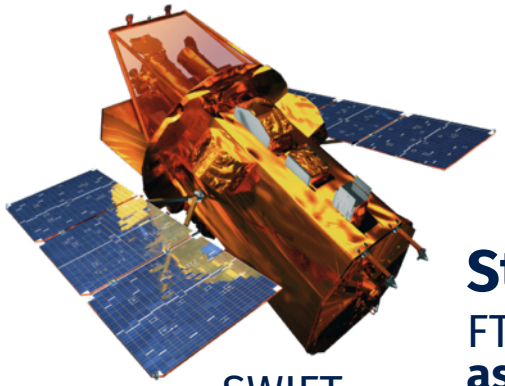
- What is the idea behind ctools and GammaLib
- Why there are two packages
- What is inside GammaLib and ctools
- How to use GammaLib and ctools
- What rules to follow during the code development
- Tutorials on how you can
  - add a new ctool to ctools
  - add a new cscript to ctools

Where do the ctools come from?

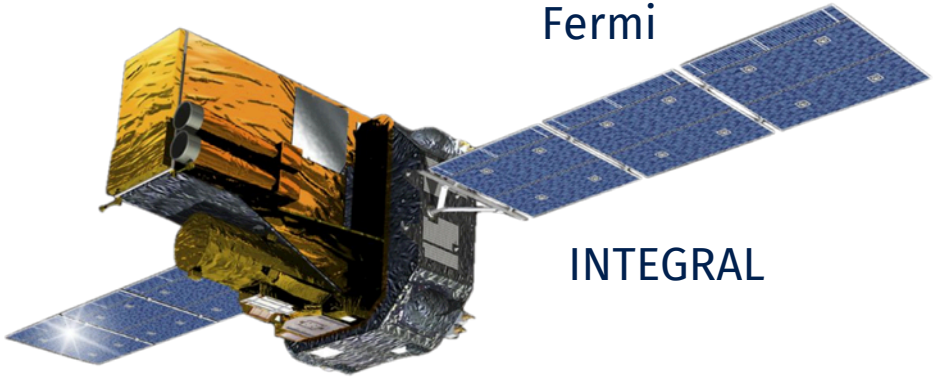


Chandra

Fermi



SWIFT

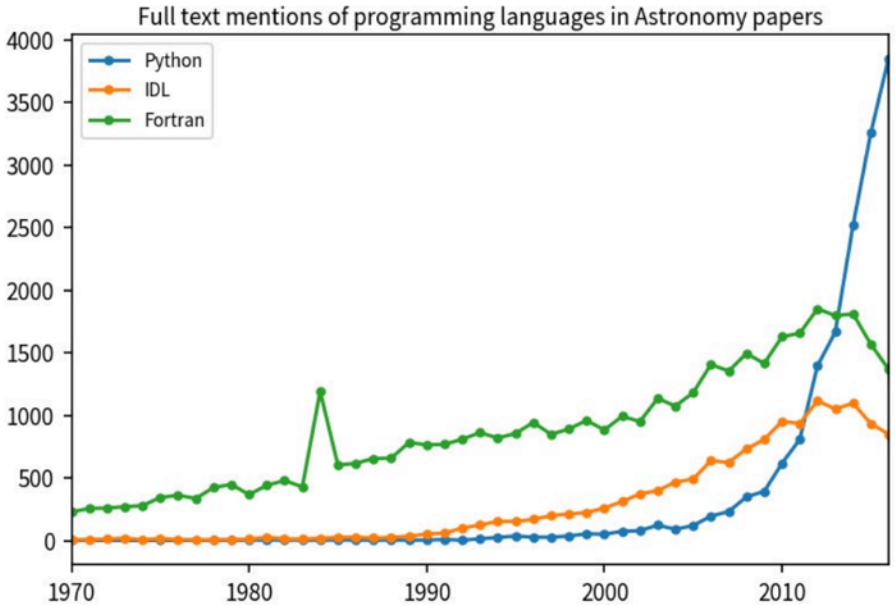


INTEGRAL

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?



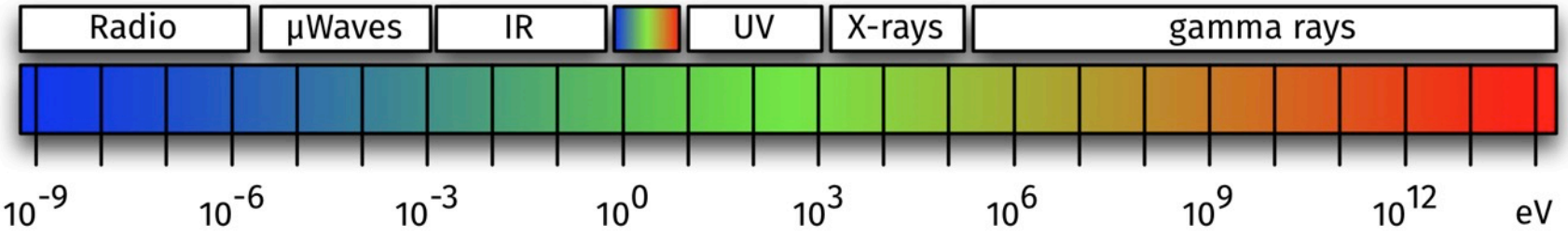
Credits: Thomas Robitaille

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

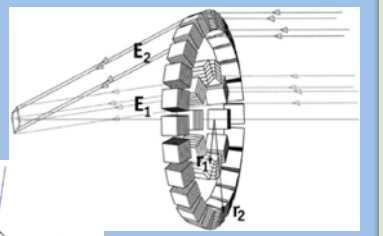
- Fermi/LAT Science Tools proven successful
- Basically **no learning** curve for **Fermi/LAT users**
- **Low learning curve** for users of other **HE observatories**
- But admittedly **some learning curve** for people from the **VHE community**

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

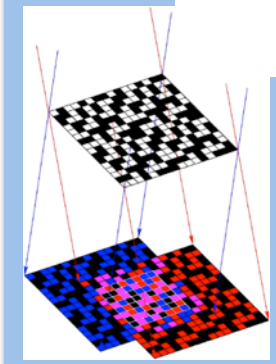
Observing gamma-rays



photoelectric effect

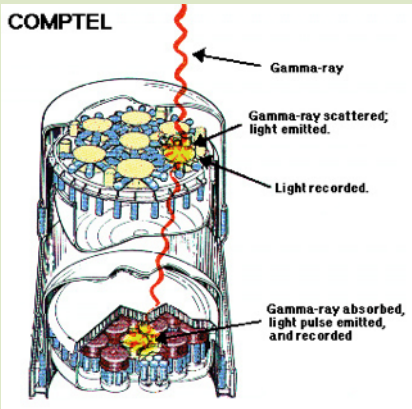


lenses



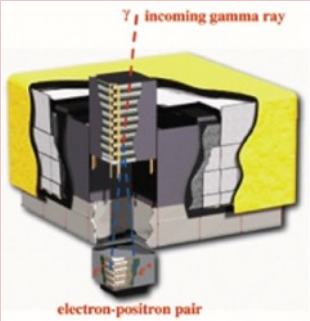
coded masks

Compton scattering



Compton telescopes

pair creation



pair convertors

Cherenkov telescopes

Particle detectors

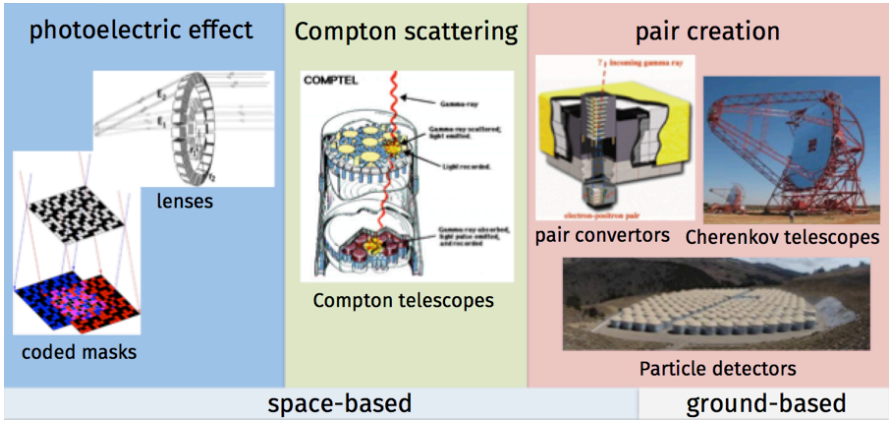
space-based | ground-based

Why developing GammaLib & ctools?

All gamma-ray telescopes measure individual photons as events

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

- FITS files
- likelihood fitting
- modular design



It should be possible to **analyse data from all gamma-ray telescopes with a single software framework**

⇒ develop `GammaLib` as **a universal toolbox** for astronomical gamma-ray data analysis
⇒ propose `ctools` as a `GammaLib` based **data analysis framework** for CTA

Why two packages?

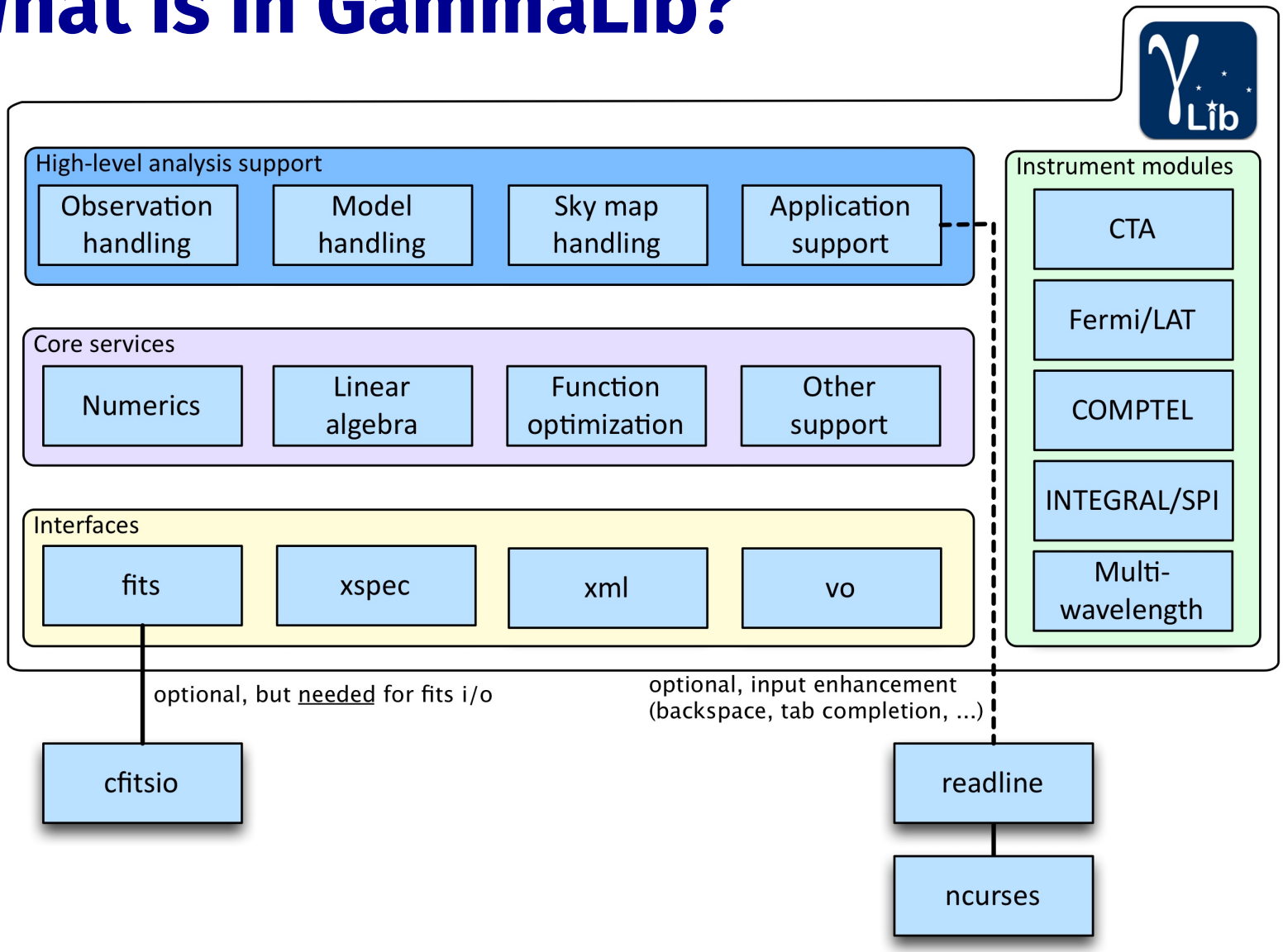
ctools is a IACT/CTA frontend to GammaLib

- Analysis bricks
  (e.g. “simulate events”, “fit model”, “create spectrum”)
- CTA/IACT specific
- Also useful for Fermi/LAT, INTEGRAL/SPI, COMPTEL data (joint) analysis
- Written in Python and C++

- Code bricks
  (e.g. “read FITS file”, “invert matrix”, “optimise function”)
- Largely independent of instrument
- Mostly written in C++

Code development during the sprint will happen on ctools and GammaLib level

What is in GammaLib?



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

```python
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.

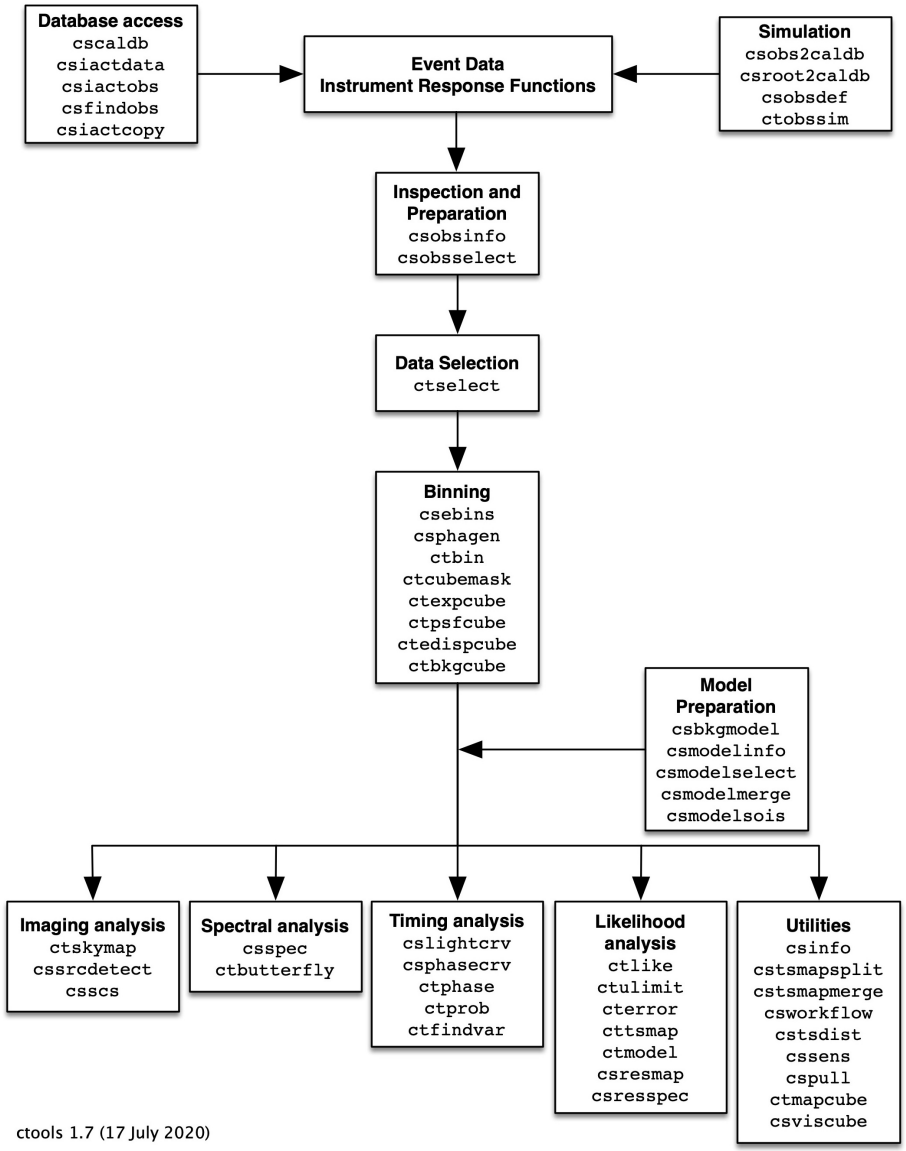
Some GammaLib resources

- GammaLib web site

  (see http://cta.irap.omp.eu/gammalib-devel/)
- Doxygen source code documentation
  (see http://cta.irap.omp.eu/gammalib-devel/doxygen/index.html)
- GammaLib Redmine
  (see https://cta-redmine.irap.omp.eu/projects/gammalib/wiki)

Current developer version: 2.0.0.dev

What is in ctools?



ctools 1.7 (17 July 2020)

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

```python
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
```

Some ctools resources

- ctools web site

  (see http://cta.irap.omp.eu/ctools-devel/)
- Doxygen source code documentation
  (see http://cta.irap.omp.eu/ctools-devel/doxygen/index.html)
- ctools Redmine
  (see https://cta-redmine.irap.omp.eu/projects/ctools/wiki)

Current developer version: 2.0.0.dev

ctools and GammaLib paradigms

- The User is the 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

Some rules to follow

- Make sure you read and follow the coding conventions
  - C++98 ISO (some servers still don't support C++11)
  - Python 2.3 – 3.8 (Python 2.3 is "historical", but why break a working system?)
  - 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)

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
```

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

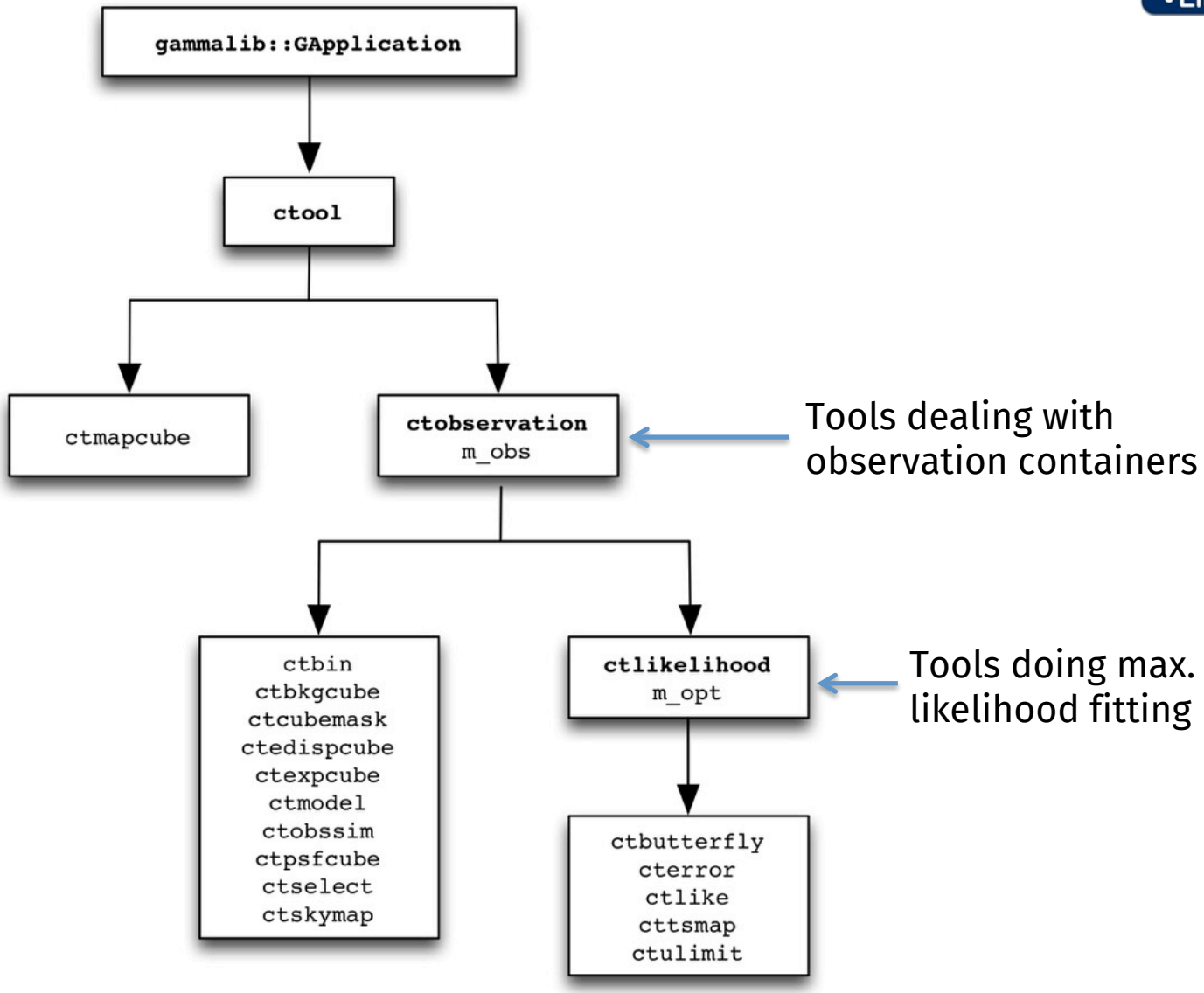
ctools code generator

```
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;
}
```

Main method

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

Here the real stuff happens

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

    log_header1(TERSE, "Save something");

    // TODO: Your code goes here

    // Return
    return;
}
```

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 out

``(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!