URL: http://cxc.harvard.edu/ciao3.4/parameter.html
Last modified: December 2006

AHELP for CIAO 3.4

parameter

Context: concept

Synopsis

Describes the parameter interface used by CIAO.

Description

The CIAO tools use ASCII parameter files to get and store processing parameters. This interface is similar to the IRAF and FTOOLs systems and includes all their features. It provides great flexibility in specifying parameters to programs, since their values can be obtained either from the command line or from a parameter file.

Setting parameters in a GUI and from S−Lang

Parameters for a tool can also be set from a GUI using the peg tool (see "ahelp peg"), and a S−Lang interface to the parameter interface has been made available (see "ahelp paramio").

Location of parameter files

When a tool is run for the first time, it copies its parameter file from the system directory into a user's local directory. As described below the location of these files is controlled by environment variables which are set to default values when you start CIAO.

Whenever the tools are run thereafter, the version in the user's local directory is used. The environment variables that control where to search for the parameter files are (in search order) PDIRS, PFILES, and UPARM. The path listed before the ";" is the user writeable location where updated parameter files are kept. The ":" separated list of directories that follow shows where to search for the system default files.

Throughout the rest of this document we shall use PFILES to mean any of PDIRS, PFILES, or UPARM. We strongly suggest that people use the PFILES envrionment variable in order to reduce conflicts with the FTOOLS package. After setting up the CIAO software, your PFILES variable should look something like:

```
unix% echo $PFILES
/home/username/cxcds_param;/soft/ciao/param
```

If you have started up FTOOLS before CIAO, then it would look something like:

```
unix% echo $PFILES
/home/username/cxcds_param:/home/username/pfiles;/soft/ciao/param:
/soft/lheasoft/SunOS_5.7_sparc/syspfiles
```

(here we have used version 5 of FTOOLS). The "Running multiple copies of a tool" section at the end of this help file suggests different ways to allow you to run multiple copies of the same tool at the same time.

Using parameter files

The paccess command−line tool can be used to find out the location of a tool's parameter file:

```
unix% paccess dmlist
/home/username/cxcds_param/dmlist.par
```

and the plist command can be used to list its contents:

```
unix% plist dmlist

Parameters for /home/dburke/cxcds_paramx/dmlist.par

      infile =                  Input dataset/block specification
         opt = data             Option
    (outfile = )                Output file (optional)
       (rows = )                Range of table rows to print (min:max)
      (cells = )                Range of array indices to print (min:max)
    (verbose = 0)               Debug Level(0−5)
       (mode = ql)
```

The output of the plist command shows you the order the parameters are asked for, the current value of the parameter − if any − and the prompt for that parameter. Those parameters surrounded by "()", such as outfile above, are so−called hidden parameters. These parameters will not be queried for when the tool is run, so if you wish to change their values you need to either pset them before running the tool, or include them on the command line.

The above output tells us that if dmcopy is run with no arguments, the user will be prompted for the infile and opt parameters, as shown below:

```
unix% dmlist
Input dataset/block specification (): in.fits
Option (data): blocks
```

These parameters could also have been included on the command line or set via the pset tool.

```
unix% pset dmlist opt=blocks
unix% dmlist in.fits
Option (blocks):
```

Since the infile parameter was specified on the command line, the tool did not need to ask for its value. It still asks for the opt parameter, although the default value has now changed to "blocks" because of the call to pset.

Setting parameter values

A number of features are available when setting parameters, either via calls to pset or when calling the tool. Many of these are used in the examples below as well as described in the following table (since the features may use characters that have a meaning to the shell, it is often safest to surround the parameter value with quotes).

Feature	Description
Shortening parameter names	When specifying parameter values by name, you do not need to specify the full parameter name, as long as it uniquely identifies that parameter. So, for dmextract you can use "i=in.fits" to set infile to in.fits but not "bkg=gaussian" since there is more than one parameter beginning with

	"bkg".
Setting boolean parameters	As a shortcut, boolean parameters can be set to yes using name+ and set to no using name−. So, for dmextract clobber can be set to yes using cl+ and to no using cl− (here we are also takes advantage of the fact that "cl" uniquely indentifies the clobber parameter for dmextract).
${name}	If a parameter value contains the string ${name} and name is an environment variable, then the environment variable will be expanded when the parameter is queried by the tool.
)paramname	The value for one parameter can be re−directed to use the value for another parameter of the same tool using the ")paramname" syntax.
)tool.paramname	This is similar to the previous re−direction except that the value is taken from the parameter of a different tool.
))command	Here the parameter value is found by executing the given command and using the value it returns (i.e. the standard output of the command).
%xpa(xpa_target,xpa_access_point)	Here, the specified xpa_target is queried for the value of xpa_access_point, and the parameter is set to the returned value. Generally, xpa_target will be a CIAO GUI or ds9, although any XPA−enabled application is possible.

Note that while the ${name} and %xpa(...) forms can appear multiple times in a single parameter value, the) and)) forms can appear only once and must comprise the entire parameter value. Hence,

```
dmlist "%xpa(prism,file)[cols %xpa(prism,selected cols)]"
```

is okay, but

```
detsys="ACIS−)dmcoords.chip"
```

is not. Also, mixing different forms of indirection, as in

```
infile="${MY_OBSID}/primary/evt2.fits[sky=%xpa(ds9,region)]"
```

may not always work as expected and should be used with caution.

For "interpreted" parameter values − such as those that contain environment variables, those that are re−directed to another value, and those set to the output of a command − the parameter is evaluated when the tool is run, which need not be the time when the parameter was set. If you use pget then the expression will be evaluated and the value printed out; if you use plist you will see both the expression and the current value, separated by " −> ".

The following example illustrates some of this by setting the sigma parameter equal to that of the median parameter, and then changing the value of the median parameter.

```
unix% punlearn dmstat
unix% pset dmstat sigma=")median"
unix% plist dmstat

Parameters for /home/username/cxcds_param/dmstat.par

        infile =                  Input file specification
     (centroid = yes)             Calculate centroid if image?
       (median = no)              Calculate median value?
        (sigma = )median −> no)   Calculate the population standard deviation?
         (clip = no)              Calculate stats using sigma clipping?
       (nsigma = 3)               Number of sigma to clip
      (maxiter = 20)              Maximum number of iterations
         (mode = ql)
```

```
unix% pget dmstat sigma
no
unix% pset dmstat median+
unix% pget dmstat sigma
yes
```

Features available at a parameter prompt

When prompted for a parameter, there are several special features that can be invoked:

Feature	Description
Tab completion	This is similar to tcsh's tab completion functionality. If the input string is a path, hitting the TAB key will list all the filenames that match. If only one matches, it will fill in the parameter string.
History	Using the UP and DOWN arrow keys, you can cycle through the previous data values for the current parameter and the parameters queried for earlier. If the value being entered has enumerated values, these are also cycled through.
Help	If the user enters "?" as a parameter value, the help file for that parameter will be displayed. The user can also enter "??name" to get help for a different subject (e.g. you may want to try "??stack", "??autoname", or "??dmsyntax").

Stacks and autonames

The ahelp file for a tool also lists the parameters for it, along with information on whether the parameter works with autoname−ing or whether it accepts stacks. See "ahelp autoname" and "ahelp stack" for more information on these features, as well as the examples below.

The mode parameter

Each parameter in a parameter file has a mode setting, as discussed in the "Description of a parameter file" section below. This mode − which has a value of "a", "h", "l", "q", or "ql" − is used to determine whether the parameter is prompted for when a tool is run. Automatic parameters − those with a mode of "a" − are prompted for when a tool is run, and are indicated as such in the output of plist by not having "()" around them. The values of the other parameters are taken from the command line or parameter file.

The behaviour of querying automatic parameters can be over−ridden by setting the mode parameter of the tool (if it does not have one listed in its parameter file then the parameter library assumes a value of "ql"). If the mode parameter is set to "h" then this over−rides the individual settings of the parameters and no prompting will occur; in this case the parameter values will be taken from the command line or parameter file.

One way this can come in useful is if you are running a tool repeatedly with most parameters being the same. You can use pset to set the parameters that do not change and then call the tool with the parameters that do change followed by "mode=h".

Example 1

```
punlearn dmcopy
dmcopy
Input dataset/block specification (): in.fits
Output dataset name (): out.fits
```

If a tool (in this example we use dmcopy) is run without any command−line options then each parameter that the tool needs will be read from the user's copy of the parameter file, then all the automatic parameters will be

prompted for (with defaults taken from the parameter file). If the user does not have a copy of the parameter file, one will be copied to the user's work directory (the location is controlled by the PFILES environment variable).

The paccess command can be used to find out where the parameter file for a given tool is, and the plist command the contents of the parameter file.

In this example the user's copy of the dmcopy parameter file was reset to the default values using punlearn and then dmcopy was called. The infile and outfile parameters were prompted for, since these are the only automatic parameters it has. The default values for these parameters are blank, as indicated by the fact that there is nothing between the "()" brackets at each prompt.

Example 2

```
dmcopy in.fits out.fits
```

Setting values by position

The values are read from the command line and applied to the parameters in the order that they appear in the parameter file, which is not necessarily the order the tool prompts for them. This only works for automatic − ie non−hidden − parameters.

If there are any parameters that have not been specified on the command line that have mode=q, they will be prompted for. In this example we have specified all the automatic parameters for the tool (infile is set to in.fits and outfile to out.fits).

Example 3

```
dmcopy in.fits out=out.fits clobber+
```

Setting values by name

Each "key=value" pair will set the named parameter to the given value. If the value is invalid (below min, above max, etc) the tool will prompt for a correct value. If the parameter listed is not in the tool's parameter file, the tool will exit with an error.

Since the parameter name is included then they do not need to be specified in the order given in the parameter file. It is valid to start off specifying parameter values by position − i.e. the previous example − and then switch to this method. In fact, you need to use this method for setting hidden parameters since the "positional" method above does not work for them.

It is important to note that the opposite DOES NOT apply; if you start out using "key=value" pairs, you must do so for the entire command.

In this example we have set the infile parameter by position (to in.fits) and both the outfile and clobber parameters by name. Since outfile is the only dmcopy parameter beginning with "out" we can set its value using "out=...". As the clobber parameter is a boolean, we use the shortcut "clobber+", which has the same meaning as "clobber=yes".

Example 4

```
dmstat infile=img1.fits,img2.fits,img3.fits centroid−
dmstat infile=@images.lis centroid−
```

Stack support

Some files support parameters containing multiple values, which is referred to as a stack; normally this is used to indicate a list of input files but it need not be. See "ahelp stack" for more information on the use of stacks in CIAO, and the ahelp page for the particular tool for specific details on how it interprets stacks.

In the first example we have set the infile parameter of dmstat to equal "img1.fits,img2.fits,img3.fits"; the tool will run on each file individually with the centroid parameter set to no. The second example introduces the "@" syntax to indicate that the contents of the file after the "@" symbol − so here images.lis − should be used to define the stack. If the contents of this file are:

```
  unix% cat images.lis
  img1.fits
  img2.fits
  img3.fits
```

then the results will be the same as the first example.

Example 5

```
csmooth infile=img.fits outfile=.
```

Autonaming support

If the parameter value is given as "." and autonaming is supported for that parameter, then the tool will automatically generate a file name for the user. See 'ahelp autoname' for information on autonaming support and the ahelp file for the particular tool for specific details.

In this example we tell csmooth to use img.fits as the input file and that the output file should be generated from this name (for csmooth this means that the output will be called img_asm.fits; see "ahelp autoname" for more details of how this is worked out).

Example 6

```
pset acis_process_events eventdef=")cclev1"
```

Parameter redirection: the same parameter file

This syntax tells the user to set the value for a parameter equal to that of another parameter (the name after the ")" character) in the same parameter file.

A common example of this is for the event processing tools where the event definition string is redirected from one of several listed in the .par file; for instance the example above shows the eventdef parameter of acis_process_events being set equal to the cclev1 parameter.

Note that the value is set to ")cclev1" and this is evaluated each time the parameter value is requested. This means that when a tool is run with a "redirection" value for a parameter, that "redirection" is evaluated at that time, rather than using the value that was there at the time the redirection was set. This can be seen by

comparing the output of pget with plist after the above example (the value of cclev1 has been reduced to avoid the value over−runing the edge of the help display):

```
unix% pget acis_process_events eventdef
{d:time,...,x:status}
unix% plist acis_process_events | grep eventdef
   (eventdef = )stdlev1 -> {d:time,...,x:status}) output format definition
```

In many cases this will not matter (for example the cclev1 parameter in acis_process_events does not change).

Example 7

```
mkrmf infile=")acis_fef_lookup.outfile"
```

Parameter redirection: using a different parameter file

The parameter to which we redirect the tool does not need to be in the same file. In this example we set the infile parameter of mkrmf to be equal to the value of the outfile parameter of acis_fef_lookup.

This can be useful when writing scripts that take the output file from one tool and use it as the input file of another. Note that it is important to know when the value of the re−directed parameter is evaluated; the same rules apply as discussed in the previous example.

Example 8

```
punlearn ciao.par
cd /data/Chandra
plist ciao.par | grep workdir
workdir = ))pwd -> /data/Chandra default session analysis dir
pget ciao.par workdir
/data/Chandra
cd /tmp
pget ciao.par workdir
/tmp
```

Parameter redirection: using an external command

The default setting for the workdir parameter of ciao.par is set to "))pwd", which means that it will be set to the name of the directory in which the tool was run. In the example above we show how this value changes when we change directory using the plist command (the grep is used to restrict the output of plist to just the workdir parameter).

Example 9

```
dmlist infile="%xpa(ds9,file)[sky=%xpa(ds9,region)]"
dmcoords acis_evt2.fits option=sky x="%xpa(ds9,x)" y="%xpa(ds9,y)"
```

Parameter redirection: using XPA access points

In the first example, the infile parameter is constructed from information obtained from an active ds9 session. If the current file in ds9 is img.fits, and the image has the region circle(4096,4096,20) displayed on it, then infile will evaluate to "img.fits[sky=circle(4096,4096,20)]". This simplifies the examination of multiple regions, since the region definitions do not need to be stored in external files.

In the second example, the x and y parameters of dmcoords will evaluate to the x and y sky coordinates, respectively, of the current location of ds9's crosshairs. This allows one to interactively select points on an image and feed their coordinates to dmcoords. Note that it is also possible to use logical and WCS coordinates by replacing x/y with lx/ly or ra/dec, respectively.

Example 10

```
dmcopy $ASCDS_INSTALL/param/dmcopy.par /tmp/dmcopy.par
dmcopy @@/tmp/dmcopy.par
```

Use an alternative parameter file

The default behavior of the parameter interface is to look for the parameter file in the user's search path, as defined by the $PFILES environment variable. To use a file outside this location you precede its name by the "@@" characters. In the example above we have copied the default dmcopy parameter file to /tmp/dmcopy.par and then use that; it is likely that you would change the parameter values between the two operation.

This facility allows you to store parameter files with a relative path and still access them when running the tool from another location. If you often use the same parameters for a tool, you can save a copy of the parameter file to a specific location and point to it whenever you want to use these values.

CHANGES IN CIAO 3.0

Editing parameters using a GUI

The peg tool has been introduced which creates a GUI with which you can edit parameter files and then run the tools.

S−Lang interface

The paramio module provides a S−Lang interface to the parameter library. See "ahelp paramio" for more details.

Using environment variables

It is now possible to embed environment variables within a parameter value using the syntax "${name}", where name is replaced by the name of the environment variable. These behave like re−directions in that they are evaluated when the tool is run, and so will pick up the current setting of the variable.

Prior to CIAO 3.0 you had to use "))" to run the echo command to provide this functionality, so what used to be

```
  unix% pset dmextract defaults="))echo $ASCDS_CALIB/cxo.mdb"
```

is now

```
  unix% pset dmextract defaults="${ASCDS_CALIB}/cxo.mdb"
```

type=f

Parameters can now have a type of "f", which indicates that the value is a filename. Whilst this makes no difference to the operation of the tool − which just treats it as a string − it has been added so that peg, the parameter editor GUI, can display a "browse" button next to these parameters.

Improved adherance to the mode parameter for hidden parameters

In earlier versions of CIAO, hidden parameters − those with a mode of "h" − could be updated at runtime without setting the "learn" mode. Now the values of hidden parameters in a parameter file can not be changed except by:

- Using the pset command−line utility or pset() S−Lang function (from the paramio library).
- Explicitly editing the value field in the parameter file.
- Calling the S−Lang paramopen() function with the mode containing "L" − the learned mode override.

RUNNING MULTIPLE COPIES OF A TOOL

If you run several copies of the same tool at the same time you will probably want to use multiple copies of the parameter file, as it is unlikely that the parameters will be unchanged for the different copies. This situation can also apply if you are analysing multiple datasets, since you may want to have separate ardlib − and perhaps other − parameter files for each dataset. You therefore need to use different parameter files for each copy, and there are several ways of doing this.

In the following we shall consider the example of running aconvolve to smooth an image (img.fits) with a two−dimensional gaussian with sigma values of 3 and 5 pixels along each dimension.

Using '@@'

If this is a one−off situation then you can set up the parameter file, copy it to a separate location and then explicitly use that copy. In the example below we set up two parameter files, one for using a gaussian with a sigma of 3 pixels in each direction (/tmp/aconvolve.g3.par) and the other with the sigma value set to 5 pixels (/tmp/aconvolve.g5.par). We then run two copies of aconvolve with these individual parameter files, setting the mode to "h" to avoid any queries of automatic parameters.

```
unix% punlearn aconvolve
unix% pset aconvolve infile=img.fits method=fft
unix% pset aconvolve outfile=img.sm3.fits
unix% pset aconvolve kernelspec="lib:gauss(2,5,1,3,3)"
unix% cp `paccess aconvolve` /tmp/aconvolve.g3.par
unix% pset aconvolve outfile=img.sm5.fits
unix% pset aconvolve kernelspec="lib:gauss(2,5,1,5,5)"
unix% cp `paccess aconvolve` /tmp/aconvolve.g5.par
unix% aconvolve @@/tmp/aconvolve.g3.par mode=h &
unix% aconvolve @@/tmp/aconvolve.g5.par mode=h &
```

Changing the PFILES environment variable

A more robust solution is to run each copy of the tool with the PFILES environment variable set to a different local directory. Assuming that the default PFILES setting after starting CIAO looks like

```
/home/username/cxcds_param;/soft/ciao/param
```

and you want to use the directories "/home/username/cxcds_param1/" and "/home/username/cxcds_param2/", then you could say

```
unix% setenv PFILES "/home/username/cxcds_param1;/soft/ciao/param"
unix% punlearn aconvolve
unix% pset aconvolve infile=img.fits method=fft
unix% pset aconvolve outfile=img.sm3.fits
unix% pset aconvolve kernelspec="lib:gauss(2,5,1,3,3)"
unix% aconvolve mode=h &
unix% setenv PFILES "/home/username/cxcds_param2;/soft/ciao/param"
```

```
unix% punlearn aconvolve
unix% pset aconvolve infile=img.fits method=fft
unix% pset aconvolve outfile=img.sm5.fits
unix% pset aconvolve kernelspec="lib:gauss(2,5,1,5,5)"
unix% aconvolve mode=h &
```

Whilst this is − in general − a better approach than the previous suggestion it does suffer from the issue of having to remember which parameter file/directory you are currently using.

You do not need to use an absolute path when setting PFILES, it can often make sense to use a relative path. So, if you set PFILES to

```
./param:/home/username/cxcds_param;/soft/ciao/param
```

then the parameter library will look for the file first in the subdirectory param/ of the local directory, then in your cxcds_param/ subdirectory of your home directory, and then finally fall back on the system default copy in the CIAO distribution (here taken to be /soft/ciao).

Setting the parameter file directory to the current working directory, e.g. "." (dot), is a special case. If you want to put the files in the current working directory, you must include a slash in the PFILES variable:

```
./:/home/username/cxcds_param;/soft/ciao/param
```

CONTENTS OF A PARAMETER FILE

Each parameter is described by a comma−separated line of text listing the name, type, mode, value, minimum or enumeration, maximum, and prompt text for that parameter. See the table below for a description of these fields, and note that not all fields need to contain data. Blank lines and lines beginning with "#" are ignored.

The following fields are used for each parameter:

Parameter	Description
name	Name of the parameter used by the tool.
type	datatype of the parameter. Valid values are s=string, f=filename, i=integer, r=real, b=boolean, and "pset".
mode	controls whether the parameter is prompted for or not. Valid mnemonics are q=query, h=hidden, a=automatic. Can be modified with l=learn, e.g. "ql" means to query and learn. Note that the mode parameter of the tool itself can modify the setting of an individual parameter.
value	default parameter value. Can be blank if no sensible default exists. If the mode value contains the letter 'l' then the last value becomes the new default after the tool is run.
minimum or enumeration	minimum data value. Can be blank if no sensible value exists. Can also be a "|" separated list of values. The value can only take one of the values in the enumeration.
maximum	maximum data value.
prompt	The prompt the user will see if they are prompted for the value.

As an example, consider the default parameter file for dmappend:

```
unix% punlearn dmappend
unix% plist dmappend

Parameters for /home/username/cxcds_param/dmappend.par

      infile =                  Input dataset/block specification
     outfile =                  Output dataset name
    (verbose = 0)               Debug Level
       (mode = ql)
```

```
unix% cat `paccess dmappend`
infile,f,a,"",,,"Input dataset/block specification"
outfile,f,a,"",,,"Output dataset name"
verbose,i,h,0,0,5,"Debug Level "
mode,s,h,"ql",,,
```

The infile and outfile parameters do not have any default value (the field is set to ""), and are listed as auto parameters (the "a" value) which is why plist does not include "()" around their names above. The verbose parameter defaults to 0 and is limited to be between 0 and 5, inclusive.

The acis_fef_lookup parameter contains an example of an enumerated list for the chipid parameter:

```
unix% cat `paccess acis_fef_lookup`
infile,f,a,"",,,"Source file (event or spectrum)"
chipid,s,a,"none",none|NONE|0|1|2|3|4|5|6|7|8|9,,"ACIS chip number"
chipx,i,a,1,1,1024,"ACIS chip x coordinate"
chipy,i,a,1,1,1024,"ACIS chip y coordinate"
outfile,s,h,"",,,"FEF file to use"
verbose,i,h,0,0,5,"Verbose level"
mode,s,h,"ql",,,
```

The |−separated list of items for the fifth field of the chipid parameter lists the valid values for this parameter. If a value other than none, NONE, or an integer between 0 and 9 is entered for the chipid parameter, then a warning message will be displayed and the user will be prompted for another value, as illustrated below.

```
unix% pset acis_fef_lookup chipid=aciss3
pquery: invalid enumerated value : chipid
ACIS chip number (none|NONE|0|1|2|3|4|5|6|7|8|9) (none):
```

Note that you can use the up and down arrows to cycle through the list of possible answers at the prompt.

See Also

calibration
caldb
chandra
coords, guide, isis, level, pileup, times
chips
chips
concept
autoname, stack, subspace
dm
dm, dmbinning, dmcols, dmfiltering, dmimages, dmimfiltering, dmintro, dmopt, dmregions, dmsyntax
gui
gui
modules
paramio, pixlib, stackio
paramio
paccess, paramclose, paramopen, pget, pgets, plist_names, pquery, pset, punlearn
slang
overview, slang, tips
tools
dmhistory, dmkeypar, dmmakepar, dmreadpar, paccess, pdump, pget, pline, plist, pquery, pset, punlearn

URL: http://cxc.harvard.edu/ciao3.4/parameter.html
Last modified: December 2006