ctools - Feature #1508

Allow to run analysis via config file

07/09/2015 10:35 AM - Mayer Michael

Status:	Closed	**Start date:**	04/18/2016
Priority:	Low	**Due date:**	
Assigned To:	Knödlseder Jürgen	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	1.1.0		

Description

It came up during the coding sprint that running an analysis using a simple config file as input should be supported.
I guess a simple csrun or csrunconf should be sufficient to achieve that.
The obstacle is more about the config file itself. There are packages to support config file parsing form python (e.g. configobj). But I understand this is not compliant with the gammalib philosophy. Should we develop an own config reader for such a purpose?

History

#1 - 07/09/2015 11:07 AM - Knödlseder Jürgen

I was wondering whether a config file is in fact different from a parfile?

Could we not use a simple high-level parfile to steer a pipeline?

The pipeline would then just be another cscript.

#2 - 07/09/2015 11:20 AM - Mayer Michael

I agree. That in fact make sense. However, the config- or pfile would have to use some parameters multiple times with different values.
We would need to create sth like sections in the pfile:

```
[ctselect]
 * inobs = obs.xml
 * usepnt = no
 * outobs = sel_obs.xml
 * ...

[ctlike]
 * inobs (this should be possible to be set to "ctselect outobs" or "sel_obs.xml"
 * ...
```

Is such a handling possible in pfiles?

#3 - 07/09/2015 11:36 AM - Knödlseder Jürgen

I'm not sure what you mean by using parameters multiple times with different values.

I would expect that a pipeline cscript has high-level parameters and that the script then derives from those parameters for each ctool and handles them consistently (for example piping an output of one tool towards and input of another tool).

Maybe we can work out an explicit use case (i.e. example) to see what is needed and how this could be implemented.

#4 - 07/09/2015 01:15 PM - Mayer Michael

Good idea about the use case. I guess what is needed is the following:

The user should be able to create a pipeline using a single config file but having full control over every single parameter (even the hidden ones). In the config file, have a section of general parameters including a list of tools I want to run in order

```
[general]
  commands = ctselect ctlike ctbutterfly csspec cssresmap
  target = Crab
  ra=83.63
  dec=22.01
  inobs = obs.xml
  model = models.xml
  nxpix = 100
  nypix = 100
  nebins = 20
  binsz = 0.02
  edisp = yes
  ...
```

The other tools then get fed from these bunch of general parameters but the user always has the option to individually control every parameter and step:

```
[ctselect]
  ra = general ra
  dec = general dec
  usepnt=no
  inobs = general inobs
  outobs = sel_obs.xml
  ...

[ctlike]
  inobs = ctselect outobs
  inmodel = general model
  outmodel = unbinned_fitted_, general model
  edisp = general edisp
  refit=no
  ...
```

In addition it should also be possible to rerun e.g. ctbutterfly with a different max_iter parameter. Then the user just changes the "command" parameter in the general section and the max_iter parameter in [ctbutterfly].

```
[general]
  command = ctbutterfly
  ...
[ctbutterfly]
  inobs = ctselect outobs
  inmodel = ctlike outmodel
  max_iter = 100
  ...
```

At the end, one would always run:

```
$ csrunconf configfile.conf
```

and the cscript sets up the pipeline according to the config file content.

This whole scheme very much inspired by Rolf's 'rungt' tool for the Fermi Science Tools. It uses the same logic.

The advantage is that the user can create an own pipeline with a config file by passing the files (i.e. their names) from one tool to another. Intermediate products are stored on disk, so individual steps can be redone if needed.

I hope I presented this idea comprehensive enough smile.png

#5 - 07/10/2015 10:08 AM - Mayer Michael

- File config.xml added

Another option which came to my mind is to use the XML interface to pass the config file.
I attached how such a file could possibly look like. No new functionality on the gammalib level would be needed because the I/O is already supported.
Any thoughts?

#6 - 02/20/2016 12:54 PM - Deil Christoph

- Start date deleted (07/09/2015)

I've looked quite a bit at analysis config in the past years, I think having a config-file driven analysis is valuable for many users. Not just beginners ... it's also how I like to run analyses ... declare parameters and configure algorithm in one file, execute one or very few commands.

So here are my 2 cents ...

IMO a pfile isn't a good choice, because it's flat, and having something hierarchichal is better to structure the config file.

XML works, but is IMO a bit hard to read / write ... but since ctools uses it for many other things such as model specs or obs specs, it might be a very good / consistent choice.

Just using Python scripts is IMO very nice. In this approach you'd add few XYZAnalysis classes with a config attribute and a run method that execute the right ctools in the right order with the right options based on config.
Then the end-user would write short Python scripts that create such analysis classes and set all the config objects.
I.e. the main difference to a config file is that you have a few extra lines: the import at the top, creating the object and calling run at the end.
If you don't like that users have to write these extra lines, you can do something like scons: use Python scripts, but have them executed by scons instead of Python directly, which executes some things (like the imports) automatically:
https://en.wikipedia.org/wiki/SCons#Examples

Other good options are INI or YAML.

A few years ago I used INI files via configobj (a single .py file you can bundle):
http://enrico.readthedocs.org/en/latest/configfile.html

By now I like YAML better (more common, more flexible), so a few months ago in Gammapy we decided to use YAML for config files and result output (example: https://gammapy.readthedocs.org/en/latest/tutorials/gammapy-spectrum/index.html#spectral-fitting-with-gammapy-spectrum).
It's also what Matt chose for Fermipy. I think looking at this example how to configure a Fermi analysis makes it clear that a flat pfile doesn't work well, no?
http://fermipy.readthedocs.org/en/latest/quickstart.html
Here's another example how YAML is used for playbooks:
http://docs.ansible.com/ansible/playbooks_intro.html

So to summarise ... if you don't want to write a YAML parser, my recommendation would be: use Python.
(and actually, even if YAML were readily available, just using Python XYZAnalysis high-level classes might still be better).

Looking forward to what you decide on for ctools!

#7 - 02/22/2016 10:32 PM - Knödlseder Jürgen

This discussion shows already that how exactly to run the tools is often a question of taste.

I guess for Fermi (and even within the Fermi collaboration) there have been tens of systems set up that reflect the way how people want to analyze their data. I would argue that there is no "best" way to do the job, so I would leave it to the user to implement the glue code that he/she likes best.

In other words, I won't put such a script as part of the ctools. ctools should be the bricks of a flexible analysis workflow, but how to put things together should be the user's job. But I would put example workflows as Python scripts in the example folder that can be easily modified by a user to fit his/her needs.

#8 - 02/23/2016 08:33 AM - Knödlseder Jürgen

I thought a bit more about this, and I think it's maybe not a bad idea to include a workflow manager in ctools smile.png

I was focusing on the term "config file" and was not happy with introducing yet another user interface. But at the end it's a workflow manager, and this is definitely useful.

The question is: how to implement this. There are several options.

One is to use an existing workflow manager (e.g. http://www.taverna.org.uk/download/workbench/2-5/astronomy/) but 1) I think it's little used by the community and 2) it will introduce a dependency versus an external system. The interesting thing about Taverna is the sharing of workflows with the community. Taverna uses XML to store the workflows, but you don't edit these files, but use a graphical user interface.

I was in the past thinking about implementing a graphical user interface to allow implementing of workflows, and if we would follow this path, we could attempt sharing the same XML format so that we are compatible with Taverna, while not depending on it.

An alternative I had in mind was a cscript code generator that generates in fact a cscript based on some graphically created workflow. The nice thing about this is that you can build up a workflow hierarchically, building super bricks that can the itself be combined in a larger workflow. So a cscript code generator would be another option (graphically or not).

Or we can introduce a new format for defining a workflow, and then implement a cscript that executes this workflow (I think this was your initial proposal, Michael).

Here a bunch of interesting links concerning workflows:

- https://en.wikipedia.org/wiki/Workflow_Management_Coalition
- http://www.wfmc.org/

We could try different solutions during the coding sprint next week and see what the code sprint community likes best at the end.

#9 - 02/24/2016 02:00 AM - Knödlseder Jürgen

- Status changed from New to In Progress

- % Done changed from 0 to 10

To make a proof of principle for an XML-driven analysis workflow I pushed a branch 1508-implement-workflow into the ctools repository. It contains a script csworkflow.py to execute a workflow described by the test/data/workflow.xml file.

The format of the XML file is:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<workflow>
  <actor name="input">
    <output>
      <parameter name="inmodel" value="$CTOOLS/share/models/crab.xml" />
      <parameter name="caldb"   value="prod2" />
      <parameter name="irf"     value="South_50h" />
      <parameter name="edisp"   value="no" />
      <parameter name="deadc"   value="0.95" />
      <parameter name="ra"      value="83.63" />
      <parameter name="dec"     value="22.01" />
      <parameter name="target"  value="Crab" />
      <parameter name="emin"    value="0.1" />
      <parameter name="emax"    value="100.0" />
      <parameter name="tmin"    value="0.0" />
      <parameter name="tmax"    value="1800.0" />
      <parameter name="chatter" value="2" />
      <parameter name="clobber" value="yes" />
      <parameter name="debug"   value="no" />
      <parameter name="mode"    value="ql" />
    </output>
  </actor>
  <actor name="ctobssim" tool="ctobssim">
    <input>
      <parameter name="inobs"   value="NONE" />
      <parameter name="inmodel" value="inmodel" actor="input" />
      <parameter name="caldb"   value="caldb" actor="input" />
      <parameter name="irf"     value="irf"   actor="input" />
      <parameter name="edisp"   value="edisp" actor="input" />
      <parameter name="prefix"  value="sim_events_" />
      <parameter name="seed"    value="1" />
      ...
```

The file contains a single <workflow> element that is composed by a number of <actor> elements. Each <actor> has optional <input> and <output> elements. The syntax of using an output element of an actor as input of another actor is:

```
<parameter name="inmodel" value="inmodel" actor="input" />
```

where value is the name of the parameter of the actor, and actor is the name of the actor.

The csworkflow.py script then analysis the XML file to find out which actor depends of which other actors, and then executes the actors in the order that resolves the dependencies (so far, no parallel processing is implemented in case that actors could be executed simultaneously). It uses the tool attribute to execute the actor, so that for example different ctobssim actors could be defined by using different names, e.g. ctobssim1, ctobssim2, etc.

Note that the input actor has no tool associated and only output parameters. That's the way how "general" parameters are specified. The script can certainly be modified to query for example all input parameters. One could also imagine to have intrinsic functionalities attached to a given tool. Or a general Python command.

It would be interesting to see whether complex workflows can be implemented by this scheme, and what features are missing.

#10 - 04/18/2016 12:03 AM - Knödlseder Jürgen

- Assigned To set to Knödlseder Jürgen

- Target version set to 1.1.0

- Start date set to 04/18/2016

- % Done changed from 10 to 100

I merged in the draft version of the csworkflow script. I also added a unit test.

We still need to gain more experience on that, but at least we have a workflow manager that can be used now.

#11 - 06/03/2016 12:10 AM - Knödlseder Jürgen

- Status changed from In Progress to Closed

Since the workflow manager has been merged in I close the issue now.

Files

config.xml	2.98 KB	07/10/2015	Mayer Michael