

ctools - Feature #1838

Use ctools like a dictionary in Python

08/06/2016 12:48 AM - Knödseder Jürgen

Status:	Closed	Start date:	08/06/2016
Priority:	Normal	Due date:	
Assigned To:	Knödseder Jürgen	% Done:	100%
Category:		Estimated time:	0.00 hour
Target version:	1.2.0		

Description


The setting of parameters of ctools and cscripts is already equivalent in Python to the setting of a dictionary. This logic could be extended so that lists of parameters can be defined in a dictionary and passed to a ctool. The following code may be possible:

```
>>> select = ctools.ctselect()
>>> select.pars({'inobs': 'events.fits', 'ra': 83.63, 'dec': 22.01, 'rad': 3.0, 'emin': 0.1, 'emax': 100.0, 'tmin': 0.0, 'tmax': 100.0})
>>> select.run()
>>> print(select.pars())
{'inobs': 'events.fits', 'rad': 3.0, 'tmin': 0.0, 'ra': 83.63, 'emin': 0.1, 'emax': 100.0, 'tmax': 100.0, 'dec': 22.01}
```

History

#1 - 08/06/2016 02:26 AM - Knödseder Jürgen

- Status changed from New to In Progress
- Assigned To set to Knödseder Jürgen
- Target version set to 1.2.0
- Start date set to 08/06/2016
- % Done changed from 0 to 90

And here it is  This little code does the job (is part of ctool.i):

```
def _pardict(self, *args):
    if len(args) == 0:
        d = {}
        for par in self._pars():
            if par.type() == 'b':
                v = gammalib.tolower(par.current_value())
                value = (v == "yes" or v == "y" or v == "true" or v == "t")
                d[par.name()] = value
            elif par.type() == 'i':
                d[par.name()] = int(par.current_value())
            elif par.type() == 'r':
                d[par.name()] = float(par.current_value())
            else:
                d[par.name()] = str(par.current_value())
        return d
    elif len(args) == 1:
        for key in args[0]:
            self[key] = args[0][key]
    else:
        raise TypeError('pars() takes 0 or 1 arguments (%d given)' % len(args))
ctool.pardict = _pardict
cscript.pardict = _pardict
```

For this code to work I had to add two GammaLib methods:

- `GApplication::pars()` to retrieve the parameters from an application
- `GApplicationPar::current_value()` to retrieve the current parameter values without querying the parameters

Here a little example code that I added to the `ctbin` unit test:

```
# And finally go for a fully Pythonic version with all parameters
# being specified in a dictionary
pars = {'inobs': self._events, 'ebinalg': 'LOG', 'emin': 0.1,
        'emax': 100.0, 'enumbins': 10, 'nxpix': 40, 'nypix': 40,
        'binsz': 0.1, 'coordsys': 'CEL', 'proj': 'CAR',
        'xref': 83.63, 'yref': 22.01, 'outcube': 'ctbin_py4.fits',
        'logfile': 'ctbin_py4.log', 'chatter': 2}
bin = ctools.ctbin()
bin.pardict(pars)
bin.logFileOpen()
bin.execute()
```

#2 - 08/06/2016 02:55 AM - Knödlseider Jürgen

Moved into devel - for now.

I'm still not sure about the name of the method (`pars`). The reason is that `GApplication` has a `pars()` method that returns the parameters, and in Python this method is renamed to `protected_pars()`. But then a `pars()` method is added in Python that gives access to the dictionary. I'm not sure that (almost) the same name should be used for two methods that take or return different types.

I better find some better names `sad.png` `sad.png` `sad.png` `sad.png`

#3 - 08/06/2016 03:04 AM - Knödlseider Jürgen

What about `pardict()` in Python? Makes clear that we expect and get a dictionary.

#4 - 08/08/2016 10:45 AM - Mayer Michael

What about `pardict()` in Python? Makes clear that we expect and get a dictionary.

Yes that sounds clearer.

I like that the following code is now possible:

```
import ctools
mytool = "ctobssim"
tool = ctools.__dict__[mytool]()
for par in tool:
    print(par) # or do sth else
tool.run()
```

This way people could easily add support for simple ascii config files, too.

We should add some example code for this on the documentation as well.

#5 - 08/09/2016 12:15 AM - Knödlseeder Jürgen

- *Status changed from In Progress to Closed*

Merged into devel.

#6 - 08/09/2016 12:19 AM - Knödlseeder Jürgen

user#77 wrote:

What about `pardict()` in Python? Makes clear that we expect and get a dictionary.

Yes that sounds clearer.

I like that the following code is now possible:

[...]

This way people could easily add support for simple ascii config files, too.

We should add some example code for this on the documentation as well.

I tried to run you few lines of code but could not manage to make it work. How should this example work?

#7 - 08/09/2016 10:10 AM - Mayer Michael

I tried to run you few lines of code but could not manage to make it work. How should this example work?

Apologies, I made a typo. The for loop should of course look like this:

```
for par in tool._pars():  
    print(par)
```

I realised that `tool.pars()` doesn't work but we need the underscore - any idea why?

It might even be useful to add the operator[`int index`] to `GApplication` to allow a direct loop over the parameters via

```
for par in tool:
```

What do you think?

#8 - 08/09/2016 10:58 AM - Knödlseeder Jürgen

user#77 wrote:

I tried to run you few lines of code but could not manage to make it work. How should this example work?

Apologies, I made a typo. The for loop should of course look like this:

[...]

I realised that `tool.pars()` doesn't work but we need the underscore - any idea why?

I made all public methods of the `GApplication` base class private for `ctools` (by prepending an underscore), mainly because the code quality checker (pylint) complained about having too many public methods. The reason behind not having too many public methods is maintenance of the public interface.

I'm still a bit hesitant about this public versus private thing in Python. Python has not really private methods since a user can access all methods, but the convention - recognised by pylint - is that prepending an underscore means that a method is private. There are methods that a user probably should not really use (such as the application logger), but I agree that access to the parameter interface is useful for a user.

You can see how this works in `GApplication.i`:

```
// Ignore base class methods and make methods private in Python by  
// prepending an underscore  
%ignore          clear;  
%ignore          clone;  
%ignore          classname;  
%rename(_name)    name;  
%rename(_version) version;  
%rename(_logTerse) logTerse;  
%rename(_logNormal) logNormal;  
%rename(_logExplicit) logExplicit;  
%rename(_logVerbose) logVerbose;  
%rename(_logDebug) logDebug;  
%rename(_clobber) clobber;  
%rename(_has_par) has_par;  
%rename(_par_filename) par_filename;  
%rename(_log_filename) log_filename;
```

```
%rename(_log_header)    log_header;
%rename(_log_trailer)   log_trailer;
%rename(_need_help)     need_help;
%rename(_log)           log;
%rename(_pars)          pars;
```

We could remove `has_har()` and `pars` from the list of renames which would make these methods public (i.e. accessible using `has_par()` and `pars`).

It might even be useful to add the operator `[int index]` to `GApplication` to allow a direct loop over the parameters via `[...]` What do you think?

I agree that this would be useful. Note that to make it work as an iterator you have to add an exception in case that the index is out of range (that's how Python iterators detect the end). An example is in `GModels.i`:

```
GModel* __getitem__(const int& index) {
    if (index >= 0 && index < self->size()) {
        return (*self)[index];
    }
    else {
        throw GException::out_of_range("__getitem__(int)", "Model index",
                                         index, self->size());
    }
}

void __setitem__(const int& index, const GModel& val) {
    if (index >= 0 && index < self->size()) {
        self->set(index, val);
        return;
    }
    else {
        throw GException::out_of_range("__setitem__(int)", "Model index",
                                         index, self->size());
    }
}
```

And we should also add a Python unit test (see #1582, and #1581 for the reason why).

#9 - 08/09/2016 11:42 AM - Mayer Michael

Ok, I can make the change to the GApplication interface.

Concerning the unit tests: I believe a simple loop in Python from start to end should be sufficient. All these tests will thus be in \$GammaLib/test/test_*.py files, right?

#10 - 08/09/2016 11:54 AM - Knödlseeder Jürgen

user#77 wrote:

Ok, I can make the change to the GApplication interface.

Concerning the unit tests: I believe a simple loop in Python from start to end should be sufficient. All these tests will thus be in \$GammaLib/test/test_*.py files, right?

Thanks!

For the unit test you may check test_GSky.py which has a corresponding test for the GSkyMap class:

```
# Access operator (tests also proper iteration)
sum = 0.0
for pix in map:
    sum += pix
self.test_value(sum, 16.0)
```

#11 - 08/09/2016 03:39 PM - Mayer Michael

Modifications are on branch *1838-access-GApplicationPars* (in GammaLib).

Note that I added an iterator test to test_GModels.py as well.

#12 - 08/10/2016 09:47 AM - Knödlseeder Jürgen

- % Done changed from 90 to 100

Merged branch 1838-access-GApplicationPars into devel.