

# Python Scripting for CIAO Data Analysis

Elizabeth C. Galle, Craig S. Anderson, Nina Bonaventura, D. J. Burke,  
Antonella Fruscione, Nicholas P. Lee, Jonathan C. McDowell  
Smithsonian Astrophysical Observatory, 60 Garden Street Cambridge, MA 02138

## Abstract

The Chandra X-ray Center has adopted Python as the primary scripting language in the Chandra Interactive Analysis of Observations software package (CIAO, <http://cxc.harvard.edu/ciao/>). Python is a dynamic object-oriented programming language that offers strong support for integration with other languages and tools and comes with extensive standard libraries.

Integrating Python into CIAO allows us to develop powerful new scripts for data analysis, as well as rewrite and improve upon popular CIAO contributed scripts.

We discuss the coding guidelines that we have developed during this process, using specific CIAO contributed scripts—available for download online—as examples.

## The CIAO Contributed Package

The CIAO contributed tarfile contains analysis scripts and modules that automate repetitive tasks and extend the functionality of the CIAO software package by filling specific analysis needs. Many of the scripts were conceived and written by CXC scientists in the early years of the mission, before CIAO became a robust and mature software package. Over the course of ten years, the contributed package evolved into a collection of shell, Perl, S-Lang, and slsh scripts.

Since Python was adopted as the primary scripting language in CIAO, a targeted effort has been made to review the code of these scripts and to rewrite them in Python, bringing them up-to-date and making them easier to maintain over the future of the mission.

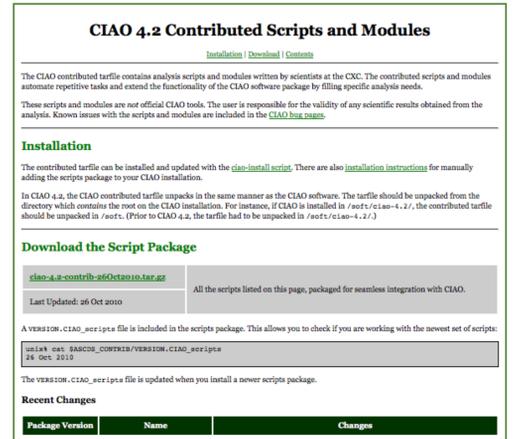


Figure 1. The CIAO Contributed package website <http://cxc.harvard.edu/ciao/download/scripts/>

## Style Guide

The contributed tarfile had been developed in a multiple-author environment without coding standards in place. The scripts spanned a number of languages with varying levels of parameter file support, consistency of warning and error messages, and cross-platform compatibility.

In addition to being written in Python, a baseline requirement was that every script should have a parameter file and an XML help file for used in the CIAO ahelp system.

The next step was to establish style guidelines for the script authors. The standard Python style guide, known as PEP 8 (<http://www.python.org/dev/peps/pep-0008/>), was used as a primary resource. In addition, a scripting style guide was developed to ensure that the scripts were CIAO-like in their appearance and operation.

### VERBOSE LEVELS

The script should default to verbose=0 or 1. If the script doesn't really create any screen output that the user really needs to see then use 0 otherwise 1.

A verbose level of 2 should be considered to be useful for a user tracking what the script has done - so at some level this replaces an explicit log file. So things like listing the parameters used, and what steps are being taken would happen at this level. The `ciao_contrib.runtool` module will print out the command line for each tool (i.e. so the user can see what is actually run).

A verbose level of 3, 4 and 5 are for debugging the tool/script (some of the contributed modules will spew copious amounts of verbiage at level 5).

### HOW TO ACTUALLY WRITE THE MESSAGES

Try using the `ciao_contrib.logger_wrapper` module, which internally uses the Python logging module but hopefully provides a slightly simpler interface for the script writer. By using this module, you can get information from some of the other contributed modules too (e.g. `ciao_contrib.runtool` [see Burke, P084]).

Once the module has been loaded, the `initialize_logger()` routine is used to make sure the messages will get displayed on screen, and the `set_verbosity()` routine sets the verbose level of the tool and any libraries that also use this setup. As shown below, the `make_verbose_level()` routine is used to create a routine that will display a message if the verbosity is a given level or higher. We assume that the variable `toolname` has been defined previously:

```
import ciao_contrib.logger_wrapper as lw
```

```
lw.initialize_logger(toolname)
v1 = make_verbose_level(toolname, 1)
v2 = make_verbose_level(toolname, 2)
```

(you only need to call `make_verbose_level()` for the verbose levels your script uses, and the choice of v1, v2, etc. is up to you).

## A Case Study: Rewriting `acisspec`

The `acisspec` script is a textbook example of a contributed script. It was written by a CXC scientist for his specific analysis needs, but was found to be useful to many users doing imaging spectroscopy. The shell script was added to the contributed package in 2001.

Originally, `acisspec` was designed to:

- extract ACIS PI spectra and associated WMAPs for both extended sources (and background)
- coadd or average two ACIS PI spectra and build weighted responses

The extended source extraction was replaced by the `specextract` tool in CIAO 3.3 (November 2005), but `acisspec` was still required for coadding imaging spectra.

There were two initial goals for the `acisspec` rewrite:

- replicate the coadding and weighting functionality but remove the spectral extraction steps.
- extend the script to be capable of combining N spectra and responses (`acisspec` is restricted to two inputs).

The script, renamed to "`combine_spectra`," was released in August 2010 for use in CIAO 4.2. `combine_spectra` sums multiple imaging source PHA spectra and (optionally) the ARFs, background spectra, and background ARFs.

The Python source code is easier to maintain and update than the shell syntax used for `acisspec`, creating a lighter and faster development cycle. Upcoming enhancements to `combine_spectra` include:

- the addition of the new tool `addresp`—coming in CIAO 4.3 (December 2010)—will allow `combine_spectra` to coadd and weight source and background RMF files as well as ARF files.

- a new parameter which will allow the user to choose between combining imaging spectra via an exposure 'mean' or 'sum'; currently, `combine_spectra` only supports the summed exposure case.

- the ability to combine grating spectra - replacing another contributed shell script, `add_grating_spectra`

As a Python script, `combine_spectra` can also be used as an importable module. The script itself, as well as any functions it contains, can be invoked by other scripts to extend their functionality. The scripts team is evaluating using this modularity to incorporate `combine_spectra` into the spectral extraction tool, `specextract`.

The original `acisspec` tool applied a combination of the UNIX echo and awk commands to the CIAO output, from which the BACKSCALE header values were calculated. In the latest version, we used the mathematical functions from the Python module NumPy (<http://numpy.scipy.org/>) to simplify this operation. (Notice also the improved code commenting in the rewritten code—another point for the style guide.)

```
# get_keyword file keyword
# runs 'dmskeypar file keyword >/dev/null 2>&1'
# and returns the value
# use foo='get_keyword file keyword' to get the value
# into a shell variable
#
get_keyword() {
  dmskeypar $1 $2 >/dev/null 2>&1
  echo `pgrep dmskeypar value`
}
...
backscsoul=`get_keyword ${root}_soul.${ptype} BACKSCAL`
echo `b`
backscsou2=`get_keyword ${root}_sou2.${ptype} BACKSCAL`
if [ "$bgfile1" != "NONE" ]; then
  echo `c`
  backscbgd1=`get_keyword ${root}_bgd1.${ptype} BACKSCAL`
  echo `d`
  backscbgd2=`get_keyword ${root}_bgd2.${ptype} BACKSCAL`
# echo "$backscsoul, $backscsou2, $backscbgd1, $backscbgd2"
backscbgd=`echo "$expsou $expsou1 $backscsoul $backscbgd1 $expsou2
$backscsou2 $backscbgd2" | awk '{ printf "%.6e", $1/($2*($3/$4) + $5*($6/$7)); }'`
# echo `e`
# echo "$backscbgd"
f1
backscsou1.0
if [ "$bgfile1" != "NONE" ]; then
  f1=`echo "$backscbgd $expsou $expsou1 $expsou2 $expsou3 $expsou4 $backscsoul
$backscbgd1" | awk '{ printf "%.6e", $1*(($2/$3)*($4/$5)*($6/$7)); }'`
  f2=`echo "$backscbgd $expsou $expsou1 $expsou2 $expsou3 $expsou4 $backscsoul
$backscbgd2" | awk '{ printf "%.6e", $1*(($2/$3)*($4/$5)*($6/$7)); }'`
  f2=`printf '%7.5f \n' $f2`
```

Figure 2. code sample from `acisspec` The backscale value calculations are done with a combination of Unix tools.

```
#####
# Define the total source and background exposure values, the HEASARC
# BACKSCALE value for the combined background spectrum, and the
# coefficients
# used to scale the background spectra before combining.
#
# Reference: http://heasarc.gsfc.nasa.gov/docs/asca/abc_backscal.html
#
...
#####
srpcha = numpy.array(spha_strarray)
expsrc = expsrc_array
totexpsrc = sum(expsrc_array)
backscalerc = backsrc_array
cbkscalerc = 1.0 # backscal value for combined source spectrum
if bkg != "NULL":
  bkgpha = numpy.array(bpha_strarray)
  expbkg = expbkg_array
  totexpbkg = sum(expbkg_array)
  backscalbkg = backbkg_array
  bkg_coadd_factor = expsrc*(backscalerc/backscalbkg)
  cbkscalbkg = totexpsrc/sum(bkg_coadd_factor) # backscal value for
  #combined background
  f = range(fcount) # initialize the array of background
  # scaling coefficients to be computed below
  # (also referred to as 'c' in HEASARC docs)
for i in fc:
  ...
  if bkg != "NULL":
    f[i] =
    cbkscalbkg*(totexpbkg/totexpsrc)*(expsrc[i]/expbkg[i])*(backscalerc[i]/bac
    kscalbkg[i])
```

Figure 3. code sample from `combine_spectra` The NumPy module provides easy access to mathematical functions.

## The `ciao_contrib.runtool` Module

The `ciao_contrib.runtool` module allows CIAO tools to be run as if they were Python functions, and supports a `pset`-like mode where tool parameters can be set before the tool is run, named parameters when calling the tool, and easy access to the parameter values after the tool has completed.

The easy access to and handling of CIAO tools provided by the `ciao_contrib.runtool`, `excdm_wrapper`, and `param_wrapper` modules, accessing header and table data from FITS files, and writing data to headers and tables of output FITS files are used extensively in `combine_spectra`.

Poster P084 - "Charming users into scripting CIAO with Python - the `ciao_contrib.runtool` module" - discusses the `runtool` module in detail.