GammaLib - Action #3592

Add information about equivalent CO2 emissions after execution of a tool or script

04/02/2021 12:44 PM - Knödlseder Jürgen

Status:	Closed	**Start date:**	04/02/2021
Priority:	Normal	**Due date:**	
Assigned To:	Knödlseder Jürgen	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	2.0.0		

Description

Each ctool and cscript terminates execution with the logging of the used CPU time.

This CPU time can be converted into equivalent CO2 emissions assuming some emission factors that convert CPU time into equivalent CO2 emissions. These emission factors may be location dependent, and a location dependent lookup table could be used for the conversion.

To inform the user about the carbon imprint of the job execution, an estimate of the equivalent CO2 emissions should be added to the log files in GApplication::log_trailer() and a method GApplication::eCO2() should be added that returns the current equivalent CO2 emissions produced by an applications in grams.

History

#1 - 04/02/2021 12:44 PM - Knödlseder Jürgen

- Project changed from ctools to GammaLib

- Target version changed from 2.0.0 to 2.0.0

#2 - 04/02/2021 03:25 PM - Knödlseder Jürgen

- File GES-h-coeur-GRICAD-2020.pdf added

- Status changed from New to In Progress

- Assigned To set to Knödlseder Jürgen

- % Done changed from 0 to 10

It seems complicated to extract the host country of the server although a possible solution could be to use the database at http://software77.net/geo-ip/ and a code inspired by https://gist.github.com/markusl/1087604 to convert the host IP into a country code. But even after implementing this solution it is not obvious that emission or impact factors exist for all the countries.

As a first step, an easy working solution is to use a single emission factor for all countries. A comprehensive study can be found at https://hal.archives-ouvertes.fr/hal-02549565v4/document which is also attached here: attachment:GES-h-coeur-GRICAD-2020.pdf.

The study dates from 2020 and estimates the imprint of 1h.core for the DAHU cluster of the UMS GRICAD in 2019. The study determines an imprint of 4.68 g eCO2 / CPU hour. The imprint factor includes:

- server fabrication
- server environment
- server usage (electricity)
- travelling of personnel in the context of their work
- travelling of personnel from home to office
- personnel equipment
- personnel energy

Note that the impact of electricity was computed assuming 108 g eCO2 / kWh consumed, and about 50% of the total imprint was due to electricity consumption. For countries with a more carbon intensive electricity production, the CO2 imprint will be accordingly larger.

#3 - 04/02/2021 03:28 PM - Knödlseder Jürgen

- % Done changed from 10 to 20

One possibility would be to subtract the electricity production part from the impact study and to replace it with a country-dependent estimate. This may be done in a future version of the code, yet since this would be quite opaque it remains questionable whether this is actually useful.

#4 - 04/07/2021 11:47 AM - Knödlseder Jürgen

- % Done changed from 20 to 30

Luigi also pointed out this tool: https://codecarbon.io

#5 - 02/18/2022 12:56 PM - Knödlseder Jürgen

To globally track resource usage, application usage information should be written in the GApplication::free_members method into a global ASCII file. This will allow to derive usage statistics, including information about the carbon footprint, as function of time.

A proper location of the ASCII file would be in a dedicated directory $HOME/.gamma situated in the user's home directory.

Writing a CSV ASCII file, including one line for the header and one line each time that the GApplication::free_members method is called, is probably adequate for the purpose. A method should be devised that prevents the file to grow huge.

Columns should include:

- a time stamp
- the GammaLib version
- the country code where the code was executed (this would be needed for country-dependent eCO2 tracking)
- the application name
- the application version
- the application elapsed wall clock time
- the application elapsed CPU time
- the application carbon footprint

The appending of a line to this file should be OMP protected.

#6 - 02/18/2022 02:05 PM - Knödlseder Jürgen

- % Done changed from 30 to 40

I added a method GApplication::write_statistics() that writes the application statistics upon destruction of an application. I added a function gammalib::strdate() to write the date string. The function was copied from the GLog::strdate() which was removed and replaced by the function to avoid code duplication.

I check that the folder and file are created if they do not exist, and that lines are added if a tool or script is run:

```
$ more /Users/jurgen/.gamma/statistics.csv
Date,GammaLib version,Country,Application,Version,Wall clock seconds,CPU seconds,g eCO2
2022-02-18T12:53:23,2.0.0.dev,FR,cscaldb,2.0.0.dev,0.000000e+00,2.141300e-02,2.783690e-05
```

2022-02-18T12:54:22,2.0.0.dev,FR,test_GApplication,1.0.0,0.000000e+00,3.933000e-03,5.114200e-06

#7 - 02/18/2022 08:37 PM - Knödlseder Jürgen

- % Done changed from 40 to 50

I implemented the function gammalib::host_country() that retrieves the country code using a curl query to http://ip-api.com/line/?fields=countryCode.

#8 - 02/19/2022 12:16 AM - Knödlseder Jürgen

- % Done changed from 50 to 70

I added a file refdata/emission-factors.fits that contains the emission factors for electricity generation for a large number of countries. I built the file from the excel table of the Bilan Carbone method, and added two-digit country codes so that they can be used to find the appropriate emission factors.

I added code to the GApplication::gCO2e() method so that the information is read from the file and used. Some caching is implemented, so that information is only read once, and the if the country argument to the method does not change, no search through the table is required. This should assure that the information can be accessed pretty fast.

The computation now splits the electricity-dependent emission from the rest, and the country specific emission factors are only used for the electricity-dependent part.

I added some documentation to the ctools site so that users understand what's behind the numbers in the log file.

I still need to think about how to prevent the generation of a huge statistics.csv file.

I also should add an example script to ctools to display carbon footprint data.

#9 - 02/19/2022 01:29 PM - Knödlseder Jürgen

A solution to the size of the statistics file problem may be the creation of a GammaLib daemon that regularly purges the statistics file while creating high-level statistics information, for example on a daily basis. This high-level information can then be used by plotting scripts. The high-level file should be in XML format.

The size of the high-level statistic file will be rather limited. It may contain the daily wall clock and CPU hours as well as carbon footprint per application. With currently 61 different applications, the maximum number of lines per day would be of the order of 100, counting additional lines for summary information. This means that 36500 lines would be written per year, and if each line has 100 characters, the file size would be about 3.7 MB / year. So even after ten years of usage, the file size should not exceed 40 MB.

The daemon class can be inspired by the GVOHub class which is already a kind of daemon. The GVOHub daemon is started using the GVOClient::require_hub() method, and a similar method could be implemented in GApplication. The daemon would be started if an application is instantiated. In principle it would never die, but the GApplication launcher should make sure that no one killed the daemon, and if the daemon was killed, a new daemon should be started.

#10 - 02/19/2022 11:30 PM - Knödlseder Jürgen

- % Done changed from 70 to 80

I added the GDaemon class that implements a GammaLib daemon. For the time being the only task of the daemon is to generate and update on an hourly basis a high-level application statistics XML file that provides for each day the applications that were run, their wall clock time, their CPU time and their carbon footprint. After the update the low-level file is deleted.

The daemon is started whenever an application is instanciated in the GApplication::init_members method. In case that a daemon is already running no other daemon will be started. The daemon never dies, unless one issues a kill -9 command. Any time that a new application is instanciated a new daemon is started again. A lock file daemon.lock is written in the .gamma folder containing the process ID of the daemon. If the daemon is killed the lock file will prevail, however it will not block starting of a new daemon which will then overwrite process ID in the lock file.

Here is an example of the XML file that is generated by the daemon:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<statistics title="High-level statistic">
  <header>
    <dates>
      <creation>2022-02-19T22:09:05</creation>
      <modified>2022-02-19T22:13:05</modified>
      <start>2022-02-18T22:21:56</start>
      <stop>2022-02-19T22:13:04</stop>
    </dates>
  </header>
  <data>
    <daily>
      <date>
        <value>2022-02-18</value>
        <tools>
          <test_GApplication version="1.0.0" calls="10" wall="0" cpu="0.034568" gCO2e="3.6524768e-05" />
          <ctobssim version="2.0.0.dev" calls="1795" wall="178" cpu="175.230736" gCO2e="0.172060195882" />
          <ctselect version="2.0.0.dev" calls="87" wall="65" cpu="45.005446" gCO2e="0.0441931424436" />
          <ctphase version="2.0.0.dev" calls="13" wall="0" cpu="0.162489" gCO2e="0.0001595595941" />
          <ctfindvar version="2.0.0.dev" calls="9" wall="3" cpu="2.356012" gCO2e="0.002313321049" />
          <ctbin version="2.0.0.dev" calls="31" wall="6" cpu="4.978268" gCO2e="0.004889514038" />
          <ctlike version="2.0.0.dev" calls="220" wall="322" cpu="323.513057" gCO2e="0.31765157135" />
          <cttsmap version="2.0.0.dev" calls="10" wall="12" cpu="10.679273" gCO2e="0.01048572233" />
          <ctmodel version="2.0.0.dev" calls="28" wall="6" cpu="4.591355" gCO2e="0.004508190556" />
          ...
          <comlixfit version="2.0.0.dev" calls="5" wall="44" cpu="42.639466" gCO2e="0.041866632735" />
          <comlixmap version="2.0.0.dev" calls="5" wall="71" cpu="70.110307" gCO2e="0.06883955744" />
          <comsrcdetect version="2.0.0.dev" calls="3" wall="0" cpu="0.035303" gCO2e="3.4668041e-05" />
        </tools>
      </date>
      <date>
        <value>2022-02-19</value>
        <tools>
          <cscaldb version="2.0.0.dev" calls="79" wall="27" cpu="2.098166" gCO2e="0.00206024862" />
          <ctobssim version="2.0.0.dev" calls="1" wall="35" cpu="7.797325" gCO2e="0.007656001" />
        </tools>
      </date>
    </daily>
  </data>
</statistics>
```

What is missing now is a cscript that analysis the XML file and eventually creates a graphical representation of the statistics.

#11 - 02/20/2022 09:03 AM - Knödlseder Jürgen

There is apparently some multiple counting involved when running a cscript. Here the output in the low-level file when cspull is called with 10 trials. There are 10 calls to ctobssim, 10 calls to ctlike and 11 calls to cspull. The 11 calls to cspull are due to the multi threading.

Note also that the final message in the cspull log file noted 28 wall clock seconds, 0.177706 CPU seconds and a carbon footprint of 0.000175366 g eCO2 (this message corresponds to the last line of the file). While the wall clock seconds seem to correspond to the experienced time, CPU seconds and carbon footprint do not seem to be realistic. The sum of all values below gives 266 wall clock seconds, 202 CPU seconds and a carbon footprint of 0.1988 g eCO2.

2022-02-20T07:27:56,2.0.0.dev,FR,ctobssim,2.0.0.dev,1.200000e+01,1.040837e+01,1.022041e-02
2022-02-20T07:27:57,2.0.0.dev,FR,ctobssim,2.0.0.dev,1.300000e+01,1.053125e+01,1.034103e-02
2022-02-20T07:27:57,2.0.0.dev,FR,ctobssim,2.0.0.dev,1.300000e+01,1.061121e+01,1.041946e-02
2022-02-20T07:27:57,2.0.0.dev,FR,ctlike,2.0.0.dev,1.000000e+00,2.971870e-01,2.918024e-04
2022-02-20T07:27:57,2.0.0.dev,FR,cspull,2.0.0.dev,1.300000e+01,1.071088e+01,1.051675e-02
2022-02-20T07:27:57,2.0.0.dev,FR,ctobssim,2.0.0.dev,1.300000e+01,1.062828e+01,1.043623e-02
2022-02-20T07:27:57,2.0.0.dev,FR,ctobssim,2.0.0.dev,1.300000e+01,1.063707e+01,1.044500e-02
2022-02-20T07:27:57,2.0.0.dev,FR,ctlike,2.0.0.dev,0.000000e+00,4.197420e-01,4.121361e-04
2022-02-20T07:27:57,2.0.0.dev,FR,cspull,2.0.0.dev,1.300000e+01,1.095616e+01,1.075758e-02
2022-02-20T07:27:57,2.0.0.dev,FR,ctobssim,2.0.0.dev,1.200000e+01,1.063485e+01,1.044266e-02
2022-02-20T07:27:57,2.0.0.dev,FR,ctlike,2.0.0.dev,0.000000e+00,4.236440e-01,4.159674e-04
2022-02-20T07:27:57,2.0.0.dev,FR,cspull,2.0.0.dev,1.300000e+01,1.103980e+01,1.083971e-02
2022-02-20T07:27:57,2.0.0.dev,FR,ctlike,2.0.0.dev,0.000000e+00,4.187610e-01,4.111739e-04
2022-02-20T07:27:57,2.0.0.dev,FR,cspull,2.0.0.dev,1.300000e+01,1.105285e+01,1.085252e-02
2022-02-20T07:27:58,2.0.0.dev,FR,ctobssim,2.0.0.dev,1.300000e+01,1.063223e+01,1.044005e-02
2022-02-20T07:27:58,2.0.0.dev,FR,ctobssim,2.0.0.dev,1.300000e+01,1.069416e+01,1.050092e-02
2022-02-20T07:27:58,2.0.0.dev,FR,ctlike,2.0.0.dev,1.000000e+00,3.895300e-01,3.824717e-04
2022-02-20T07:27:58,2.0.0.dev,FR,ctlike,2.0.0.dev,1.000000e+00,3.390940e-01,3.329489e-04
2022-02-20T07:27:58,2.0.0.dev,FR,cspull,2.0.0.dev,1.400000e+01,1.098224e+01,1.078319e-02
2022-02-20T07:27:58,2.0.0.dev,FR,cspull,2.0.0.dev,1.300000e+01,1.103023e+01,1.083031e-02
2022-02-20T07:27:58,2.0.0.dev,FR,ctlike,2.0.0.dev,0.000000e+00,2.088240e-01,2.050410e-04
2022-02-20T07:27:58,2.0.0.dev,FR,cspull,2.0.0.dev,1.300000e+01,1.084684e+01,1.065024e-02
2022-02-20T07:27:58,2.0.0.dev,FR,ctlike,2.0.0.dev,0.000000e+00,1.368120e-01,1.343352e-04
2022-02-20T07:27:58,2.0.0.dev,FR,cspull,2.0.0.dev,1.300000e+01,1.083616e+01,1.063976e-02
2022-02-20T07:28:04,2.0.0.dev,FR,ctobssim,2.0.0.dev,7.000000e+00,6.679064e+00,6.558009e-03
2022-02-20T07:28:04,2.0.0.dev,FR,ctobssim,2.0.0.dev,7.000000e+00,6.499302e+00,6.381505e-03
2022-02-20T07:28:04,2.0.0.dev,FR,ctlike,2.0.0.dev,0.000000e+00,3.422930e-01,3.360909e-04
2022-02-20T07:28:04,2.0.0.dev,FR,cspull,2.0.0.dev,7.000000e+00,7.023621e+00,6.896319e-03
2022-02-20T07:28:04,2.0.0.dev,FR,ctlike,2.0.0.dev,0.000000e+00,1.878400e-01,1.844374e-04
2022-02-20T07:28:04,2.0.0.dev,FR,cspull,2.0.0.dev,7.000000e+00,6.689537e+00,6.568290e-03
2022-02-20T07:28:04,2.0.0.dev,FR,cspull,2.0.0.dev,2.800000e+01,1.789560e-01,1.757134e-04

Without multithreading there are 10 calls to ctobssim, 10 calls to ctlike and 1 call to cspull.

2022-02-20T07:41:23,2.0.0.dev,FR,ctobssim,2.0.0.dev,6.000000e+00,6.372908e+00,6.258124e-03
2022-02-20T07:41:23,2.0.0.dev,FR,ctlike,2.0.0.dev,0.000000e+00,1.595920e-01,1.567014e-04
2022-02-20T07:41:30,2.0.0.dev,FR,ctobssim,2.0.0.dev,7.000000e+00,6.247217e+00,6.133987e-03
2022-02-20T07:41:30,2.0.0.dev,FR,ctlike,2.0.0.dev,0.000000e+00,2.289820e-01,2.248327e-04
2022-02-20T07:41:36,2.0.0.dev,FR,ctobssim,2.0.0.dev,6.000000e+00,6.291500e+00,6.177467e-03
2022-02-20T07:41:36,2.0.0.dev,FR,ctlike,2.0.0.dev,0.000000e+00,2.199920e-01,2.160066e-04
2022-02-20T07:41:43,2.0.0.dev,FR,ctobssim,2.0.0.dev,7.000000e+00,6.254114e+00,6.140759e-03
2022-02-20T07:41:43,2.0.0.dev,FR,ctlike,2.0.0.dev,0.000000e+00,2.235340e-01,2.194844e-04
2022-02-20T07:41:49,2.0.0.dev,FR,ctobssim,2.0.0.dev,6.000000e+00,6.291946e+00,6.177906e-03
2022-02-20T07:41:49,2.0.0.dev,FR,ctlike,2.0.0.dev,0.000000e+00,1.958030e-01,1.922560e-04
2022-02-20T07:41:56,2.0.0.dev,FR,ctobssim,2.0.0.dev,7.000000e+00,6.277482e+00,6.163704e-03
2022-02-20T07:41:56,2.0.0.dev,FR,ctlike,2.0.0.dev,0.000000e+00,2.556650e-01,2.510321e-04
2022-02-20T07:42:02,2.0.0.dev,FR,ctobssim,2.0.0.dev,6.000000e+00,6.233919e+00,6.120931e-03
2022-02-20T07:42:02,2.0.0.dev,FR,ctlike,2.0.0.dev,0.000000e+00,1.598260e-01,1.569301e-04
2022-02-20T07:42:09,2.0.0.dev,FR,ctobssim,2.0.0.dev,7.000000e+00,6.248352e+00,6.135103e-03
2022-02-20T07:42:09,2.0.0.dev,FR,ctlike,2.0.0.dev,0.000000e+00,1.264480e-01,1.241581e-04
2022-02-20T07:42:15,2.0.0.dev,FR,ctobssim,2.0.0.dev,6.000000e+00,6.238046e+00,6.124983e-03
2022-02-20T07:42:15,2.0.0.dev,FR,ctlike,2.0.0.dev,0.000000e+00,3.434310e-01,3.372073e-04
2022-02-20T07:42:22,2.0.0.dev,FR,ctobssim,2.0.0.dev,7.000000e+00,6.287184e+00,6.173230e-03
2022-02-20T07:42:22,2.0.0.dev,FR,ctlike,2.0.0.dev,0.000000e+00,1.943400e-01,1.908195e-04
2022-02-20T07:42:22,2.0.0.dev,FR,cspull,2.0.0.dev,7.000000e+01,6.498156e+01,6.380377e-02

Here the summary of the outputs:

Value	Wall clock	CPU seconds	gCO2e	Comment

Multi-threading log file	28	0.177706	0.000175366	CPU time incorrect
Multi-threading sum in statistics.csv	266	202	0.1988	
Single thread log file	70	64.9814	0.0638036	Times correct
Single thread sum in statistics.csv	135	129.832	0.127	

For the single thread there is simply a double counting, as the cspull script counts the times that was used by all the child processes.

For the multiple threads summing the 10 cspull child processes gives a wall clock of 119 s, CPU time of 101 s and footprint of 0.1 which is a bit larger but of the same order as the values given in the log file of the single thread run. Hence the issue here is that a multi-thread Python script does not properly sum up the CPU seconds to the threads, hence the resulting carbon footprint is wrong.

#12 - 02/20/2022 09:04 AM - Knödlseder Jürgen

- File cspull-nthreads1-statistics.xlsx added

- File cspull-statistics.xlsx added

Here the low-level statistics output as excel files.

#13 - 02/20/2022 10:26 AM - Knödlseder Jürgen

I added a logic where only the parent application writes out the statistics information. Here is the result when calling cspull with nthreads=1

Date,GammaLib version,Country,Application,Version,Wall clock seconds,CPU seconds,g eCO2
2022-02-20T09:23:08,2.0.0.dev,FR,cspull,2.0.0.dev,7.200000e+01,6.563266e+01,6.444307e-02

and here the results for multi-threading

Date,GammaLib version,Country,Application,Version,Wall clock seconds,CPU seconds,g eCO2
2022-02-20T09:25:11,2.0.0.dev,FR,cspull,2.0.0.dev,2.300000e+01,1.790080e-01,1.757654e-04

I now have to add some code to collect the CPU seconds from the multiple threads.

#14 - 02/20/2022 10:57 AM - Knödlseder Jürgen

- % Done changed from 80 to 90

I added a method GApplication::add_celapse() that allows adding CPU seconds to the result returned by GApplication::celapse(). Using this method in the mputils.process() function allows to collected elapsed CPU times from all threads. This results in a correct results in the log file

Application "cspull" terminated after 23 wall clock seconds, consuming 97.5124 seconds of CPU time and generating a carbon footprint of 0.0957461 g eCO2.

as well as in the statistics.csv file:

Date,GammaLib version,Country,Application,Version,Wall clock seconds,CPU seconds,g eCO2
2022-02-20T09:54:44,2.0.0.dev,FR,cspull,2.0.0.dev,2.300000e+01,9.751393e+01,9.574649e-02

It looks like the problem was fixed.

As a side note, it seems that multi-threading increases the CPU time and carbon footprint, maybe due to the additional computational overhead.

#15 - 02/20/2022 04:03 PM - Knödlseder Jürgen

I finally implemented an initial version of the csfootprint script. For the moment it just produces a text dump, addition of nice matplotlib graphics is still needed. For that I need to collect a bit of usage statistics over the week to get some first data to play with.

#16 - 02/21/2022 08:55 AM - Knödlseder Jürgen

In integrating the current code, the continuous Mac OS X release gets stuck at the following point

Test COMPTEL class pickeling: ..... ok
Test GSPIEventCube class: ..... ok
Test GSPIObservation class: ..... ok
Test INTEGRAL/SPI class pickeling: ..... ok

This should be followed by

make[1]: Leaving directory `/home/jenkins/jenkins/workspace/gammalib-integrate-os/label/centos6_64/test'
make[1]: Entering directory `/home/jenkins/jenkins/workspace/gammalib-integrate-os/label/centos6_64'
make[1]: Nothing to be done for `installcheck-am'.
make[1]: Leaving directory `/home/jenkins/jenkins/workspace/gammalib-integrate-os/label/centos6_64'

yet it is not. Maybe this is related to the tight disk space on the Mac OS continuous integration server. I will for the time being offload the Mac OS 10.8 machine, since it is rather old and not used by any other pipeline. Hope that this fixes the issue.

#17 - 02/21/2022 12:16 PM - Knödlseder Jürgen

On kepler the daemon did update the file with a single line, yet after that it became defunct and the lock file disappeared. Maybe there is some issue with a single line .gamma/statistics.csv file. I need to investigate that. Furthermore code should be added that puts all activities into a try-catch block, writing out any exception in the log file. This will help further tracking of issues.

#18 - 02/21/2022 05:28 PM - Knödlseder Jürgen

- File multiple-daemons.png added

On kepler starting a process apparently starts multiple daemons. Note that all daemons were started by the same process, so for some reason the daemon was not considered alive. I need to check whether this can arise from multi-threading, since the underlying Python script is cspull, hence uses multi-threading.

```
1 S   600   552 13687  0  80   0 - 994699 hrtime pts/11  00:00:00 gammalibd
1 S   600   602 13687  0  80   0 - 1005274 hrtime pts/11 00:00:00 gammalibd
1 S   600   657 13687  0  80   0 - 1005274 hrtime pts/11 00:00:00 gammalibd
1 Z   600 13759 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600 22811 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600 33983 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 S   600 48319 48262  0  80   0 - 73590 hrtime pts/12   00:00:00 gammalibd
1 S   600 49097 13687  0  80   0 - 999114 hrtime pts/11  00:00:00 gammalibd
(ctools-1.7.4) [knodlseder@kepler pulls]$
```

It turned out that the daemons become defunct without any obvious reason, removing the log file. Not clear how this can happen?

```
[knodlseder@kepler test]$ ps -lA | grep gammalib
1 Z   600   552 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600   602 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600   657 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 S   600  2010 48262  0  80   0 - 966110 hrtime pts/12  00:00:00 gammalibd
1 Z   600 13759 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600 22811 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600 33983 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600 48319 48262  0  80   0 -     0 do_exi pts/12   00:00:00 gammalibd <defunct>
1 Z   600 49097 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
[knodlseder@kepler test]$ ps -lA | grep gammalib
1 Z   600   552 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600   602 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600   657 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600  2010 48262  0  80   0 -     0 do_exi pts/12   00:00:00 gammalibd <defunct>
1 Z   600 13759 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600 22811 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600 33983 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600 48319 48262  0  80   0 -     0 do_exi pts/12   00:00:00 gammalibd <defunct>
1 Z   600 49097 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
[knodlseder@kepler test]$ ps -lA | grep 48262
1 Z   600  2010 48262  0  80   0 -     0 do_exi pts/12   00:00:00 gammalibd <defunct>
0 R   600 48262 47800 99  80   0 - 966170 -      pts/12   01:36:32 python
1 Z   600 48319 48262  0  80   0 -     0 do_exi pts/12   00:00:00 gammalibd <defunct>
[knodlseder@kepler test]$ ps -lA | grep gammalib
1 Z   600   552 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600   602 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600   657 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600  2010 48262  0  80   0 -     0 do_exi pts/12   00:00:00 gammalibd <defunct>
1 Z   600 13759 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600 22811 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600 33983 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600 48319 48262  0  80   0 -     0 do_exi pts/12   00:00:00 gammalibd <defunct>
1 Z   600 49097 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
```

#19 - 02/21/2022 06:00 PM - Knödlseder Jürgen

I emulated the code

```
bool GDaemon::alive(void) const
{
    // Initialise flag
    bool alive = false;

    // Get process ID in lock file
    pid_t pid = lock_pid();

    // If process ID is positive then check if process is still alive
    if (pid > 0) {

        // Check if process is alive
        if (0 == kill(pid, 0)) {
            alive = true;
        }

    } // endif: process ID was positive

    // Return flag
    return alive;
}
```

in a little C program and it turns out that defunct processes on Linux are still considered being alive:

```
$ ps -lA | grep gammalib
1 Z   600   552 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600   602 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600   657 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600  2010 48262  0  80   0 -     0 do_exi pts/12   00:00:00 gammalibd <defunct>
1 Z   600 13759 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600 22811 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600 33983 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600 48319 48262  0  80   0 -     0 do_exi pts/12   00:00:00 gammalibd <defunct>
1 Z   600 49097 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
[knodlseder@kepler test]$ ./daemon_alive
Result of kill: 0
```

This appears to be a known problem:

- https://stackoverflow.com/questions/61760834/kill-does-always-return-0-sucess-even-after-child-process-has-already-ended
- https://stackoverflow.com/questions/6898337/determine-programmatically-if-a-program-is-running

I need to fix the daemon handling on Linux.

Just as a side note, four new daemons just started without any intervention from my side:

```
$ ps -lA | grep gammalib
1 Z   600   552 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600   602 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600   657 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600  2010 48262  0  80   0 -     0 do_exi pts/12   00:00:00 gammalibd <defunct>
1 S   600  3552 13687  0  80   0 - 982730 hrtime pts/11  00:00:00 gammalibd
1 S   600  3865 13687  0  80   0 - 978188 hrtime pts/11  00:00:00 gammalibd
1 S   600  3877 13687  0  80   0 - 979049 hrtime pts/11  00:00:00 gammalibd
1 S   600  3915 13687  0  80   0 - 979049 hrtime pts/11  00:00:00 gammalibd
1 Z   600 13759 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600 22811 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600 33983 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600 48319 48262  0  80   0 -     0 do_exi pts/12   00:00:00 gammalibd <defunct>
1 Z   600 49097 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
```

And as another side note, the zombie state of a process can be seen in

```
$ more /proc/2010/status
Name:   gammalibd
State:  Z (zombie)
```

while a running (or better sleeping process) gives

```
$ more /proc/3552/status
Name:   gammalibd
Umask:  0022
State:  S (sleeping)
```

But the /proc directory does not exists on Mac OS.

#20 - 02/21/2022 06:03 PM - Knödlseder Jürgen

Even stranger, the daemon.lock file does not correspond to an existing daemon:

```
$ ps -lA | grep gammalib
1 Z   600   552 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600   602 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600   657 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600  2010 48262  0  80   0 -     0 do_exi pts/12   00:00:00 gammalibd <defunct>
1 S   600  3552 13687  0  80   0 - 982730 hrtime pts/11  00:00:00 gammalibd
1 S   600  3865 13687  0  80   0 - 978188 hrtime pts/11  00:00:00 gammalibd
1 S   600  3877 13687  0  80   0 - 979049 hrtime pts/11  00:00:00 gammalibd
1 S   600  3915 13687  0  80   0 - 979049 hrtime pts/11  00:00:00 gammalibd
1 Z   600 13759 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600 22811 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600 33983 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
1 Z   600 48319 48262  0  80   0 -     0 do_exi pts/12   00:00:00 gammalibd <defunct>
1 Z   600 49097 13687  0  80   0 -     0 do_exi pts/11   00:00:00 gammalibd <defunct>
[knodlseder@kepler test]$ more ~/.gamma/daemon.lock
48130
[knodlseder@kepler test]$ ps -lA | grep 48130
```

#21 - 02/22/2022 04:06 PM - Knödlseder Jürgen

Digging into the problem on kepler I found out that there was still a problem with the carbon accounting and the identification whether an application was a child or a parent application.

I therefore implemented a new scheme which relies on a static counter running() in GApplication, and only when running()<1 an application is considered as a parent, hence the corresponding instance is writing statistics data.

running() is incremented in the ctool::run() at the beginning and decremented at the end, which means that only calls to ctool::run() will change the static counter. If now another instance of GApplication is created within the method, it will recognise that a parent instance is already running, and it will not write statistics data.

To avoid implementation of the logic at the level of individual tools, I remanded the virtual abstract ctool::run() method to ctool::process() and implemented a non-abstract ctool::run() method that handles the static counter. Now all ctools and cscripts implement the ctool::process() instead of the ctool::run() method.

I quickly checked that the logic works. Working still needs to be confirmed using tests on kepler.

#22 - 02/22/2022 04:24 PM - Knödlseder Jürgen

Since yesterday evening I have the weird problem that the daemon gets killed after a file was transferred via scp from my Mac to kepler. I should notice that the daemon was before started via a ssh terminal, yet this also happens to a daemon started for example from a different terminal by calling cscaldb. The same problem occurs when a ssh connection is established to kepler. The problem occurs if the daemon is started using a ctool or a cscript.

This is a huge mystery. Why does scp and ssh kill a process?

#23 - 02/22/2022 04:41 PM - Knödlseder Jürgen

- File image.png added

I found the problem. The process is killed when calling source $GAMMALIB/bin/gammalib-init.sh since there is the following code:

```
#
# Kill a running GammaLib daemon to make sure than a fresh one is started
# ======================================================================
if [ -f $HOME/.gamma/daemon.lock ]; then
  kill -9 `cat $HOME/.gamma/daemon.lock` > /dev/null 2>&1
  rm -rf $HOME/.gamma/daemon.lock > /dev/null 2>&1
fi
```

This is apparently a very bad idea tongue.png

The problem is now fixed!

#24 - 02/22/2022 04:49 PM - Knödlseder Jürgen

- File deleted (image.png)

#25 - 02/24/2022 10:08 AM - Knödlseder Jürgen

I added a gammalibd executable to the src/support directory that emulates the GApplication::start_daemon() method to launch the GammaLib daemon. In that way the daemon can be launched independently of any application, and in addition has the correct process name on Mac OS:

```
$ gammalibd
$ more daemon.lock
75780
$ ps -lA | grep 75780
  501 75780     1       4   0  31  0  4305516   1972 -      S                0 ??         0:00.00 gammalibd
```

#26 - 02/24/2022 02:07 PM - Knödlseder Jürgen

I tried to implement also a logic that launches a daemon before running make check, yet I encountered an issue that occures when make is invoked for parallel compilation, e.g. make -j100. In fact the make step just hangs. The issue may be related to the problem described here https://bugzilla.redhat.com/show_bug.cgi?id=654822.

#27 - 02/24/2022 03:13 PM - Knödlseder Jürgen

I finally managed to make it work. Still, the thing that hangs is a make -j10 check, so this should be avoided if possible. Not very satisfactory, but maybe no one is actually doing these kind of things.

#28 - 02/24/2022 03:45 PM - Knödlseder Jürgen

Since the daemon is now launched by the GammaLib initialisation script there is no longer a need to launch it from an GApplication. I therefore disabled this functionality which gets rid of the make -j10 check problem and also any other race conditions that may occur. Launching the daemon is now much cleaner.

#29 - 02/24/2022 05:34 PM - Knödlseder Jürgen

I now have an issue on Ubuntu 12 during the make installcheck step which hangs. When I type ctrl-C I get the following output. So the issues is that some processes are unfinished:

```
nohup: redirecting stderr to stdout
^Cmake[1]: *** wait: No child processes.  Stop.
make[1]: *** Waiting for unfinished jobs....
make[1]: *** wait: No child processes.  Stop.
make: *** wait: No child processes.  Stop.
make: *** Waiting for unfinished jobs....
make: *** wait: No child processes.  Stop.
```

Note that these hangs are related to the ones encountered in #3998.

#33 - 02/24/2022 05:53 PM - Knödlseder Jürgen

I added some code that prevents that a daemon is launched during make installcheck. No the daemon should never be launched during some make activity, hopefully this solves any issues.

#34 - 02/25/2022 03:53 PM - Knödlseder Jürgen

- Status changed from In Progress to Closed

- % Done changed from 90 to 100

Code merged into devel.

#35 - 02/26/2022 07:34 AM - Knödlseder Jürgen

- Status changed from Closed to In Progress

- % Done changed from 100 to 80

I turned out that on a certain number of OS platforms, such as Debian, the GammaLib installation script hangs. Here for example the outcome on Debian:

```
jenkins@CI02:~$ export GAMMALIB=$HOME/jenkins/install/integrate/gammalib
jenkins@CI02:~$ . $GAMMALIB/bin/gammalib-init.sh
nohup: redirecting stderr to stdout
```

The command hangs after the nohup step. It is impossible to kill the hang with ctrl-C or ctrl-Z.

Calling directly $GAMMALIB/bin/gammalibd on the Debian does not pose any problem. Calling the following also works:

```
jenkins@CI02:~$ $GAMMALIB/bin/gammalibd
jenkins@CI02:~$ nohup $GAMMALIB/bin/gammalibd
nohup: ignoring input and appending output to `nohup.out'
jenkins@CI02:~$ nohup $GAMMALIB/bin/gammalibd &>/dev/null
jenkins@CI02:~$ nohup $GAMMALIB/bin/gammalibd </dev/null &>/dev/null
jenkins@CI02:~$ nohup $GAMMALIB/bin/gammalibd </dev/null &>/dev/null &
[1] 32604
[1]+  Done                    nohup $GAMMALIB/bin/gammalibd < /dev/null &>/dev/null
```

#36 - 02/27/2022 01:26 PM - Knödlseder Jürgen

On Ubuntu the following command, executed by the Jenkins ctools integration step, still hangs:

```
. /home/jenkins/jenkins/install/integrate/gammalib/bin/gammalib-init.sh
```

Changing the code in gammalib-setup as follows does also not help

```
gammalibd 0<&- &>/dev/null &
```

Adding nohup just hangs with

```
nohup: redirecting stderr to stdout
```

Note that the daemon is actually started, yet the call to the process never terminates.
Changing the code in gammalib-setup as follows does also not help

```
setsid gammalibd 0<&- &>/dev/null &
```

#37 - 02/28/2022 10:00 AM - Knödlseder Jürgen

Calling directly

```
. /home/jenkins/jenkins/install/integrate/gammalib/bin/gammalib-setup sh
/home/jenkins/gammalib-config-CI162931.sh
Start gammalibd
Started
$
```

works (note that I added some echo for debugging).

It turns out that the following line in the gammalib-init.sh script produces the hang:

```
gammalib_init=`$GAMMALIB/bin/gammalib-setup sh`
```

#38 - 02/28/2022 10:03 AM - Knödlseder Jürgen

I fixed the script. Initially I had added the daemon launch at the end of gammalib-setup, yet this script should actually not launch anything. The script that should launch the daemon is gammalib-init.sh or gammalib-init.csh, which means that gammalib-setup needs to write the appropriate command to the temporary configuration script that is created by gammalib-setup.

This works, except for Mac OS 10.13 - 10.15 which fail after the test of the configuration as follows:

```
+ ./testconf.sh
Configuration test successful.
FATAL: Remote call on Mac OS X 10.13 failed
java.io.IOException: Remote call on Mac OS X 10.13 failed
    at hudson.remoting.Channel.call(Channel.java:786)
    at hudson.Launcher$RemoteLauncher.kill(Launcher.java:954)
    at hudson.model.AbstractBuild$AbstractBuildExecution.run(AbstractBuild.java:543)
    at hudson.model.Run.execute(Run.java:1741)
    at hudson.matrix.MatrixRun.run(MatrixRun.java:146)
    at hudson.model.ResourceController.execute(ResourceController.java:98)
    at hudson.model.Executor.run(Executor.java:410)
Caused by: java.lang.NoClassDefFoundError: Could not initialize class hudson.util.ProcessTree$UnixReflection
    at hudson.util.ProcessTree$UnixProcess.kill(ProcessTree.java:571)
    at hudson.util.ProcessTree$UnixProcess.killRecursively(ProcessTree.java:592)
    at hudson.util.ProcessTree$Unix.killAll(ProcessTree.java:513)
    at hudson.Launcher$RemoteLauncher$KillTask.call(Launcher.java:966)
    at hudson.Launcher$RemoteLauncher$KillTask.call(Launcher.java:957)
    at hudson.remoting.UserRequest.perform(UserRequest.java:121)
    at hudson.remoting.UserRequest.perform(UserRequest.java:49)
    at hudson.remoting.Request$2.run(Request.java:326)
    at hudson.remoting.InterceptingExecutorService$1.call(InterceptingExecutorService.java:68)
    at java.util.concurrent.FutureTask.run(FutureTask.java:264)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1167)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:641)
    at java.lang.Thread.run(Thread.java:844)
    at ......remote call to Mac OS X 10.13(Native Method)
    at hudson.remoting.Channel.attachCallSiteStackTrace(Channel.java:1413)
    at hudson.remoting.UserResponse.retrieve(UserRequest.java:221)
    at hudson.remoting.Channel.call(Channel.java:778)
    ... 6 more
```

#39 - 02/28/2022 11:30 AM - Knödlseder Jürgen

I rebooted the Mac OS server to see whether this fixes the Jenkins issue.

#40 - 02/28/2022 01:59 PM - Knödlseder Jürgen

Reboot did not fix the issue. The issue may be related to the Java version, see https://issues.jenkins.io/browse/JENKINS-56046.

Here the Java versions for the different Mac OS VMs, as determined by java -version

Mac OS	Java
10.7	1.8.0_66
10.9	1.8.0_66
10.10	1.8.0_66
10.11	1.8.0_65
10.12	1.8.0_65
10.13	9.0.1
10.14	14.0.2
10.15	14.0.2
10.16	1.8.0_261

Since Java 1.8 is also installed on Mac OS 10.13 I added the following a .bashrc file to the jenkins home directory on Mac OS 10.13 with the following line:

export JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk1.8.0_151.jdk/Contents/Home

Restarting the node enabled Java 1.8.

#41 - 03/01/2022 07:47 PM - Knödlseder Jürgen

There is now some hang in the continuous Mac OS X release job:

```
+ make pkgcheck
Check Mac OS X installation image
...
Test INTEGRAL/SPI class pickeling: ..... ok
```

```
dev/pkgcheck-macosx.sh: line 94: 47453 Trace/BPT trap: 5       nohup gammalibd < /dev/null >&/dev/null
...
Test unbinned pipeline with FITS file saving: . ok
Test unbinned in-memory pipeline: . ok
```

So

```
python -c 'import gammalib; gammalib.test()' | tee -a $LOGFILE
```

seems to pass now, although there is a Trace/BPT trap, while the job hangs at

```
python -c 'import ctools; ctools.test()' | tee -a $LOGFILE
```

It turned our that the gammalibd daemon is still running on the Mac OS VM.

#42 - 03/01/2022 10:38 PM - Knödlseder Jürgen

I removed the nohup from the GammaLib daemon launcher.

#43 - 03/02/2022 09:32 AM - Knödlseder Jürgen

- Status changed from In Progress to Feedback

After removing nohup I still get

```
dev/pkgcheck-macosx.sh: line 94:  8669 Trace/BPT trap: 5       gammalibd < /dev/null >&/dev/null
```

yet the Jenkins job terminates with success. I put the issue now on feedback.

#44 - 03/14/2022 12:21 PM - Knödlseder Jürgen

- Status changed from Feedback to Closed

- % Done changed from 80 to 100

Things seem to work now. Close the issue.

Files

GES-h-coeur-GRICAD-2020.pdf	666 KB	04/02/2021	Knödlseder Jürgen

cspull-statistics.xlsx	54.1 KB	02/20/2022	Knödlseder Jürgen
cspull-nthreads1-statistics.xlsx	53.8 KB	02/20/2022	Knödlseder Jürgen
multiple-daemons.png	15 KB	02/21/2022	Knödlseder Jürgen