ctools - Action #1360

Extend ctools base class for observation setup

11/07/2014 04:46 PM - Mayer Michael

Status:	Closed	**Start date:**	11/07/2014
Priority:	Normal	**Due date:**	
Assigned To:	Mayer Michael	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	1.0.0		

Description

All ctools currently have in common that they contain a GObservations object as protected member. This object can either be passed in the constructor or setup by the user via an XML file or individual parameters.
If this is a common functionality all ctools should share, we could make GObservations m_obs part of the ctools-base class. The base class could further host functions to setup the observation like ctools::setup_obs(). Implementing this scheme would allow for a uniform observation setup throughout the tools.

History

#1 - 11/07/2014 05:31 PM - Knödlseder Jürgen

I though about that indeed, but I was not sure whether all ctools would ultimately need an observation container. You may argue of course that it's not a lot of overhead if every ctool would have one by default. But somehow I felt that we may need a better understanding of the additional ctool needs before implementing such a change.

#2 - 11/07/2014 05:45 PM - Mayer Michael

I see your point. Maybe, as an intermediate step, we could add another base class ctool_obs (or similar), which inherits from ctool and has an observation container stored. In my opinion it would largely simplify every get_parameters() function, if the observation setup was handled in one function. Moreover, the ctools would look more homogenous if they query for the same parameters. This would also feed the discussion about the parameter names you initiated in the Wiki, right?

#3 - 11/07/2014 10:15 PM - Knödlseder Jürgen

I was going over the actual ctools to see how each tool handles an observation container. Some tools do the same things, but some tools do things differently:

- in ctcubemask::get_parameters() and in ctselect::get_parameters() the code sets a member m_use_xml
- in ctlike::get_parameters() an addition stat parameter is read
- ctmodel::get_parameters() follows a different logic in that it also allows the construction of an observation container from scratch that is later on handled in ctmodel::setup_obs().
- in ctobssim::get_parameters() there is for the moment an on-the-fly observation creation, however with requested parameters different from ctmodel::get_parameters() (the difference is that one needs a binned the other a unbinned observation)
- ctskymap::get_parameters() does not deal with XML files, but this was anyways planned to be changed

So probably on-the-fly observation creation has to be handled differently, although one could imagine a method that checks for the presence of some parameters to decide whether on-the-fly creation should be supported or not, and whether on-the-fly creation should be creating a binned or an unbinned observation. Alternatively, 3 methods could be implemented, and the appropriate method could be used in the ctool. For the additional parameters to be read, there are certainly workarounds.

I'm still struggling with the idea of having another intermediate base class. Maybe it's just that I don't like the ctool_obs name (I guess the underscore, which is not very GammaLib/ctools style as there are normally no underscores in class names). Maybe semantically cobstool would be better, as it would be a "Cta OBServation TOOL", hence a tool dealing with CTA observations. This derived class could then look like this:

```
public:
   // Constructors and destructors
   cobstool(void);
   explicit cobstool(const GObservations& obs);
   cobstool(const std::string& name, const std::string& version);
   cobstool(const std::string& name, const std::string& version,
         int argc, char* argv[]);
```

```
    cobstool(const cobstool& app);
    virtual ~cobstool(void);

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

    // Pure virtual methods
    virtual void clear(void) = 0;
    virtual void run(void) = 0;
    virtual void save(void) = 0;

    // Public methods
    virtual const GObservations& obs(void) const;

protected:
    // Protected methods
    void init_members(void);
    void copy_members(const cobstool& app);
    void free_members(void);

    // Protected members
    GObservations           m_obs;       //!< Observation container
};
```

#4 - 11/10/2014 10:01 AM - Mayer Michael

The structure and name sounds good! I would suggest to add a protected method get_obs(), similar to set_ebounds() in ctool-base class. This method can then be called from the get_parameters()-functions of the individual tool. The function could look the following (basically copied from ctlike):

```
GObservations & cobstool::set_obs() const
{
    if (m_obs.size() == 0) {
        // Allocate CTA observation
        GCTAObservation obs;

        // Get event file name
        std::string filename = (*this)["infile"].filename();

        // Try first to open as FITS file
        try {

            // Load data
            obs.load(filename);

            // Set response
```

```
            set_obs_response(&obs);

            // Append observation to container
            m_obs.append(obs);

        }

        // ... otherwise try to open as XML file
        catch (GException::fits_open_error &e) {

            // Load observations from XML file
            m_obs.load(filename);

            // For all observations that have no response, set the response
            // from the task parameters
            set_response(m_obs);

        } // endcatch: file was an XML file

    } // endif: there was no observation in the container

}
```

We might also think (e.g. for ctmodel, ctbin and others) to split the function into cobstool::set_unbinned_obs() and cobstool::set_binned_obs(). For ctobssim, we could only call this function only if the parameter inobs is given (see #1359).

#5 - 11/12/2014 12:20 AM - Knödlseder Jürgen

Agree.

#6 - 11/12/2014 04:54 PM - Mayer Michael

Would it also be another possibility to just extend the ctools-base class by the function:

GObservations& set_obs();

I mean we could also do this completely analoguous to ctool::get_ebounds(). Then we would not need an intermediate cobstool-class. In each respective tool we could build the observations in get_parameters() using:

m_obs = set_obs();

What do you think?

#7 - 11/19/2014 11:21 AM - Knödlseder Jürgen

Indeed, this is maybe a better way to go. So keep things simple, having just one base class, but add support methods that will reduce the lines of code of each tool.

#8 - 11/20/2014 03:09 PM - Mayer Michael

- Assigned To set to Mayer Michael

- % Done changed from 0 to 40

I started working on the code. While adapting the get_parameters()-function of each tool, I was wondering about the purpose of ctskymap. To me, it seems like a very special case of ctbin (just using one energy bin). Am I right?

#9 - 11/20/2014 05:09 PM - Knödlseder Jürgen

Indeed, for the moment ctskymap is basically a special case of ctbin. The goal was to implement some classical imaging algorithms into the tool in the long run.

#10 - 11/21/2014 04:02 PM - Mayer Michael

- Status changed from New to Feedback

- % Done changed from 40 to 70

I worked further on the ctools-base class and successively adapted the individual ctools-classes. The changes I made are available on branch *1360-extend-ctools-base-class*. I only required minor changes on the gammalib-level. I've added a new constructor to GCTAExposure and GCTAMeanPsf. The corresponding branch on gammalib is *1360-extend-ctools-base-class-gammalib*.
Warning: I changed a lot of the code. Every get_parameters()-function has been adapted to use methods which were implemented in the base class. In addition, I changed the parameter names following the proposal on https://cta-redmine.irap.omp.eu/boards/14/topics/209

The base class was enhanced by the following methods:

- GObservations get_obs(bool get_response): It builds the observation container depending on user parameters. Currently there are four possibilities:
  - pass an observation container on the tool constructor
  - provide an XML observation definition file as parameter "inobs"
  - provide an FITS observation file (event list or counts cube) as parameter "inobs"
  - use inobs="NONE", inducing a query for parameters to build an observation from scratch
- GCTAEventCube get_cube(): builds a cta event cube object from user parameters. In case a GObservations-object is passed, this function uses the pointing position instead of xref and yref values.
- GSkyDir get_pointing(const Observations& obs). Is called in the above function to find the pointing position from an observation container. In case several observations are in the container, a simple averaging is applied to find the mean pointing
- GSkymap get_map(): Similar as the get_cube() - function. However without querying for the GEbounds.
- GCTAEventCube set_from_cntmap(const std::string filename);: Loads an event cube definition from file.

Each tool has the "freedom" to call these functions as required. By outsourcing some of these functions, I managed to get rid of many protected members from the individual tools.
The current version is working when using the script test_ctools.sh. However, make check fails due to an import error in the python test suite (which I don't quite undertstand):

```
Test cspull:
Traceback (most recent call last):
  File "/Users/mimayer/Software/ctools/scripts/cspull.py", line 22, in <module>
    import ctools
  File "/Users/mimayer/Software/ctools/pyext/ctools/__init__.py", line 3, in <module>
    import obsutils
  File "/Users/mimayer/Software/ctools/pyext/ctools/obsutils.py", line 22, in <module>
    import ctools
  File "/Users/mimayer/Software/ctools/pyext/ctools/ctools.py", line 26, in <module>
    _ctools = swig_import_helper()
  File "/Users/mimayer/Software/ctools/pyext/ctools/ctools.py", line 18, in swig_import_helper
    import _ctools
ImportError: No module named _ctools
```

Running test/test_python.py individually works just fine. In test/test_csscripts.sh, I get a segmentation fault in cstsdist.py, which occurs when return a fit object from obsutils. I still do not understand this.

I hope I did not break anything by changing so much of the code. However, the parameter interface seems now more homogenous throughout the tools.

What do you think?

#11 - 01/12/2015 11:08 AM - Mayer Michael

Any updates on this issue? Did you have time to look into the changes yet?

#12 - 01/12/2015 01:51 PM - Knödlseder Jürgen

This one escaped my attention. Sorry for that. I'll look into the issue as soon as possible (next days).

#13 - 01/14/2015 01:25 AM - Knödlseder Jürgen

- % Done changed from 70 to 80

I went through the code. Here some comments in the order I checked the code:

I recognised that you added an event cube constructor to GCTAExposure that copies exposure information from an event cube. I added a statement that sets all pixel values to zero as a constructor should provide an object in a well defined clean state. For GCTAMeanPsf I change the constructor in the same way by using a newly coded GSkymap::nmaps method that allows changing the number of maps in a sky map object. I pushed the modification in the 1360-extend-ctools-base-class-gammalib branch.

I could not reproduce your cspull import error. make check works on my side without any problems. Have you make a make clean before building? Maybe you have still some old code around?

I see some stuff in cspull that looks like debugging code. Same for cstsdist. I did remove this stuff. Unit tests still work.

Concerning the ctool::set_from_cntmap method I was wondering whether the WCS check was needed? It would be good if the code could - at least in principle - apply to any kind of projections. I see no reason why this should not work. In any case, I added a filename constructor to GCTAEventCube which makes this method basically obsolete.

I renamed ctool::get_pointing to ctool::get_mean_pointing to emphasize that the method does some averaging. However, the method needs to be improved to take into account wrap arounds in Right Ascension and to handle also coordinates near the celestial poles.

I changed the code in ctool::get_cube to reuse the ctool::get_map methods.

Finally, I still see some code repetition among the different tools (for example the get_cube and get_map methods that need two instances that are always used the same way). I modified the ctool base class so that the usepnt parameter is handled at that level, hence only a single instance of the method is used. I furthermore renamed the methods to create_cube and create_map as the get term could be mis-interpreted as loading a cube from a file (get a cube from somewhere). I propose to reserve the get methods for methods that get for example some data from a file.

With the same logic I renamed:

- get_ebounds to create_ebounds
- setup_obs to create_cta_obs

I furthermore added a require_inobs method to throw an exception in case that the inobs parameter is NONE or empty. This further reduces redundant code in many of the tools.

I pushed the code into 1360-extend-ctools-base-class.

I have not yet done tests of the tools beyond the unit tests. In particular, I have not yet checked the results of the tools (i.e. if they still operate as expected and produce the results as expected). As this is a major change, we should make sure that no problems occur before we merge the code into the trunk.

#14 - 01/15/2015 10:39 AM - Mayer Michael

- % Done changed from 80 to 90

Thanks for the extensive feedback.

> I recognised that you added an event cube constructor to GCTAExposure that copies exposure information from an event cube. I added a statement that sets all pixel values to zero as a constructor should provide an object in a well defined clean state. For GCTAMeanPsf I change the constructor in the same way by using a newly coded GSkymap::nmaps method that allows changing the number of maps in a sky map object. I pushed the modification in the 1360-extend-ctools-base-class-gammalib branch.

Perfect, of course the constructor should give a clean object. Thanks for catching this.

> I could not reproduce your cspull import error. make check works on my side without any problems. Have you make a make clean before building? Maybe you have still some old code around?

A clean checkout of gammalib and ctools solved this problem. I guess there were some version confusions.

> I see some stuff in cspull that looks like debugging code. Same for cstsdist. I did remove this stuff. Unit tests still work.

Probably same as above, due to a commit error. Thanks for cleaning this up.

> Concerning the ctool::set_from_cntmap method I was wondering whether the WCS check was needed? It would be good if the code could - at least in principle - apply to any kind of projections. I see no reason why this should not work. In any case, I added a filename constructor to GCTAEventCube which makes this method basically obsolete.

Agreed, this is a much better approach.

> I renamed ctool::get_pointing to ctool::get_mean_pointing to emphasize that the method does some averaging. However, the method needs to be improved to take into account wrap arounds in Right Ascension and to handle also coordinates near the celestial poles.

This is very true. One has to think about a proper way how to average the given coordinates. Ultimately, we might need this functionality anyway if we go to pointing tables. The potential GCTAPointings class (#1176) could return its average direction.

> I changed the code in ctool::get_cube to reuse the ctool::get_map methods.

great!

> Finally, I still see some code repetition among the different tools (for example the get_cube and get_map methods that need two instances that are always used the same way). I modified the ctool base class so that the usepnt parameter is handled at that level, hence only a single instance of the method is used. I furthermore renamed the methods to create_cube and create_map as the get term could be mis-interpreted as loading a cube from a file (get a cube from somewhere). I propose to reserve the get methods for methods that get for example some data from a file.

I also agree to the name changes, which make the thing more clear.

> I have not yet done tests of the tools beyond the unit tests. In particular, I have not yet checked the results of the tools (i.e. if they still operate as expected and produce the results as expected). As this is a major change, we should make sure that no problems occur before we merge the code into the trunk.

I went over the code and did not find any obvious problems. The test cases work for me as well. I will conduct some more sophisticated analyses to see if we encounter problems somewhere. I will provide further feedback very soon.

Another thing:
The cscripts are currently not affected by this change. However, it would be great to be able to use the same functionality from these python scripts to setup the observations, maps, etc.. I did not find a way to call protected functions from the ctool-base class via python. Would it therefore make sense to make all the new methods public and add them to the ctool.i-file. Or do you have a better idea?

#15 - 01/16/2015 09:56 AM - Knödlseder Jürgen

Mayer Michael wrote:

> Another thing:
> The cscripts are currently not affected by this change. However, it would be great to be able to use the same functionality from these python scripts to setup the observations, maps, etc.. I did not find a way to call protected functions from the ctool-base class via python. Would it therefore make sense to make all the new methods public and add them to the ctool.i-file. Or do you have a better idea?

Fully agree that we should also modify the Python interface.

It is annoying that SWIG cannot handle the protected members.

I poked a little around and found a dirty trick, that however should not be used (no guarantee that it will always work):

```
%{
/* Put headers and other declarations here that are needed for compilation */
#define protected public
#include "ctool.hpp"
%}
```

However adding the following to the ctool.hpp file

```
#ifdef SWIG
public:
#else
protected:
#endif
```

and the following to the ctool.i file

```
%{
/* Put headers and other declarations here that are needed for compilation */
#define SWIG
#include "ctool.hpp"
%}
```

is clean and seems to work.

I pushed this modification into the 1360-extend-ctools-base-class, you can now use the protected methods in the cscripts (where they will be public, however).

#16 - 01/19/2015 01:43 PM - Mayer Michael

- % Done changed from 90 to 100

ok great. Thanks for the quick solution. I created an issue to adapt the cscripts accordingly (#1408).

In the meantime, I checked all individual tools carefully. In general, everything went fine. Binned and unbinned results looked reasonable. However, I did of course not check every possibility of parameter combinations.

I discovered two minor issues, which should be fixed before the merge:

1. the .par file from ctpsfcube has a copy&paste error from ctexpcube (it just asks for output exposure cube file)
2. in ctbkgcube::run() (line 266), there is a check whether the background model is valid. However, this function checks for validity with the identifier "", which always gives me false in case I provide a proper id. I therefore changed the code that it only checks for the possible instrument names ("CTA","HESS","MAGIC" or "VERITAS").

I pushed these changes to the working branch. From my side there is green light to proceed to merge the changes. Are there any additional tests you want to be performed first?

#17 - 01/20/2015 09:11 AM - Mayer Michael

- Status changed from Feedback to Pull request

#18 - 01/30/2015 09:32 PM - Knödlseder Jürgen

Mayer Michael wrote:

> ok great. Thanks for the quick solution. I created an issue to adapt the cscripts accordingly (#1408).
>
> In the meantime, I checked all individual tools carefully. In general, everything went fine. Binned and unbinned results looked reasonable. However, I did of course not check every possibility of parameter combinations.

Thanks for having made these checks

> I discovered two minor issues, which should be fixed before the merge:
>
> 1. the .par file from ctpsfcube has a copy&paste error from ctexpcube (it just asks for output exposure cube file)

Ok

> 1. in ctbkgcube::run() (line 266), there is a check whether the background model is valid. However, this function checks for validity with the identifier "", which always gives me false in case I provide a proper id. I therefore changed the code that it only checks for the possible instrument names ("CTA","HESS","MAGIC" or "VERITAS").

This one could lead to problems when a model applies to several instruments, as the instruments() method then returns, e.g., "CTA,HESS". I thus reverted the change but made a modification to GModel::is_valid() that ignores a check if a blank string is provided.

> I pushed these changes to the working branch. From my side there is green light to proceed to merge the changes. Are there any additional tests you want to be performed first?

No, I think this is fine now. I'm going to merge the code in.

#19 - 01/30/2015 09:52 PM - Knödlseder Jürgen

- Status changed from Pull request to Closed

- Remaining (hours) set to 0.0

Merged into devel.

#20 - 02/02/2015 11:38 AM - Mayer Michael

- Estimated time set to 0.00

> This one could lead to problems when a model applies to several instruments, as the instruments() method then returns, e.g., "CTA,HESS".

Very true. Haven't thought of this possibility.

> I thus reverted the change but made a modification to GModel::is_valid() that ignores a check if a blank string is provided.

Great.

> Merged into devel.

Thanks.