# Charming users into scripting CIAO with Python The ciao\_contrib.runtool module D. J. Burke,

Smithsonian Astrophysical Observatory, 60 Garden Street Cambridge, MA 02138

## Abstract

The Science Data Systems group of the Chandra X-ray Center provides a number of scripts and Python modules that extend the capabilities of CIAO (http://cxc.harvard.edu/ciao/). Experience in converting the existing scripts - written in a variety of languages such as bash, csh/tcsh, Perl and S-Lang - to Python, and conversations with users, led to the development

#### **Parameter files and tools**

The CIAO data analysis system uses parameter files to define the user-interface of tools and scripts that it provides. These parameter files provide the names, types, default values, possible constraints, and help text for the parameters, or arguments, that the tools use. Users can query these files to find out what inputs and outputs a tool uses, find what the current values are, or change the values. Below we show a typical sequence of commands that a user may write at the command line or in a Shell script, and to the right the equivalent calls using the ciao\_contrib.runtool module:

% plist acis\_bkgrnd\_lookup

(blname = none)

infile =

(verbose = 0)
 (mode = ql)

outfile =

Parameters for /Users/dburke/cxcds\_param4/acis\_bkgrnd\_lookup.par

Event file for which you want background files ACIS background file(s) to use What block identifier should be added to the filename? Debug level (0=no debug information)

% pset acis\_bkgrnd\_lookup verbose=1
% pset acis\_bkgrnd\_lookup blname=all
pquery: invalid enumerated value : blname

#### of the ciao\_contrib.runtool module.

This allows users to easily run CIAO tools from Python scripts, and utilizes the metadata provided by the parameter-file system to create an API that provides the flexibility and safety guarantees of the command-line. The module is provided to the user community and is being used within our group to create new scripts, which is described in poster P085 "Python Scripting for CIAO Data Analysis".

# An excerpt from a mythical shell script that filters
# an event file, finds out the matching "blank-sky" background
# file for the resulting file, and then filters that file
#

dmcopy "evt2.fits[sky=region(src.reg),energy=500:7000]" src.fits clobber=yes acis\_bkgrnd\_lookup src.fits verbose=0 set bfile = `pget acis\_bkgrnd\_lookup outfile` dmcopy "\${bfile}[energy=500:7000]" bg.fits What block identifier should be added to the filename? (none|name|number|cfitsio) (none): name

% cat `paccess acis\_bkgrnd\_lookup` infile,f,a,"",,,"Event file for which you want background files" outfile,f,1,"",,,"ACIS background file(s) to use" blname,s,h,"name",none|name|number|cfitsio,, "What block identifier should be added to the filename?" verbose,i,h,1,0,5,"Debug level (0=no debug information)" mode,s,h,"ql",,,

#### # Conversion to Python

from ciao\_contrib.runtool import dmcopy, acis\_bkgrnd\_lookup

- acis\_bkgrnd\_lookup("src.fits", verbose=0)
  bfile = acis\_bkgrnd\_lookup.outfile
  dmcopy("{0}[energy=500:7000]".format(bfile), "bg.fits")

## **Goals of the Python interface**

Prioritize familiarity and simplicity over functionality.
Map tool names to functions (actually objects).
Avoid worries about quoting special characters (CIAO filters can contain "awkward" characters such as []!\*'").
Provide names arguments matching those of the tools.
Type conversion and verification for argument values.
Easy access to screen output of the tool.
Convert tool failures to Python IOError exceptions.
Mimic parameter access via object attributes.
Support running multiple copies of the tool simultaneously.

chips-1> from ciao\_contrib.runtool import acis\_bkgrnd\_lookup chips-2> print(acis\_bkgrnd\_lookup) Parameters for acis\_bkgrnd\_lookup:

#### Required parameters:

infile = Event file for which you want background files

#### Optional parameters:

outfile =ACIS background file(s) to useblname = noneWhat block identifier should be added to the filename?verbose = 0Debug level (0=no debug information)chips-3> abl = acis\_bkgrnd\_lookupchips-4> abl.bln = "foo"ValueError: The parameter blname was set to foo when it must be one of:none name number cfitsionchips-5> abl.blname = "name"chips-6> abl.verbose = 10ValueError: acis\_bkgrnd\_lookup.verbose must be <= 5 but set to 10</td>

chips-7> abl("missing.fits", verb=0) IOError: An error occurred while running 'acis\_bkgrnd\_lookup': # acia\_bkgrnd\_lookup (25 October 2010); EBBOB Upshlate error infile='missing chips-8> for (pname,pval) in abl: print("{0}.{1} -> {2}".format(abl.toolname(), pname, pval))

acis\_bkgrnd\_lookup.infile -> missing.fits acis\_bkgrnd\_lookup.outfile -> None acis\_bkgrnd\_lookup.blname -> name acis\_bkgrnd\_lookup.verbose -> 0 chips-9> help abl Help on CIAOToolParFile in module ciao\_contrib.runtool object:

class CIAOToolParFile(CIAOTool)| Run a CIAO tool using a separate parameter file.

See the help for CIAOTool for information on this class.

Method resolution order: CIAOToolParFile CIAOTool CIAOParameter \_\_builtin\_\_.object

## **Implementation details**

#### The functionality is provided by the following class structure



where the CIAOParameter class supports the reading and writing of parameter values and is used to support the small set of CIAO parameter files which are not associated with an individual tool (configuration or informational files). The CIAOTool class is the basis for callable tools but the actual implementation is left to the CIAOToolParFile and CIAOToolDirect classes. Most tools are handled by the former, with the latter class provided to handle those few tools and scripts which do not support the @@parfile syntax (as discussed in the "Running multiple copies of a tool" section). Actually running a tool is a thin layer around the subprocess module; most of the code goes into validating the parameter values.

The module is created by code generation - based on a parsed view of the CIAO parameter files - rather than having the necessary instances created either when the module is loaded or explicitly by the user. The trade off here was ease, and speed, of use for the user versus a more complicated development environment. Since the functionality is encapsulated within a class

## **Running multiple copies of a tool**

One of the design goals of the module is to allow the user to easily run multiple copies of a tool simultaneously, whether via the multiprocessing module or by repeated runs of the same script. Simultaneous runs of a tool is likely to cause corruption of the parameter file, since each copy is reading and writing the same file, which can lead to invalid output. To avoid this, each copy of the tool is run with its own unique (temporary) parameter file, supplied using the @@parname functionality of the CIAO parameter library, which is removed once the tool has finished.

#### There are two problems with this approach:

1) a small subset of CIAO tools do not support the @@parname syntax; these tools are run with all its parameters set on the command line to reduce the chance of corruption, and

# 2) some tools, in particular those that are wrappers around other tools, require multiple parameter files.

Whilst the CIAO 4.3 release essentially removes the first problem (only the dmgti tool remains in this category), the second problem can only be avoided by explicitly creating separate directories to store the parameter files for each task. This can be achieved using the set\_pfiles() routine provided by the module, which changes the user-directory portion of the PFILES environment variable used by the parameter library.

structure, it would be relatively easy to switch to the run-time approach.

Although CIAO provides a Python module that binds to the parameter library, this interface is low level and does not provide the required functionality - in particular parameter validation without error messages or requiring user interaction - which means that a significant part of the module is essentially replicating the functionality of the parameter library.

Since CIAO tools occasionally use both the stdout and stderr channels to output information, so both channels are combined into one and returned to the caller when the tool finishes. If the tool returns a non-zero exit status the screen output is instead returned to the user as the message payload of an IOError exception. There are several tools which do not set the exit status on certain errors, which will result in the Python routine apparently succeeding.

As the module is intended for use from within a Python, the interactive mode of operation of CIAO tools, where users are prompted for missing or invalid parameter values, is not supported by the module. Attempts to include the tool or parameter information in the Python docstrings for the routines was not successful; once it started to require the use of Python metaclasses the complexities of the interface outweighed the benefits to the user.

### **Future Work**

The main aims of the module have been met, so future development depends on user feed-back. The main areas that have been identified so far are:

extending to support other systems with parameter files, notably the FTOOLS package,
 support for piping tools together to avoid the need to create intermediary files, and
 complete support for the parameter interface.

## **Further Information**

http://cxc.harvard.edu/ciao/scripting/runtool.html

http://cxc.harvard.edu/ciao/ahelp/ciao\_runtool.html

This work was supported by the Chandra X-ray Center under NASA contract NAS8-03060.