GammaLib - Feature #1598

Loading of CTA event list via additional HDU name

12/10/2015 06:34 PM - Mayer Michael

Status:	Closed	**Start date:**	01/18/2016
Priority:	Normal	**Due date:**	
Assigned To:	Mayer Michael	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	1.1.0		

Description

Following #1597, we decided that loading of CTA event lists should be possible by specifying the name of the FITS extension in addition to the filename. I think we have to split the loading of GTIs from the loading of the event list.
We should probably modify the following functions:

- GCTAObservation::read(GXmlElement&): Look for "extname" attribute in xml file and store m_extname as protected member.
- GCTAObservation::read(GXmlElement&): Look for parameter "GTIs" or "GoodTimeIntervals" an store protected strings m_gti_filename, m_gti_extname.
- GEvents::load() (in all derived classes) to add std::string& extname. Use "EVENTS" as default.
- GCTAEventList::read() to use the correct HDU.
- GCTAEventList add function load_gti(std::string filename, std::string extname)@
- GCTAObservation::load(std::string filename, std::string extname) to call m_events->load_gti(m_gti_filename, m_gti_extname)

History

#1 - 12/10/2015 06:36 PM - Mayer Michael

- Description updated

#2 - 12/17/2015 05:37 PM - Mayer Michael

I have worked a bit on this feature to allow the reading of event lists with the scheme

```
GCTAEventList::load(std::string = "filename.fits[hduname]")
```

However, this is not straight forward since we would have to adapt GCTAEventList::read(GFits& fits) to GCTAEventList::read(GFitsTable& table). This is not possible since GCTAEventList is embedded in the structure of GEvents.
My idea now is to create GCTAObservations::load_gti() which would call GCTAEventList::read_gti(). In the current scheme GTIs are handled inside event lists. Thus we need to adapt GCTAObservation::read(GXmlElement& xml) to also look for the parameter attribute "GoodTimeIntervals" and reads them from file if given.

It is however, not clear to me how the writing would work:

- in GCTAObservation::save() we can pass one filename
- should we create GCTAObservation::save_gti(std::string filename)?
- We could have a flag that steers whether GTI was given in the same file as events or a different one and pass that to the writing function

We might think about this more in detail. I am off for vacations now and will continue after Christmas.

#3 - 01/11/2016 10:15 PM - Knödlseder Jürgen

- Release set to gammalib-1.1.0

#4 - 01/12/2016 02:00 PM - Mayer Michael

- Status changed from New to Pull request

- Assigned To set to Mayer Michael

- % Done changed from 0 to 100

While working on this feature, I realised that GFilename requires an addition: selection expressions should be allowed in the filename. Until now "file.fits[EVENTS][ENERGY>0.1]" would have thrown an exception. I therefore modified GFilename::set_filename() to read this scheme and added a protected member m_expression and included some unit tests for this.

Here are the additional changes I have made:

- GCTAObservation::save() now internally uses a GFilename instance.
- GCTAObservation::write() now takes and additional argument const std::string extname = "EVENTS"
- GCTAObservation now has a protected member std::string m_gtifile that can store a separate filename for GTIs.
- GCTAObservation::events() now calls the new function load_gti if m_gtifile is not empty.
- There is the new function GCTAObservation::save_gti(const std::string& filename, const bool& clobber) that saves GTIs to a file. This functions also opens an existing file and appends the GTI extension.
- Added GCTAEventList::load_gti(const std::string& filename) to allow separate loading of GTIs
- Added flag m_has_gti_ext that signals if GTIs were present on file loading. This is used for writing again.
- GCTAEventList::read(const GFits& fits) now recovers the FITS filename and extracts the extension name to be loaded for the events.

All unit tests run smoothly.

Changes to gammalib are available on branch *1598-loading-of-CTA-Eventlist-HDU*.

I had to make a minor change to ctselect on the ctools side to support GTIs in a separate file (also added a test case for this). The changes are in ctools on branch *1598-ctools-loading-CTA-events-HDU*.

#5 - 01/13/2016 04:56 PM - Knödlseder Jürgen

I'm wondering how to handle best the GTIs in that case (I should have reacted on this earlier).

So far I have assumed that the GTI lives in the same file as the events, and hence that you simply need the file name. I think that many missions implement that logic, but I'm also aware that INTEGRAL for example does not follow this logic (for INTEGRAL, the GTIs are in a separate file as they wanted to avoid duplication). I'm not sure that there is a OGIP rule on this.

The problem is of course on how to assure coherence between the events and the GTI. We need this coherence for a correct computation of the exposure, and we want to avoid that the end user can make a mistake. INTEGRAL assures this through a complex indexing scheme (basically storing the locations of all FITS file in another FITS file), which makes it basically impossible to move files. I think that we do not want to follow that path, but leave the flexibility to the user to move the files around and start from a new location.

So my question is: how do you plan to organize the event lists and GTIs? Are unique GTIs always in the same file, event if the extension name can change? Or do you plan to have different GTIs in the same file, or have them in different files? Having things in different files would probable prohibit the moving around of files.

One solution could be to put the GTI location as a header keyword in the event list, so that from a given event HDU one can infer which GTI to use.

This would work perfectly if everything is in the same file (you only would need to provide the GTI extension name). We could for example use the data selection keywords for that, see:

- https://confluence.slac.stanford.edu/display/ST/Data+SubSpace+keywords
- http://hea-www.harvard.edu/~jcm/asc/docs/ps/SDS07.ps (section 1.13)

Note that the DSS is not really a standard, but used by others, so somehow tested.

#6 - 01/13/2016 05:17 PM - Mayer Michael

> So my question is: how do you plan to organize the event lists and GTIs? Are unique GTIs always in the same file, event if the extension name can change? Or do you plan to have different GTIs in the same file, or have them in different files? Having things in different files would probable prohibit the moving around of files.

For the moment we actually think about a scheme using index files:
http://gamma-astro-data-formats.readthedocs.org/en/latest/data_storage/index.html
When it comes to IACT data, the user doesn't need to move around the initial files.
The idea is to start with csiactobs and all data products (e.g. from ctselect, potential ctmktime, ...) will be written according to the user input.
For storage the idea is to have an index file that contains the locations of all files and lists all HDUs on the users machine. Christoph came even up with the idea to put all HESS data into one single FITS file, which probably isn't very practical. Nevertheless, for this we would need to make GTIs accessible by HDU name.

I am not sure if for CTA we will bundle several observations into one file. But when observations become short, we want to omit large number of files since it is quite unhandy.

I think the pull request could handle both scenarios behind the scenes. What do you think?

#7 - 01/13/2016 06:28 PM - Knödlseder Jürgen

I looked a bit at the link but I could not find an answer to whether you plan to separate events from GTI (i.e. put them in different files).

I would argue that separating events and GTI in different files is a bad idea, since it won't be possible later to assure the coherence of the data.

The actual master index file is probably well suited for work within collaborations (or the CTA consortium) where a hugh number of runs is accumulated. We have however to think also about the Guest Observer!

A typical Guest Observer will get a very small set of data. The corresponding Use Case would be that someone obtains 50 hours of observing time on a given source. The Guest Observer will get O(100) event files (if they were split in individual runs) and the corresponding response functions. I propose to pack the events, the GTIs, and the IRF HDUs for a given run into a single file. It can't be simpler.

The user may want to be flexible in moving around these files (he/she could for example want to give a subset of it to another user, etc.). From my experience on INTEGRAL is that in this case the index file becomes pretty unhandy, and you start editing file locations in this index file by hand to arrange things (something that you do not want that a Guest Observer has to do).

This model does not exclude that you put all runs into a single file, differentiating the runs by extension names, but this would still mean that you have one relocatable file that the user can put where he wants (and move around). This model does also not exclude that you use an index file to organise things, but you won't put the existence of such as file as a requirement, hence the system needs to work properly and savely without the index file.

So I guess my only worry is that the system becomes too complex and user unfriendly. We need to keep the system as simple as possible, assuring always automatically that the data are coherent. Expert users may do what they want (at their own risk :-).

This would mean that we implement a way to infer from the event list where the GTI is (which could be done through the DSS keywords). Then we do not rely on specifying this explicitly. Things will be straight forward if we assume that the GTI will always be in the same file as the events are.

#8 - 01/13/2016 08:38 PM - Knödlseder Jürgen

- File SDS07.pdf added

user#3 wrote:

> - http://hea-www.harvard.edu/~jcm/asc/docs/ps/SDS07.ps (section 1.13)

Attached the relevant document attachment:SDS07.pdf
And the FITS standard document attachment:fits_standard30aa.pdf

#9 - 01/13/2016 08:43 PM - Knödlseder Jürgen

- File fits_standard30aa.pdf added

#10 - 01/14/2016 10:08 AM - Mayer Michael

> I looked a bit at the link but I could not find an answer to whether you plan to separate events from GTI (i.e. put them in different files).

I guess the answer is "maybe". I personally think that separating is not a good idea, but it might be something to try for other people. We could support this case.

> I would argue that separating events and GTI in different files is a bad idea, since it won't be possible later to assure the coherence of the data.

I also completely agree with that.

> A typical Guest Observer will get a very small set of data. The corresponding Use Case would be that someone obtains 50 hours of observing time on a given source. The Guest Observer will get O(100) event files (if they were split in individual runs) and the corresponding response functions. I propose to pack the events, the GTIs, and the IRF HDUs for a given run into a single file. It can't be simpler.

Again, I couldn't agree more. A guest observer will get several files and then just do a ls *.fits > files.txt and then pass the file list to a tool that creates observation files (like it is done in Fermi).
For the moment, we still need to allow different systematic tests (e.g. different background models, different PSF parametrisations, ...). Especially for

the background model, we have the following use case: We only have ~15 different background models. Each one is shared by many observations. E.g. all observations with similar pointing direction in horizon system have the same background model. Adding a background model to the event file would increase the storage significantly as we unnecessary duplicate information (~250kB per model). I am not sure if this is really practical.

Of course, we can think about using the index file structure only on the server and have a script (e.g. csiactdload) that downloads the files and bundles them properly. The user would get one file per observation. Do you think this might be a good idea?

> This would mean that we implement a way to infer from the event list where the GTI is (which could be done through the DSS keywords). Then we do not rely on specifying this explicitly. Things will be straight forward if we assume that the GTI will always be in the same file as the events are.

That sounds like a good idea. So we would just need sth like a GTI_EXT keyword.

I believe this topic is very controversial. In HESS we also have discussed this quite extensively and every one has a different use case and wants to try different things. At the end, probably each member institute will have a central storage for the FITS data, which is maintained by the admin who synchronises the data from a central place (not every member has access to the 'central HESS data centre'). Users just access the data by knowing where it is located (e.g. setting an environment variable). This will of course be very different for CTA guest observers who will retrieve their data via a web interface. I am not sure how we can accommodate all that, but I guess supporting different HDU names is a big step towards it.

#11 - 01/14/2016 03:18 PM - Knödlseder Jürgen

user#77 wrote:

> That sounds like a good idea. So we would just need sth like a GTI_EXT keyword.

I would indeed propose to use the DSREFx keyword that is already written in the event file and that says what the extension name is (so far the keyword value is :GTI, where the colon prefix is a convention taken over from Fermi). So far this information is (I think) basically ignored, but it should be easy to add some functionality that extracts that information from the header and uses it in GTI loading.

In the long run we may think about adding classes for a unified support of Data sub-space keywords, but for now I propose to just code this one specific usage.

I will look into the code that you have committed to see how we could implement such a scheme.

#12 - 01/14/2016 03:20 PM - Knödlseder Jürgen

I wanted to add: in case that the data sub-space selection keyword is not found, it should be assumed that the information is in the GTI extension.

#13 - 01/14/2016 03:23 PM - Mayer Michael

> I would indeed propose to use the DSREFx keyword that is already written in the event file and that says what the extension name is (so far the keyword value is :GTI, where the colon prefix is a convention taken over from Fermi). So far this information is (I think) basically ignored, but it should be easy to add some functionality that extracts that information from the header and uses it in GTI loading.

Make sense to me.

> I wanted to add: in case that the data sub-space selection keyword is not found, it should be assumed that the information is in the GTI extension.

Fully agree. In this way we keep the current scheme supported as well.

> I will look into the code that you have committed to see how we could implement such a scheme.

Ok great thanks. I am looking forward to hear your opinion.

#14 - 01/14/2016 04:02 PM - Knödlseder Jürgen

I have a few minutes so I started to look.

A first thing I recognised was that you added some logic to the XML reading that may conflict with what is existing so far. In fact, GCTAObservation::read() supports to read the following format:

```
<observation name="None" id="000001" instrument="CTA">
  <parameter name="Pointing" ra="186.156" dec="-64.019" />
  <parameter name="GoodTimeIntervals" tmin="0" tmax="34243.9" />
  <parameter name="TimeReference" mjdrefi="51544" mjdreff="0.5" timeunit="s" timesys="TT" timeref="LOCAL" />
  <parameter name="RegionOfInterest" ra="186.156" dec="-64.019" rad="5" />
  <parameter name="Deadtime" deadc="0.95" />
  <parameter name="Calibration" database="prod2" response="South_50h" />
</observation>
```

which means that a GTI can be explicitly specified in the observation definition XML file as a start and a stop time. The decoding of the GTI part is handled by the GGti::read() method (including the time reference), so it would be logic to extend that method to allow decoding of the alternative format

```
<observation name="None" id="000001" instrument="CTA">
  <parameter name="Pointing" ra="186.156" dec="-64.019" />
  <parameter name="GoodTimeIntervals" file="gti.fits[GTI]" />
  <parameter name="RegionOfInterest" ra="186.156" dec="-64.019" rad="5" />
  <parameter name="Deadtime" deadc="0.95" />
  <parameter name="Calibration" database="prod2" response="South_50h" />
</observation>
```

where the information is taken from a file (which does not need the time reference as this is found in the header of the GTI extension. The GGti::write() method should also be extended, but this would imply that GGti needs to know whether it has been constructed from a file or simply from the tmin and tmax parameters. At the end, all this means that the GTI filename would in fact be handled by the GGti class itself, and not the GCTAObservation class. This probably make some of your other modifications obsolete (but I would need to check that). I will continue to investigate this ...

#15 - 01/14/2016 04:08 PM - Mayer Michael

OK great thanks. I have seen this overlap. I guess GTIs should be handled in both ways via the XML interface then. I agree that GTIs handling the filenames itself would make some things easier. Thanks for investigating.

#16 - 01/16/2016 10:39 PM - Knödlseder Jürgen

user#77 wrote:

- Added flag m_has_gti_ext that signals if GTIs were present on file loading. This is used for writing again.

For what reason have you added this flag?

Normally, all event files should have a GTI, and the case that no GTI is in fact only a kluge as very early CTA files had no GTI (I think this was true for some of the 1DC files).

So the logic was that GammaLib autocorrects for a missing GTI file.

#17 - 01/17/2016 10:37 AM - Mayer Michael

> For what reason have you added this flag?

The main goal by implementing this flag was to ensure the philosophy "what is read in is written out". If GTIs were constructed from header keywords no GTI extension would be written out. But I have no strong opinion on that. We could write out GTIs in any case as well.

#18 - 01/18/2016 09:21 AM - Knödlseder Jürgen

user#77 wrote:

> > For what reason have you added this flag?
>
> The main goal by implementing this flag was to ensure the philosophy "what is read in is written out". If GTIs were constructed from header keywords no GTI extension would be written out. But I have no strong opinion on that. We could write out GTIs in any case as well.

I agree that it is against the philosophy, but when I introduced it it was simply a kluge that corrects for incomplete event files (i.e. event files with missing GTI). In principle we should even throw an exception to signal that no GTI were found as the analysis results without GTI may be wrong.

However, I'm still struggling with implementing the use case of having GTI in a different file than the events as this necessary needs a two step process (or in other words: GCTAEventList cannot cover that case as it does not know where this additional file is). Hence when GTIs are in a different file, the events will be loaded without GTI from GCTAEventList and the GTIs from GCTAObservation.

Otherwise, I've implemented the specification of the GTI extension name via DSS keywords in the header of the event file, and this works now fine.

#19 - 01/18/2016 09:34 AM - Knödlseder Jürgen

I looked again over http://gamma-astro-data-formats.readthedocs.org/en/latest/data_storage/index.html, here some comments (not sure where to post them otherwise, you may just echo them where appropriate):

- The OGIP standard is "DATE-OBS/TIME-OBS" and "DATE-END/TIME-END" instead of "TSTART_STR" and "TSTOP_STR" (would be nice if the format follows as close as possible the standard)
- You could store the reference time for the table in the extension header of the index file, which does not need to restrict the time information to MJD
- The GTI format should follow the OGIP standard (as OGIP has such a standard for GTI)
- I see the point of separating IRF (and specifically background) information into a specific file (and we definitely should support it), but the separation of GTIs from the events makes no sense (as the GTI should always reflect the temporal event selection that has been done). I would

> simply drop the GTI column in the HDU index file and make sure that the GTI can be found from the event file (using the DSS scheme)

#20 - 01/18/2016 10:22 AM - Mayer Michael

Thanks for your feedback.

> The OGIP standard is “DATE-OBS/TIME-OBS” and “DATE-END/TIME-END” instead of “TSTART_STR” and “TSTOP_STR” (would be nice if the format follows as close as possible the standard)

I guess the main motivation for the TIME_STR keywords instead of DATE_OBS was to have date and time in one string.

> You could store the reference time for the table in the extension header of the index file, which does not need to restrict the time information to MJD

I agree this makes sense. We have discussed to include sth like TSTART and TSTART_MJD. The main goal of this file is to simplify selections via cfitsio syntax. e.g. "ALT_PNT>30 && TSTART>???? && QUALITYhttps://github.com/gammapy/gamma-astro-data-formats/issues/20
You can ask Christoph to give you an write access to enter the discussion.

> Otherwise, I’ve implemented the specification of the GTI extension name via DSS keywords in the header of the event file, and this works now fine.

Thanks. Where can I find the changes?

#21 - 01/18/2016 11:00 AM - Knödlseder Jürgen

I pushed all changes into the gammalib/1598-loading-of-CTA-Eventlist-HDU branch, you may check them there. I have left for the moment the code specific to GTI loading from a different file in comments, but would like to remove it when merging into devel (we can always keep the branch on gammalib for record).

While working on the change I recognized that it may be better to move the loading and unloading of events to the GCTAEventList class. To handle the case of a lot of event files on a machine with limited memory capabilities, I implemented in the past a logic in GCTAObservation that loads events only when they are needed. This had the side effect that I had to add additional methods that handle extraction of ROI, energy boundary and also GTI information independently of the GCTAEventList class. To avoid code duplication I put some of the code in GCTASupport, but still, the GCTAObservation::git() method duplicates code in the GCTAEventList::read() method, which is problematic for code maintenance (it created in fact some trouble when I worked on the unit test as both methods were not doing the same :-). I added issue #1648 for this.

Another point is the GTI logic. For the moment we assume that the event list is always consistent with the GTIs, hence that the event list (on disk) has been filtered according to the GTIs. This implies that if somebody changes the GTIs he/she has to create a new copy of the event files. One may argue that this duplicates event files and requires disk space, on the other hand, only this assures the coherence of the data with the GTIs, and the disk space required is not so large. If we want to avoid data duplication, we would need to carefully evaluate first the implications. At which step would we like to modify the events according to the GTIs (upon loading or upon event access)? The first simplifies things (I think), while the latter would be more flexible (but probably slows down things). We also have to be aware that if we change GTIs we need to recompute ontime and livetime (and maybe something else?). We also would need a mechanism that avoids making GTIs "larger" than the existing GTIs (hence a carefully checking would be needed that assures that GTIs can only be more restrictive than the previous ones). So before doing any action here, I would suggest that we have a carefully discussion of this issue. Note that in any case, the events should always have a default GTI that corresponds to the initial selection. I created a discussion thread here: https://cta-redmine.irap.omp.eu/boards/6/topics/213

#22 - 01/18/2016 11:31 AM - Mayer Michael

> I pushed all changes into the gammalib/1598-loading-of-CTA-Eventlist-HDU branch, you may check them there. I have left for the moment the code specific to GTI loading from a different file in comments, but would like to remove it when merging into devel (we can always keep the branch on gammalib for record).

Excellent, I looked over the code. I really like the idea of having the GTI extension as DSS keyword. I also agree to remove the methods to load GTIs from a separate file when merging into devel. I will try to make this changes to
http://gamma-astro-data-formats.readthedocs.org/en/latest/events/index.html

> While working on the change I recognized that it may be better to move the loading and unloading of events to the GCTAEventList class. To handle the case of a lot of event files on a machine with limited memory capabilities, I implemented in the past a logic in GCTAObservation that loads events only when they are needed. This had the side effect that I had to add additional methods that handle extraction of ROI, energy boundary and also GTI information independently of the GCTAEventList class. To avoid code duplication I put some of the code in GCTASupport, but still, the GCTAObservation::git() method duplicates code in the GCTAEventList::read() method, which is problematic for code maintenance (it created in fact some trouble when I worked on the unit test as both methods were not doing the same . I added issue #1648 for this.

Yes, I agree this makes sense as well. Thanks for spotting this. If I understand correctly, the method GCTAObservation::events could thus be simplified to:

```
if (m_events == NULL) {
    m_events = new GCTAEventList;
    m_events->load(m_eventfile);
}
```

... or similar.

#23 - 01/18/2016 11:45 AM - Knödlseder Jürgen

user#77 wrote:

> Yes, I agree this makes sense as well. Thanks for spotting this. If I understand correctly, the method GCTAObservation::events could thus be simplified to:
> [...]
> ... or similar.

Even simpler: GCTAObservation::load() would just load the events, and GCTAEventList will actually deal with loading them physically or not. So all the m_events == NULL checks would disappear.

#24 - 01/18/2016 03:09 PM - Knödlseder Jürgen

- Status changed from Pull request to Closed

Merged into devel.

#25 - 01/18/2016 03:12 PM - Mayer Michael

> Even simpler: GCTAObservation::load() would just load the events, and GCTAEventList will actually deal with loading them physically or not. So all the m_events == NULL checks would disappear.

OK great.

> Merged into devel.

Argh you were a few minutes too fast smile.png
I have noticed while playing with the new code, that it actually does not support writing events in a filename with an extension addition, i.e. the following code doesn't work:

```
import gammalib
ev = gammalib.GCTAEventList("$CTOOLS/test/data/crab_events.fits.gz")
ev.save("test.fits[EVENTS2]")
```

#26 - 01/18/2016 03:24 PM - Knödlseder Jürgen

- Target version set to 1.1.0

- Start date set to 01/18/2016

#27 - 01/18/2016 03:27 PM - Knödlseder Jürgen

user#77 wrote:

> Argh you were a few minutes too fast smile.png
> I have noticed while playing with the new code, that it actually does not support writing events in a filename with an extension addition, i.e. the following code doesn't work:
> [...]

Sorry for that. We should indeed add a test case to test_CTA.cpp and make this work.

#28 - 01/18/2016 03:27 PM - Knödlseder Jürgen

- Status changed from Closed to In Progress

- % Done changed from 100 to 90

#29 - 01/18/2016 03:32 PM - Mayer Michael

I propose to change GCTAEventList::write to also take the extension name of the events as an argument. The extension can be extracted in GCTAEventList::save().
GCTAObservation::save() could then simply pass the extension to GCTAEventList::write().

Of course a test case would be useful to check for that to work. Are you on it or shall I take care of this?

#30 - 01/18/2016 03:43 PM - Mayer Michael

Btw: I also realised that GCTAObservation::save() always creates a new FITS file.
Wouldn't it be better to change

```
GFits fits;
...
fits.saveto(filename, clobber);
```

to

```
GFilename fname(filename);
GFits fits(fname.filename(), true)
write(fits, fname.extension());
fits.saveto(fname.filename(), clobber)
```

When having one file with multiple event lists, we would overwrite this file in ctselect with the last event list (if clobber=yes) if we keep it as above, right?
When we use the logic below, the output of ctselect would be one file with multiple extensions again.

#31 - 01/18/2016 04:06 PM - Knödlseder Jürgen

user#77 wrote:

> I propose to change GCTAEventList::write to also take the extension name of the events as an argument. The extension can be extracted in GCTAEventList::save().
> GCTAObservation::save() could then simply pass the extension to GCTAEventList::write().
>
> Of course a test case would be useful to check for that to work. Are you on it or shall I take care of this?

We may need to do a bit more, in particular if it is planned to have several event extensions in a single file.

The actual code

```
void GCTAEventList::save(const std::string& filename,
                         const bool& clobber) const
{
    // Create empty FITS file
    GFits fits;

    // Write event list
    write(fits);

    // Save FITS file
    fits.saveto(filename, clobber);

    // Return
    return;
}
```

will always create a new file (or overwrite an existing file) that contains a single event list. The GFits::save() and GFits::saveto() methods always write the entire FITS object from memory to disk. The difference is that GFits::save() opens any existing file before (or creates a new file if it does not yet exist), while GFits::saveto() specifies the filename only upon saving, and will not read back extensions from an existing file (as they may conflict with those in the GFits object).

I modified recently the GFits::save() method so that adding an extension to an existing file actually works.

The code would need to be modified as follows:

```
void GCTAEventList::save(const std::string& filename,
                         const bool& clobber) const
{
    // Create file name
    GFilename fname(filename);

    // Open or create FITS file
    GFits fits(fname.filename(), true);

    // Write event list
    write(fits, fname.extname("EVENTS"));

    // Save FITS file
    fits.save(clobber);

    // Return
    return;
}
```

Maybe the following code would even make the extname argument for the write() method obsolete:

```
void GCTAEventList::save(const std::string& filename,
                         const bool& clobber) const
{
    // Open or create FITS file
    GFits fits(filename, true);

    // Write event list
    write(fits);

    // Save FITS file
    fits.save(clobber);

    // Return
    return;
}
void GCTAEventList::write(const GFits& fits)
{
    // Initialise filename from fits file
    GFilename fname = GFilename(fits.filename());

    // Allocate FITS binary table HDU
    GFitsBinTable* events = new GFitsBinTable;

    // Write events
    write_events(*events);

    // Set extension name
    events->extname(fname.extname("EVENTS"));
```

The logic of using the filename that is stored when opening the FITS file is in fact used in the GCTAEventList::read method. At the end, I'm however not convinced that we want to do this as this makes the handling of the extension name "opaque". In the long run I would like to add a GFitsExtension class (see #1602) and add this class as explicit argument when it is needed. So I would suggest to add an extname argument to the write() method for now.

I'm happy if you do the change.

#32 - 01/18/2016 04:42 PM - Mayer Michael

> So I would suggest to add an extname argument to the write() method for now.

I realised I was at this point before. Since GCTAEventList is inheriting from GEvents, we cannot simply change the interface of the write method. This would require a change in all GEvents derived classes which is probably not what we want. I would therefore probably go for recovering the extension name from the filename in the read method.

#33 - 01/18/2016 04:45 PM - Knödlseder Jürgen

Good point. I agree to recover the extension name from the FITS object.

#34 - 01/18/2016 04:57 PM - Mayer Michael

Ok I am somewhat stuck now.
In the save method we want to pass a file name that might not exist yet. However, the following code does not work:

```
import gammalib
f = gammalib.GFits("file_that_doesnt.fits[EXIST]", True)
```

I get the following error:
RuntimeError: *** ERROR in GFits::open(std::string&): Unable to open FITS file "file_that_doesnt.fits[EXIST]" (status=125)
The code will only work if the file already exists.

Do you have an idea to circumvent this problem?

#35 - 01/18/2016 05:22 PM - Mayer Michael

> Do you have an idea to circumvent this problem?

Simplest workaround would be to add a protected member std::string m_extension to GCTAEventList. Thus we could simply access the property and change it according to the input filename.

#36 - 01/18/2016 05:32 PM - Knödlseder Jürgen

user#77 wrote:

> Ok I am somewhat stuck now.
> In the save method we want to pass a file name that might not exist yet. However, the following code does not work:
> [...]
> I get the following error:
> RuntimeError: *** ERROR in GFits::open(std::string&): Unable to open FITS file "file_that_doesnt.fits[EXIST]" (status=125)
> The code will only work if the file already exists.
>
> Do you have an idea to circumvent this problem?

Indeed, GFits does not like creating a file with an extension name.

All this will be solved once we replace all std::string& filename by GFilename& filename arguments, as the latter handles properly the FITS extensions.

The reason why I have not yet done this replacement is that the automatic string to GFilename conversion does not yet work in Python (I would like to be able to write open("fits.fits[EVENTS]") for a GFilename argument in Python).

We can however already switch the m_filename argument in GFits from std::string to GFilename so that internally the function works. I will do this change later (I created an issue for this as this may imply quite some code change #1649).

#37 - 01/18/2016 05:35 PM - Knödlseder Jürgen

user#77 wrote:

> > Do you have an idea to circumvent this problem?
>
> Simplest workaround would be to add a protected member std::string m_extension to GCTAEventList. Thus we could simply access the property and change it according to the input filename.

Would be a quick workaround to get it going.

#38 - 01/18/2016 07:09 PM - Mayer Michael

- Status changed from In Progress to Pull request

- % Done changed from 90 to 100

Ok I have added this change on *1598-loading-of-CTA-Eventlist-HDU*. Feel free to modify and expand when adding GFilename as member to GFits. To circumvent the const correctness, I had to add const_cast to GCTAObservation::write() and GCTAEventList::save.

#39 - 01/19/2016 12:06 AM - Knödlseder Jürgen

- Status changed from Pull request to Feedback

I added the GFilename member to GFits and made use of it saving the event files. I added a gti_extname to GCTAObservation and GCTAEventList to allow the setting of the extension name of the GTI.

Now the following code works (see test_CTA.cpp):

```
GCTAObservation run;
run.load(cta_events);
run.gti_extname("GTI2");
run.save("test_cta_events1.fits[EVENTS2]", true);
...
GFits fits2;
run.gti_extname("GTI1");
run.write(fits2, "EVENTS1");
run.gti_extname("GTI2");
run.write(fits2, "EVENTS2");
run.gti_extname("GTI3");
run.write(fits2, "EVENTS3");
fits2.saveto("test_cta_events2.fits", true);
```

Note that if multiple event extension should be written into a single file the write() method should be used instead of the save() method.

I still don't like too much the asymmetry between the EVENTS and GTI extension. We may think about adding arguments for both extension names to the GCTAObservation::write() method. This would at least allow to write:

```
run.write(fits2, "EVENTS1", "GTI1");
```

Alternatively we could think about having a format for multiple extension names. This is not FITS standard but would avoid adding additional arguments. For example:

```
run.save("test_cta_events1.fits[EVENTS2;GTI2]", true);
run.write(fits2, "EVENTS1;GTI1");
```

I added information about this to #1602. I suggest to live with the actual code for now, and think about this issue a bit more.

#40 - 01/19/2016 01:16 AM - Knödlseder Jürgen

Finally, I went ahead and modified the classes to allow specifying both extension names in the GCTAObservation::write() and GCTAEventList::write() method. For the latter case, I simply added another write() method with two additional arguments to the interface so that I did not have to modify the GEvents interface. The original GCTAEventList::write() method now calls the new GCTAEventList::write() with an auto determination of the extension names.

I further implemented a logic in GCTAObservation::save that decodes two extension strings in the filename.

Now the following example works:

```
GCTAObservation run;
run.load(cta_events);
run.save("test_cta_events1.fits[EVENTS2;GTI2]", true);
...
run.load(cta_events);
GFits fits2;
run.write(fits2, "EVENTS1", "GTI1");
run.write(fits2, "EVENTS2", "GTI2");
run.write(fits2, "EVENTS3", "GTI3");
fits2.saveto("test_cta_events2.fits", true);
```

I removed the gti_extname() methods introduced earlier.

#41 - 01/19/2016 11:12 AM - Mayer Michael

Thanks for your effort. The solution using a semicolon to separate the extension names sounds reasonable. I went over the code and did some testing.
I have a few questions to GCTAObservation::save():

- If not GTI extension name is given in the filename, we would use "GTI" instead. Don't we want to use m_gti_extname of the event list instead? Not sure how to implement this though
- Do we want to strip the white spaces? E.g. for the IRFs, we support "EFFECTIVE AREA", should "MY EVENTS" be supported, too?
- We always write to a new FITS file. Is that intended? Or do we want to support to write both extensions to a file that already exists? I am thinking about the use case of having one file with several EVENT and GTI extensions. Running e.g. ctselect would overwrite the file for each observation which is probably not ideal.
  In addition, we might need to add a check if the extension we want to write to already exists and throw an exception (or do we just overwrite?

Probably ctselect has to be adapted, too. E.g. in ctselect::save_event_list(), we explicitly check for "EVENTS" and "GTI" extensions.

#42 - 01/19/2016 04:38 PM - Mayer Michael

In addition ctselect doesn't work at the moment:
When saving the event files I get the following error:

```
2016-01-19T15:29:51: +====================+
2016-01-19T15:29:51: | Save observations |
2016-01-19T15:29:51: +====================+
2016-01-19T15:29:51: Save "selected/selected_events_081849.fits.gz[EVENTS]"
libc++abi.dylib: terminating with uncaught exception of type GException::fits_error: *** ERROR in GFits::free_members(): cannot write to readonly file
 (status=112).
Abort trap: 6
```

#43 - 01/20/2016 08:53 AM - Knödlseder Jürgen

user#77 wrote:

> In addition ctselect doesn't work at the moment:
> When saving the event files I get the following error:
> [...]

I'm wondering whether this is related to the fact that you want to write in a gzipped file, which is by definition read only (cfitsio cannot write in gzipped files). The following works here:

```
$ ctselect
```

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

#44 - 01/20/2016 09:15 AM - Mayer Michael

You are right.
I was using gzipped files as input, which wasn't a problem before. Now I was just providing extension names, too. It seems that ctselect has a problem in throwing away the .gz ending when creating the outfile in case an extension name is provided. The problem is in ctselect::set_outfile_name: the functions only cuts away trialing .gz endings. In case the infile is e.g. events.fits.gz[EVENTS], the outfile will be e.g. selected_events.fits.gz[EVENTS]. Accordingly, cfitsio can't save this file.
I guess this is the reason for the error, right? Thus we might need to introduce a GFilename instance in this function and cut away .gz from the pure input filename?

#45 - 01/20/2016 09:27 AM - Knödlseder Jürgen

user#77 wrote:

> You are right.
> I was using gzipped files as input, which wasn't a problem before. Now I was just providing extension names, too. It seems that ctselect has a problem in throwing away the .gz ending when creating the outfile in case an extension name is provided. The problem is in ctselect::set_outfile_name: the functions only cuts away trialing .gz endings. In case the infile is e.g. events.fits.gz[EVENTS], the outfile will be e.g. selected_events.fits.gz[EVENTS]. Accordingly, cfitsio can't save this file.
> I guess this is the reason for the error, right? Thus we might need to introduce a GFilename instance in this function and cut away .gz from the pure input filename?

The general rule is to not provide the .gz extension as cfitsio will automatically search also for files with such an extension. So it should work on gzipped files by providing events.fits[EVENTS] on input.

Interesting that it worked before as I don't think that I changed the code. ctselect will simply build a filename from the input filename, keeping the extension. But I think the new thing is that when adding the extension name [EVENTS] it does not work (and probably did not work before).

I definitely have to rethink the ctselect tool as it is based on the assumption that the extensions have fixed names.

#46 - 01/20/2016 09:33 AM - Mayer Michael

Ok I understand. Removing .gz from the input file name works well, thanks.

> Interesting that it worked before as I don’t think that I changed the code. ctselect will simply build a filename from the input filename, keeping the extension.

In ctselect::set_outfile_name(), the following code is used:

```
// Strip any ".gz"
outname = gammalib::strip_chars(outname, ".gz");
```

This removes .gz from "events.fits.gz" but not from "events.fits.gz[EVENTS]"

#47 - 01/20/2016 12:25 PM - Knödlseder Jürgen

user#77 wrote:

> Thanks for your effort. The solution using a semicolon to separate the extension names sounds reasonable. I went over the code and did some testing.
> I have a few questions to GCTAObservation::save():
>
> - If not GTI extension name is given in the filename, we would use "GTI" instead. Don't we want to use m_gti_extname of the event list instead? Not sure how to implement this though

Indeed, I added a GCTAEventList::gtiname() method to recover the GTI name. This method is now used in ctselect to set the proper GTI name.

> - Do we want to strip the white spaces? E.g. for the IRFs, we support "EFFECTIVE AREA", should "MY EVENTS" be supported, too?

The gammalib::strip_whitespace() functions only strips leading and trailing white spaces, hence it would not alter "MY EVENTS".
gammalib::strip_chars(), which is used by gammalib::strip_whitespace(), also works only on leading and trailing characters.

> - We always write to a new FITS file. Is that intended? Or do we want to support to write both extensions to a file that already exists? I am thinking about the use case of having one file with several EVENT and GTI extensions. Running e.g. ctselect would overwrite the file for each observation which is probably not ideal.

It is true that we always write a new FITS file, but all extensions other than the events and GTI extension on which ctselect currently works are copied over from the input file. So your use case works in case that you provide a file with several EVENT and GTI extensions on input.

> In addition, we might need to add a check if the extension we want to write to already exists and throw an exception (or do we just overwrite?

The logic is that if clobber=yes the extensions will be overwritten, with clobber=no an exception will be thrown.

> Probably ctselect has to be adapted, too. E.g. in ctselect::save_event_list(), we explicitly check for "EVENTS" and "GTI" extensions.

I'm about of doing that. I think now everything works. I will merge the code soon.

By the way: you can even specify a different extension name on output, i.e.

```
$ ctselect
Input event list or observation definition XML file [events2.fits[EVENTS2]]
RA for ROI centre (degrees) (0-360) [83.63]
Dec for ROI centre (degrees) (-90-90) [22.01]
Radius of ROI (degrees) (0-180) [3.0]
Start time (CTA MET in seconds) [0.0]
End time (CTA MET in seconds) [0.0]
Lower energy limit (TeV) [0.1]
Upper energy limit (TeV) [100.0]
Output event list or observation definition XML file [selected_events.fits] selected_events.fits[EVENTS3]
```

writes a new EVENTS3 extension.

#48 - 01/20/2016 12:27 PM - Knödlseder Jürgen

user#77 wrote:

> Ok I understand. Removing .gz from the input file name works well, thanks.
>
> > Interesting that it worked before as I don’t think that I changed the code. ctselect will simply build a filename from the input filename, keeping the extension.
>
> In ctselect::set_outfile_name(), the following code is used:
> [...]
> This removes .gz from "events.fits.gz" but not from "events.fits.gz[EVENTS]"

This should also be fixed in the new version.

#49 - 01/20/2016 04:24 PM - Knödlseder Jürgen

I merged a new version into devel. Please check the NEWS file to see what changed. I hope this takes care of all issues you identified (and even does more: you may specify extension names for the output file).

#50 - 01/25/2016 05:03 PM - Mayer Michael

Apologies for the slow reaction.
Thanks for making the updates and changes. I tried to test and run many of the use cases and everything seems to work well. I guess we finally can close this issue smile.png

#51 - 01/25/2016 05:05 PM - Knödlseder Jürgen

- Status changed from Feedback to Closed

Great. No worries for a few days of delay, your feedback is still very quick :-) I close this now.

Files

SDS07.pdf	183 KB	01/13/2016	Knödlseder Jürgen
fits_standard30aa.pdf	687 KB	01/13/2016	Knödlseder Jürgen