2.5 Catalog Information Displayed Over Images

This section deals with using the results of a catalog search (or other information in a table file) in conjunction with displayed images. As long as the information is in table file format, it can be used with the following steps. The table file format is a simple ascii text file and can be as simple as columns separated by spaces, but can also have a more complex format, as needed.[15]

2.5.1 The Table Command

The **table** command is used to specify that input to another function is to be taken from a table. That table file name is given as the first parameter to the **table** command, and the other function name and parameters follow. For example, if I have an ascii file named **pics.look** with one column of image names in it that I wish to paint, I could use the command

```
> table pics.look pa $field1
```

The keyword **$field1** denotes the first field in the table (in this case, it's the *only* field) to be used as the **filename** parameter to the **paint** command. So each image in the column is displayed, one after another, until the end of the file is reached. (Of course, you would probably want to turn on the `fr adv` so each image would be painted in a separate window.)

2.5.2 The Mark Command

The **mark** command is used to put a certain type, color and size marker on a particular position on the image. It is strictly for display. Unlike the analysis function **pick**, **mark** doesn't report any information about the pixels. Also, it does not coerce the marker location to the center of the underlying pixel, but indicates the exact position on the pixel. The possibilities for the marker type, color, and size are:

SYMBOLS	COLORS	SIZES
dot	blue	Any number of pixels is valid.
cross	green	Default is 7.
asterisk	red	
circle	white	
x	Black	
box	Or any available X-color	
diamond		
triangle		

[15]See **Appendix C** for a discussion of the table file format.

One of the newer options is that now you can specify X-colors instead of just the 5 specific colors listed above.[16] So to put a small purple triangle at 2h45m16.7s 60d29m52s on an image you have displayed, the command

```
> mark 2h45m16.7s 60d29m52s triangle purple 3
```

would do the trick. Now, you can imagine how easy it would be to use the **mark** command in conjunction with the **table** command, to say, mark a list of sources found by a catalog search. The command

```
> table search1 mark $field1 $field2 box red 5
```

would take the positions for marking from the first two columns of the file **search1**, and put a red box, 5 pixels wide on each one. If the catalog had labelled the columns "ra" and "dec" the command could be:

```
> table search1 mark $ra $dec box red 5
```

2.5.3 The Graphics Pick

Once you've **mark**ed a position (or a list of positions) with the **mark** command, you can go back and refer to the input information relative to it by using the **gpi**, or **graphics pick** command. This command is used in the same way as the data inspection tool **pick**, but the output is different. You enter the command

```
> gpi
```

and then roam the image by moving the mouse cursor. Every time you click the left mouse button, the program reports the nearest **mark**ed position, the name of the table the mark came from, and the entire corresponding line for the entry.

If the mark was not from a table, the coordinates of the **mark** are reported. To get out of the **gpi** mode, either click the middle mouse button, or hit a carriage return on the keyboard.

[16] See **Section 2.4.3** for complete details on how to **set** X-colors for any graphics overlay.

2.5.4 The Graphics Examine

Perhaps you would rather see the catalog information on a *group* of **mark**s in a particular region of the image, instead of one mark at a time.

The **gex**, or **graphics examine** command is specifically for this purpose. Once you have painted an image and marked some points, you can define an area, and then use the **gex** command to see all the table entries for all the marks that lie within that area. For example, the commands

```
> pa orionb1.fits
> table search1 mark $field1 $field2 box red 5
> da ci
> gex
```

would paint the image **orionb1.fits**, then **mark** points using the first two fields from a table named **search1**, define a circular area (interactively with the mouse), and examine the area by displaying all the table entries within it. Something like this:

```
mark came from line 15 of file search1, printed below:
212.541250 42.644722 0.25160 0.25000 0.40000 1.00000
mark came from line 20 of file search1, printed below:
213.086250 41.755556 1.27800 0.27460 0.40000 1.00000
mark came from line 25 of file search1, printed below:
214.237083 42.235278 0.65060 0.25000 0.40000 1.00000
```

2.6 The skyview Environment

This section describes how the **skyview** environment behaves and how to customize things to your liking. There's nothing difficult about them, they just all need to be in one place, and this is it!

2.6.1 Reporting the Status–Command Input With No Parameters

Several **skyview** commands will report the status or setting of the function when no parameters are specified on the command line. Here is a list of those commands, and a brief description of what is reported:

NAME	WHAT GETS REPORTED
al	list of aliases
ca	which area number is currently being used
ci	current image, range, projection type, blank value, image type, mode, frame number, zoom factor and center position
cl	clip settings
cs	coordinate system setting
ct	color table setting
cu	cutoff level
fi	file name to be used for the next command (pa, he, ho, ra, tw, rep)
fr	current frame number
ma	which color plane is currently being used
mag	magnifier window status
mo	mode for transform from file units to screen pixel values
pl	plane being used in multi-plane image
ra	range of pixel values being used for transform from file units to screen pixel values
set	values for all set-able parameters

For example,

```
> fi
```

will report the name of the file to be used in subsequent commands. But,

```
> fi ori.fits
```

will designate the file named **ori.fits** to be used in subsequent commands.

2.6.2 The Sticky Stuff

Many command parameters are 'sticky'. That is, once the parameter has been set, it will remain set to the current value until changed again by the user.

NAME	WHAT STICKS
al	any aliases that are defined.
an	*color* and *font* used in annotation.
ba	background level last defined.
ca	area number last defined.
cl	number of pixels clipped for display.
cs	coordinate system last specified.
ct	color table last specified.
cu	cutoff level: threshold of screen pixel values to be displayed.
da	sexigesimal vs. decimal coordinate format, area type, area number.
pa	*st* or *nost* (whether to reset color table back to 0-255 before paint).
pa	*zp* or *nozp* (whether to reset zoom/pan before painting a new image in the same window).
fi	filename specified for future painting.
fr	frame number or advance for displaying multiple images each in a separate window.
he	*long* or *short* version.
ma	image planes enabled.
mag	magnifier disposition (*on*, *off*, *magnification level*).
mark	marker type, color and size.
mo	mode *linear* or *logarithmic* or *loglog* or *equilization*.
pl	plane number of multi-plane image.
ra	range level values last specified.
sh	session *on* or *off*, and name of history file.
sync	X-Windows sync *on* or *off*.
ta	whether errors abort a take file.
we	color wedge display: *on* or *off*.

For example, if I specify color table 4 with the **ct** command

```
> ct 4
```

all images will be displayed using color table 4 until I change it:

```
> ct 0
```

2.6.3 Successive commands on the input line

The semi-colon (;) can be used as a delimiter between commands in order to string multiple commands on a line. This is especially helpful for defining an alias to a group of commands you plan to execute over and over (See next item!).

2.6.4 Creating and Using Aliases

The alias allows the use of a name for a long string of commands or a set of coordinates.

```
> al po1 "2h45m23.8s 60d18m52s"
```

will create the definition **po1** for the coordinate given in the quotation. Whenever **po1** is used, the coordinate string will be substituted before any action is taken. One string can refer to another **alias** name as many levels deep as desired. Alias substitution is never done on an **alias** command itself. All substitutions occur later when the **alias** is used in a command.

```
> al
```

with no arguments displays all the aliases you have defined, along with their definitions, like this:

```
Current Aliases
=============
po1 2h45m23.8s 60d18m52s
```

The command

```
> al po1
```

will **delete** the alias named **po1**.

The Dollar Sign $ – a special alias. When referring to coordinates, it can get frustrating having to type in the coordinate time after time. If you don't want to create a special alias for each of the coordinates you use, you can use the shortcut alias, **$**. Here's an example. Say you want to determine the position of your object using the **pick** command, and then want to place a circle with a 10 arcmin radius around the object, using **da**. Here's how that would work, using the **$**alias:

```
> pick
19055.8 MJY/SR   at RA = 17h42m41.04s  Dec = -28d56m42.2s

> da ci 10 $
Circle of radius 10 arcmin
Centered at RA = 17h42m41.04s  Dec = -28d56m42.2s

>
```

So the pixel position reported by the **pick** command was remembered and echoed into the **da** command line when the **$** was used.

2.6.5 Customizing Your Environment

If you are reading this for the first time, *please* do not panic when you read this section. You do not *have* to customize your environment. You can ignore this entire section if you like. More experienced users may like to set **skyview** up to have certain attributes automatically turned on or off when they start up or while using the program. There are different methods of accomplishing this, and all the options are common and fairly flexible.

There is a simple way to remember what each method is meant to be used for: The UNIX environment variables and the X-resources are used for setting up the environment for *all* **skyview** sessions. The command line options are used for setting up the environment for the duration of *this* **skyview** session. The **set** command is used mostly for changing the status of an environmental variable during, but not necessarily for the duration of the session.

The skyview Command Line Options As stated earlier, to begin working with the program, you simply type it's name: `skyview`. There are also some command line options that can be specified when **skyview** is invoked, in order get **skyview** to work in the manner you desire. For most users, only the `-server` and the `-display` option are relevant. The others are only used in unusual cases.

Parameter	Definition	Default
-d	skyview debug mode	off
-n	No graphics display	graphics on
-display	Display images on a different screen	current host ($DISPLAY)
-server	Server mode	off
-p	Select palette size	240
-c	Run in captive mode	false
-imio	Run in IRAF imio mode	off

A few examples of how to use these parameters:

```
% skyview -d
```

turns on the debug mode. This is not a recommended command, because it causes a lot of debugging messages to appear when using **skyview**.

```
% skyview -display lotus.car.podunk.edu:0
```

causes all images to be displayed on the screen of the machine called **lotus.car.podunk.edu**. This is usually used when you are sitting at **lotus.car.podunk.edu**, and you have images on another machine that you wish to look at without having to copy them to your host machine. You must also add the machine where **skyview** is running, e.g. **bigmachine.podunk.edu**, to the X-window server access control list for the machine you're sitting at, using the UNIX command:

```
% xhost +bigmachine.podunk.edu
```

The 'set' command The **set** command is used to set the environment variables *within* **skyview**. By typing the command:

```
> set
```

with no parameters, you get a report of all the possible variables that can be set, and their current status. Here's the current list of variables, their definitions, and defaults:

(per image):

Variable	Definition	Default
blank	Value of blank pixels if no BLANK keyword in FITS headers	-2e+09
blank_color	Color used when painting a blank pixel	color_table[0]
u16	Denotes that the image is unsigned 16-bit. [17]	off
projtype	Recognize the PROJTYPE keyword in FITS headers.	off

(global):

Variable	Definition	Default
mouse	Allow mouse input	on
reread	Re-read image file every time it's painted	off
conserve_mem	Conserve memory by only storing the last image painted	off
color	X-color to be used for overlays	default [18]
zap_value	Value to replace pixel with when using the zap command	blank
imio	IRAF pipes being watched	off
sexigesimal	Coordinate value formatting	default [19]

[17] The **unsigned 16-bit** option interprets image data files in a non-standard way, but is useful for working with some types of raw data.

[18] The default overlay color is **green**, with the exception of slice which uses both green and red.

[19] The default is sexigesimal, but then skyview mimicks the format that it observes on coordinate input.

The UNIX Environment Variables You can customize your environment by using certain environment variables. If you don't set these variables before starting **skyview**, the defaults are based on decisions made when the program was installed. Here is a list of the variables, and what they do. After the list, we shall discuss each variable in more detail.

- DISPLAY is the X-Window server to be used.
- SLOGIN is the name of the start-up file. **skyview** executes the commands in this file at start up if it is running in the interactive mode (-server flag not set).
- SHELP is the name of the directory containing the **skyview** help files.
- SDATA is a path containing directories to search for valid **color table** and **fov** files.
- STAPE is the name of tape device to be used when reading and writing tapes with **skyview**.
- SPRINTER is the name of printer device to be used for output of hardcopy from the **contour_plot** command.

The use of the **.slogin** file is analogous to the UNIX **.login** file. You simply create a file in your default (home) directory called **.slogin**, and put in whatever **skyview** commands you want. When you start up the **skyview** program, the commands inside the **.slogin** file are executed before you get the initial prompt. You don't have to have a **.slogin** file. If the **skyview** program doesn't find one, it won't complain. If you prefer to place this file in another directory or to give it a different name, set the UNIX environment variable **SLOGIN**. For example, the command

```
% setenv SLOGIN /test/joe/newslogin
```

tells **skyview** to read the file */test/joe/newslogin* as the **.slogin** file instead of looking in your home directory.

The **SHELP** variable probably won't ever need to be set by the average user, since it designates the path name where **skyview** should look to find the help files for its various functions. This variable is defined in the **skyview** startup script when it is installed at your site. Unless the help files are moved, you don't need to re-set it.

The **SDATA** variable is used to specify a path for **skyview** to search for data files. These data files are **not** images. They are either color table or field of view files (currently only used by the **fov** command) that don't reside in either your home directory or a directory you plan to be working in. Say you have some **fov** files in /scr/data. You can make sure **skyview** will find all of the files by using the command:

```
% setenv SDATA .:~:/scr/data
```

Or, if a user named 'joe' has created a color table[20] that you want to use, and he put it in his home directory, you can add his home directory to your SDATA path:

```
% setenv SDATA .:~:/scr/data:/usr1/home/joe
```

The **STAPE** variable is used to specify which tape drive you want **skyview** to use when reading or writing magnetic tapes. (Remember, **skyview** only reads and writes FITS tapes!) When **skyview** was installed, a tape device name was specified as the default for your site. Check with the person who did the installation for the name. If you don't wish to use that specific tape drive, you can use the command

```
% setenv STAPE /dev/xxxxx
```

to override that default (where **xxxxx** is the name of the tape device you wish to use).

The **SPRINTER** variable is used to specify which printer you want **skyview** to use when producing hardcopy output. When **skyview** was installed, a print device name was specified as the default for your site. You can determine the default by looking at the skyview startup script. If you don't wish to use that default printer, you can change your default printer to be, for example, a printer named **laserplus**, by using the command:

```
% setenv SPRINTER laserplus
```

X-Resources Some X-resources can be defined for setting up the **skyview** environment. There are many different ways to set the X-resources. Consult your local X-Windows expert for help if you don't know how. Undoubtedly, the list of X-resources that can be set will be growing in the future. Here is a current list of **skyview** X-resources and their internal defaults:

```
skyview*debug:  False
skyview*palette:  240
skyview*serverMode:  off
skyview*mainButton1:  stretch
skyview*mainButton2:  pick
skyview*mainButton3:  zoom_pan
skyview*mouse:  on
skyview*projtype:  off
skyview*unsigned16Bit:  off
skyview*reread:  off
```

[20]See **Appendix B** for more information on how to create your own color table.

```
skyview*conserveMem:  off
skyview*color:  default
skyview*imio:  off
skyview*font:  6x13
skyview*pointerShape:  crosshair
```

As you can see, some of the same variables that can be specified on the command line or **set** inside **skyview** with the **set** command can also be defined as X-resources.

2.6.6 Using skyview to display images from the IRAF environment

skyview can now be used as the image display device from within IRAF. It doesn't take any special files or set-up in order to get all the functionality of **skyview** in conjunction with all the data reduction tools within IRAF.

When starting **skyview**, use the command

```
% skyview -imio
```

to enable **skyview** to accept IRAF commands. Then, start IRAF in another window. As long as you don't have any other image display utilities running, IRAF will know to send all images to **skyview** display windows. You will still be able to use all the usual IRAF functions with the image, with the added benefit of having as many images as you wish displayed on the screen in separate windows!

2.7 Complete Function List

What follows is the complete list of **skyview** functions. Many of them were covered previously in this chapter, others were not. For specific details, use the `help` command on the individual function.

al Define an alias based on functions or parameter definitions.

an Annotate the screen with text.

ba Define a background level to be used in flux integration.

border Provide black border around a displayed image.

ca Change from one defined area to another.

cd Change directories.

ci Tell which filename is associated with the current image.

cl Clip off pixels from the image upon display.

co Create a hardcopy contour plot of an image or overlay it on the displayed image.

crop Crop out a portion of an image, creating a new file containing only that cropped portion.

cs Specify the coordinate system to work within.

ct Specify the color table to use when displaying images.

cu Cutoff the lower end of the color table.

da Define an area within a displayed image.

df Delete a file.

dir List the contents of the current directory.

echo Echo a given string to the screen.

end Exit the program.

er Erase all or specific images displayed on the screen.

ex Examine a defined area. (flux integration)

exit Exit **skyview**.

fi Set file name to work with.

fl Flatten an image by removing a least-squares fit plane.[21]

fov Display graphics overlay on an image.

fp Find the IRAS Skyflux or ISSA plate given certain coordinates.

fr Specify the display frame number to work within.

gex Report catalog data from a table based on marks placed on a defined area.

gpick Report catalog data from a table based on a mark nearest the cursor.

gpoint Gpick, one time only.

[21]The software used to create the **flatten** routine is proprietary. If you have the source code release of **skyview**, you will notice that it doesn't include the source for **flatten**.

grid Draw a coordinate grid on the image.

help Display help on a given function.

he Display the header of an image.

hi Display the histogram of a displayed image.

ho Display the histogram of an image file based on the original data values.

ma Map the display of an image to a specific color plane.

mag Display a magnifier window for seeing close-up detail while moving the mouse.

mark Mark a specific point on an image with a particular color and shape.

md Memory dump (screen dump in "no graphics" mode).

mo Set the mode for range stretch.

movie Plat a series of images as a movie.

movie_prep Prepare a series of images for a movie command.

of Turn off a specific color plane.

on Turn on a specific color plane.

pa Paint (display) image from disk onto screen.

paint_blank Display a blank window.

pause Pause and wait for a carriage-return before executing any more commands.

ph Print a log of the session history.

pi Pick a pixel value and location on a displayed image.

plane Select a plane from a multiple plane image for display.

point Pick, one time only.

pwd Print working directory name.

quit Exit the program.

ra Set the range for transfer function during paint.

rep Replicate image pixels in either direction.

rew Rewind tape.

sb Skybox–compute sky statistics.

sc Create a scatter plot comparing the pixel values in the same found in two separate images.

sd Screen dump a displayed image to tape or disk file.

set Assign values to **skyview** environment variables.

sh Keep a log of the session history.

sl Cut a slice (and display it's profile) through a displayed image.

sleep Pause for a given length of time.

st Stretch the color table.

sync Synchronize the X-windows display to keep up with commands.

ta Take commands from a text file.

table Read in columns from a table file to be used as parameters for other 'skyview' commands.

tr Tape read.

tw Tape write.

unix Exit program and go back to operating system.

ve Display a vector on a currently displayed image.

we Display a color table wedge.

write Write a zapped image to a file.

zp Zoom and pan a displayed image.

zap Zap (edit) pixels in a displayed image.

Insert a comment into the session history.

! Pass a command to unix to be executed from the shell.

? Display the command syntax of a command.

3 Skyview Server Mode

The 'server mode' facilitates straightforward interaction with **skyview** from a program. The **skyview** program can be invoked by another program, with the program directing **skyview** to execute the functions described in **Section 2** of this document.

To start up **skyview** in the server mode, use the **-server** flag[22]:

```
% skyview -server
```

There is no prompt in the server mode. **Skyview** simply waits for your program to begin sending commands, and executes each command, one at a time, in the order they are sent. The use of **skyview** commands in the server mode is no different from the interactive mode. The only difference is the output. Each command outputs a specifically formatted response for your program to deal with. For specific details on what a certain command returns, refer to either the online help (when **skyview** is **not** in the server mode) or the **Skyview Reference Manual** (printed version of online help).

Although the calling program can turn the mouse input on using the **set** command, initially the mouse is disabled, so the user can't accidentally click a button and invoke a function unintentionally.

The output is an ASCII structure, with the end of the structure followed by a <newline>. Elements within the structure may themselves be structures, making what might be called a "recursive structure".[23] The order of keyword-value pairs within the structure is undefined. Strings are delimited by white space and surrounded by quotation marks (""). Two quotation marks in a row indicates an empty string. Real values are delimited by white space but are not surrounded by quotation marks. At the top level the structure has the form:

```
[struct stat= status_string, cmd = command_name ]
```

For human readability, the command responses in this guide and in the **skyview** help pages are formatted with carriage returns and indenting:

```
[struct
  stat= status_string,
  cmd = command_name
]
```

A brief explanation of what the output means:

---

[22]See **Section 2.6.5** if you need more information on the use of **skyview** flags.

[23]See **Appendix F** for the full grammar of the **skyview** server mode.

command_name is the name of the command that was executed. It will always be spelled as the calling program spelled it.

status_string is one of two status messages: OK or ERROR.

Here is an example of what the **skyview** server mode returns when I use the **paint** command to display an image named **roseb1.fits**:

```
[struct
  stat="OK",
  cmd = "pa",
  filename = "/images/roseb1.fits",
  range_min_val = -6.27416e-14,
  range_max_val = 7.51656e-14,
  units = "W/M**2      "
]
```

The first line returned contains the **status_string** ('OK'). The second line returned contains the **command_name** ('pa'), as the calling program abbreviated it. The other lines contain **keyword** and **value** pairs.

Here is an example of what the **skyview** server mode returns when I use the **paint** command to display in image named **rosebud1.fits**, which does **not** exist:

```
[struct
  cmd = "pa",
  stat="ERROR",
  msg = "Unable to open image file /images/rosebud1.fits"
]
```

The first line returned is the **command_name** ("paint"), as abbreviated. The second line contains the **status_string** ("ERROR"). The third line contains a **message_string**, explaining the error.

Some **skyview** commands can return a list of information, depending on the type of input given. When the server mode is used, the information is returned in multiple structures. Each of the structures will have the keyword **state**. If the value of **state** is **"point"**, then the structure contains information for one item. If **state** = **"done"**, then the structure contains no new information, but instead indicates that the responses from the command are complete.

For example, when using the **graphics_examine** command to determine which marked positions fall within a defined area, the following response might be expected if two such marks were found:

```
[struct
  cmd = "gex",
  stat="OK",
  state = "point",
  coord =
    [struct
      coord_sys = "equatorial",
      equinox = "B1950.0",
      x = "329.564580",
      y = "63.750000"
    ],
  filename = "/Imaging/team/booth/images/f3.tbl",
  table_line = 208
]
[struct
  cmd = "gex",
  stat="OK",
  state = "point",
  coord =
    [struct
      coord_sys = "equatorial",
      equinox = "B1950.0",
      x = "329.992490",
      y = "63.245830"
    ],
  filename = "/Imaging/team/booth/images/f3.tbl",
  table_line = 215
]
[struct
  cmd = "gex",
  stat="OK",
  state = "done"
]
```

Since every program which calls **skyview** in the server mode must parse these recursive structures (a big chore), a library has been developed which contains C and Fortran callable functions enabling easy construction of client programs. The library, libsvc.a (Solaris), and its source files svclib.c, svclib.h, and svclibf.c (the Fortran binding) are in the contrib/server directory of the source distribution (not included in the binary distributions). In the same directory are sample programs and a sample Makefile.

The calls in the svclib library are as follows:

```
int   handle    = svc_init ( char *filepath );
int   status    = svc_send ( int handle, char *cmdstring );
char *retstr    = svc_receive ( int handle );
int   status    = svc_close ( int handle );

SVC  *svcstruct = svc_struct ( char *retstr );
void  status    = svc_free ( SVC *svcstruct );
char *val       = svc_val ( char *retstr, char *key, char *val )
```

where SVC is defined by

```
typedef struct
{
  int  nalloc;
  int  count;   /* number of key-val pairs */
  char **key;
  char **val;
} SVC;
```

Manual page descriptions of each of these functions are provided in Appendix F.

Program sample1.c from the distribution is reproduced here to show how easy it is to write such a program. This program paints an image and obtains a flux. Note that for brevity the program does no error checking. Program sample2.c is the same program with full error checking.

```
/* /imaging/config/beta/contrib/server/sample1.c */
#include <stdio.h>
#include <stdlib.h>
#include "svclib.h"

main(argc, argv)
int    argc;
char *argv[];
{
    int  handle;
    int status;

    char *retstr;
    char lat[25], lon[25], pixel_value[25], units[25];

    handle = svc_init("skyview -server");
```

```
    status = svc_send(handle, "pa /imaging/images/test/c1632b1");
    retstr = svc_receive(handle);

    status = svc_send(handle, "point 5h33m41.26s -6d53m48.9s");
    retstr = svc_receive(handle);
    svc_val(retstr, "center.coord.x", lat);
    svc_val(retstr, "center.coord.y", lon);
    svc_val(retstr, "center.pixel_value", pixel_value);
    svc_val(retstr, "center.units", units);
    printf("lat = %s  lon = %s  pixel_value %s %s\n",
      lat, lon, pixel_value, units);

    svc_close(handle);
}
```

Some commands are not available in the server mode. Each **skyview** command was carefully considered when it came to making such decisions during the design of the server mode. System functions can be called directly by the program just as easily (**ls**, for example). In others, it just doesn't make sense to expect the calling program to interpret the volume of output (as in **help**), and with some other functions, there are system security issues.

Here's a list of commands that are not available for use in while in the server mode:

al Define an alias based on functions or parameter definitions.

cd Change directories.

df Delete a file.

dir List the contents of the current directory.

fp Find the IRAS Skyflux or ISSA plate given certain coordinates.

help Display help on a given function.

he Display the header of an image.

ls List the contents of the current directory.

pause Pause and wait for a carriage-return before executing any more commands.

rep Replicate image pixels in either direction.

rew Rewind tape.

sleep Pause for a given length of time.

tr Tape read.

tw Tape write.

! Pass a command to unix to be executed from the shell.

? Display the command syntax of a command.