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

AHELP for CIAO 3.4

stack

Context: concept

Jump to: Description Examples CHANGES IN CIAO 3.0 Bugs See Also

Synopsis

How to set a parameter to more than one value using a stack.

Description

Normally tool parameters can only be set to one value at a time. However, multiple values can be specified for a parameter if is listed in the help file as accepting stacks. The most common use of stacks is to specify a set of file names − either input or output − to a tool, but they can be used for anything where multiple values are required.

As an example, consider dmstat: its infile parameter is marked as taking a stack, which means that it can handle multiple inputs in one go. These inputs can be different files or the same file but with different filters, or a combination of both.

The parameter list for a tool will tell you whether it can accept stacks and for which parameters: use "ahelp −b PARAM <toolname>" (or "ahelp −b PARAM −t <paramname> <toolname>") to quickly find this information.

Tools may treat stacks in two ways:

- as one continuous data set producing a single output file ("N to 1").
- as independent data sets that produce independent results ("N to N").

The modes that are supported depend on the function of the tool. Check each tool's help file to see how they are treated.

What is a stack?

There are several ways to specify a stack:

- As a space, comma, or semi−colon separated list
- Using UNIX wild−card substitutions
- Using an ASCII file
- Using a linear, integer, rectangular, or polar grid in a Data Model filter

The Examples section below shows how to use these different methods.

Including a Data Model filter in a stack

When a Data Model filter (see "ahelp dm") is included as part of the filename, it must be contained within double quotes:

```
unix% dmcopy "acis_evt2.fits[ccd_id=3,sky=region(ds9.reg)]" \
      out.fits
```

If that DM filter is within a stack, however, DO NOT include the quotes:

```
unix% cat in.lis
evt2.fits
evt2.fits[ccd_id=3,sky=region(ds9.reg)]
```

Creating regularly−spaced grids

If any one of the following filters is used in a Data Model filter then the tool will actually see a stack of filters, rather than just one. As shown in the Examples section, the rectangular and polar grid forms can be used to create a grid of regularly−spaced regions whilst the linear grid can be used to select a subset of rows from a table. The igrid specification forces an integer grid.

The following grid specifications are available:

grid type	grid syntax
rectangular grid	rgrid(xmin:xmax:dx,ymin:ymax:dy)
polar grid	pgrid(xcen,ycen,rmin:rmax:dr,thetamin:thetamax:dtheta)
linear grid	lgrid(min:max:step)
integer grid	igrid(min:max:step)

Creating and manipulating stacks

Whilst tools can accept stacks, it is also possible to create and manipulate stack files from the command line − by using the stk_build, stk_count, and stk_read_num tools − and from S−Lang − by using the functions made available by the stackio module. See the appropriate ahelp files − "ahelp tools /stk/" for a list of the command−line tools and "ahelp stackio" for the S−Lang module − for more information.

These tools/functions can be used to see how a stack expression is expanded. For instance, if in.lis contains

```
unix% cat in.lis
evt2.fits
evt2.fits[ccd_id=3,sky=region(ds9.reg)]
```

then we can see how "@in.lis" and "@in.lis[cols sky]" are expanded by:

```
unix% stk_build "@in.lis" stdout
evt2.fits
evt2.fits[ccd_id=3,sky=region(ds9.reg)]
unix% stk_build "@in.lis[cols sky]" stdout
evt2.fits[cols sky]
evt2.fits[ccd_id=3,sky=region(ds9.reg)][cols sky]
```

Example 1

```
dmstat "img1.fits,img2.fits"
```

Explicitly listing all stack elements

The simplest stack is just a comma−separated list of values. For dmstat − which accepts stacks for the infile parameter − this means that the statistics of each file will be calculated separately.

Note that spaces and semi−colons are also valid stack separators, so the following forms are equivalent to the first one:

```
unix% dmstat "img1.fits img2.fits"
unix% dmstat "img1.fits;img2.fits"
```

When ' ', ',', and ';' do not mean a stack

Comments, spaces, and semi−colons are not treated as stack separators when they appear inside of "()", "[]", or "{}" pairs. So

```
"evt2.fits[cols time,sky]"
```

is not treated as a stack.

If a space, comma, or semi−colon is preceeded by a "\" then it is also not treated as a stack separator, i.e. the "\" character is not removed from the text:

```
unix% stk_build "img1.fits\,img2.fits" stdout
img1.fits\,img2.fits
```

Example 2

```
dmstat "img*fits"
```

Using UNIX wild−card substitutions

Most UNIX wild−card substitutions are supported, as long as they are protected from being expanded by the shell (which is why quotes are used around img*fits above). Thus if the current directory contained the files img1.fits and img2.fits, the stack expansion would be equivalent to "img1.fits,img2.fits". For this particular example the following would produce the same result:

```
unix% dmstat "img?.fits"
```

Do not use [] as a wild−card expression

Since the "[]" characters are used to indicate a DM virtual−file specification (see "ahelp dmsyntax") they can not be used as a wild−card expression. This means that

```
"pcad*_[ao]sol1.fits"
```

will NOT select both the asol1 and osol1 files.

Example 3

```
dmstat "@imgs.lis"
```

Using an external file

The most common form of specifying a stack is to store the values in an ASCII file and then setting the parameter to "@<filename>".

Thus if imgs.lis contained

```
unix% cat imgs.lis
img1.fits
img2.fits
```

then the parameter would be equivalent to "img1.fits,img2.fits".

Recursive expansion

By default, recursive stack expansion is not supported, so if the file contained something like

```
@more_images.lis
```

then more_images.lis would NOT be expanded.

As will be shown in a later example, the "@+" syntax can be used to turn on recursive expansion.

Ignoring lines in a stack file

Blank lines and those beginning with a '#' character are excluded from the stack expansion.

Writing a long expression over multiple lines

Stack items can be divided over several lines by ending each line with a '\' character. This feature can be useful to break up dmtcalc expressions:

```
unix% cat expr.lis
GPHASE=(time-TSTART)* \
       (19.794885+(4.79e-13*(time-TSTART)))
unix% dmtcalc evt2_bary.fits evt2_bary_phase.fits @expr.lis
```

(this expression is taken from the Create a Phase−binned Spectrum thread with example numbers filled in for the frequency and frequency derivative terms).

Example 4

```
dmstat "@imgs.lis[sky=region(src.reg)]"
```

Filtering a stack

The syntax "@<filename>[DM expression]" tells the tool to apply the DM expression (as discussed in "ahelp dmsyntax") to each file listed in the file.

So, if imgs.lis contains img1.fits and img2.fits then this syntax is a shortform for setting infile equal to

```
"img1.fits[sky=region(src.reg)],img2.fits[sky=region(src.reg)]"
```

Example 5

```
dmstat "img*[sky=region(src.reg)]"
```

Filters can be applied to any form of stack

We can also apply a DM filter to a wild−card. If the current directory contained img1.fits and img2.fits then the above would be the same as setting infile equal to

```
"img1.fits[sky=region(src.reg)],img2.fits[sky=region(src.reg)]"
```

Example 6

```
dmstat "@/tmp/img.lis"
```

Using stacks outside the current directory

If the stack file is in another directory − here /tmp/ − then this directory name will automatically be added onto each entry in the stack.

In this example, if /tmp/img.lis contained img1.fits and img2.fits, then the above syntax would be equivalent to setting infile to

```
"/tmp/img1.fits,/tmp/img2.fits"
```

This makes it easy to use stacks in different directories: for instance if you create a file called asol.lis in the directory containing all the asol files for an observation − eg

```
  unix ls −1 *asol*fits* > asol.lis
```

then you can use "@/full/path/to/asol.lis" whenever you need these files (such as the acaoffile parameter of acis_process_events).

As will be discussed in the following examples, this behaviour can be over−ridden.

Example 7

```
dmstat "@/tmp/img2.lis"
```

How to avoid the stack changing a file name

Two ways to avoid pre−pending "/tmp/" to the names in the stack file are:

- Include the full path to the file.
- Begin the file name with a ! character.

So, if /tmp/img2.lis contained

```
  unix% cat/tmp/img2.lis
  /data/Chandra/img1.fits
```

```
  !img2.fits
  img3.fits
```

then the stack would be expanded to

```
"/data/Chandra/img1.fits,img2.fits,/tmp/img3.fits"
```

Example 8

```
dmimghist @in.lis @out.lis @−/tmp/bins.lis
```

Ignoring the location of a stack file

The "@−" for the third parameter − the hist parameter − of dmimghist tells the stack library not to prepend "/tmp/" to each entry in /tmp/bins.lis. This is useful here because, for dmimghist, the hist parameter does not refer to a list of files but a list of binning specifications.

So, if the contents of the three stacks are:

```
  unix% cat in.lis
  img1.fits
  img2.fits
  unix% cat out.lis
  hist1.fits
  hist2.fits
  unix% cat /tmp/bins.lis
  1:10:0.2
  −0.5:50:0.5
```

Then the example above is the equivalent to running

```
  unix% dmimghist img1.fits hist1.fits 1:10:0.2
  unix% dmimghist img2.fits hist2.fits −0.5:50:0.5
```

If we had used "@/tmp/bins.lis" instead, dmimghist would have seen the hist parameter set to "/tmp/1:10:0.2,/tmp/−0.5:50:0.5", which does not make sense.

This example also shows that stacks are not just used for input parameters − they can be used to determine the output names of files (here the contents of out.lis).

Example 9

```
dmstat "@+/tmp/in.lis"
```

Recursive stacks

The "@+" syntax tells the tool that any entries in the stack beginnning with a "@" should be treated as stacks, in other words to recursively expand the stack.

If the contents of /tmp/in.lis are:

```
  unix% cat /tmp/in.lis
  img1.fits
  @in2.lis
  !img2.fits
```

and

```
  unix% cat /tmp/in2.lis
  evt2.fits[cols sky]
```

then the stack is expanded to

```
"/tmp/img1.fits,/tmp/evt2.fits[cols sky],img2.fits"
```

Note that here we are not just expanding the contents of in2.lis into the stack but are also prepending the directory name onto the entries (apart from when we explicitly stop this using the "!" syntax). This highlights the fact that the stack expansion rules are cumulative.

Example 10

```
dmextract \
"evt2.fits[sky=region(reg.fits[component=lgrid(1:5:1)])][bin pi=::1]" \
multiple_spectra.pi
```

Using a linear grid

The lgrid option creates a linear stack of numbers: in this example the sequence 1,2,3,4,5. This stack is then used to filter the FITS region file − reg.fits − using the component column, which creates a stack of 5 regions corresponding to the first 5 entries in reg.fits. The result of this is that dmextract will extract 5 spectra − in PI space − and store the results in a single type II PHA file.

If a stack − of 5 files − had been used for the outfile parameter then dmextrat would instead have created 5 type I PHA files, one for each of the first 5 regions in reg.fits.

Example 11

```
dmstat "evt2.fits[ccd_id=3,chip=rgrid(1:1024:32,1:1024:256)][bin chip]"
```

Using a rectangular grid

Here we use the rgrid syntax to split the event file up into a stack of rectangular regions based on the chip coordinate system (as well as filtering to ensure that only ccd_id=3 is used). These regions are then binned into images and dmstat run on each image.

The stack expansion in this example comes out to:

```
evt2.fits[ccd_id=3,chip=rectangle(1,1,33,257)][bin chip]
evt2.fits[ccd_id=3,chip=rectangle(1,257,33,513)][bin chip]
...
```

Example 12

```
dmstat "img.fits[sky=pgrid(4123,3950,0:40:10,0:360:180)]"
```

Using a polar grid

Here we run dmstat on an image, using a polar grid of regions, which expands to:

```
img.fits[sky=pie(4123,3950,0,10,0,180)]
img.fits[sky=pie(4123,3950,0,10,180,360)]
img.fits[sky=pie(4123,3950,10,20,0,180)]
img.fits[sky=pie(4123,3950,10,20,180,360)]
img.fits[sky=pie(4123,3950,20,30,0,180)]
img.fits[sky=pie(4123,3950,20,30,180,360)]
img.fits[sky=pie(4123,3950,30,40,0,180)]
img.fits[sky=pie(4123,3950,30,40,180,360)]
```

Note that the x, y, and radial grids are in the units of the selected coordinate system whereas the theta units are in degrees.

CHANGES IN CIAO 3.0

The stack library can now be used from S−Lang. See "ahelp stackio" for more information on this interface.

Bugs

See the bugs page for the stack library on the CIAO website for an up−to−date listing of known bugs.

See Also

calibration
caldb
chandra
coords, guide, isis, level, pileup, times
chips
chips
concept
autoname, parameter, subspace
dm
dm, dmbinning, dmcols, dmfiltering, dmimages, dmimfiltering, dmintro, dmopt, dmregions, dmsyntax
gui
gui
modules
paramio, pixlib, stackio
slang
overview, slang, tips
tools
stk_build, stk_count, stk_read_num, stk_where

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