UNIVERSITY OF CALIFORNIA
LICK OBSERVATORY TECHNICAL REPORTS
NO. 33

THE VISTA USER'S GUIDE

Tod Lauer
Richard Stover
Donald Terndrup

Version 2
Santa Cruz, California
May 15, 1984

TABLE OF CONTENTS

Commands are listed in UPPER CASE.
Topics are listed in capitals and lower case

Definitions BASIC DEFINITIONS

The output from a CCD is called an IMAGE. An image is an array of numbers that are proportional to the amount of light that fell on the CCD during an exposure. The term for manipulation of these numbers is IMAGE PROCESSING. The VISTA program does this image processing by executing commands issued by the user.

A CCD is a solid-state chip that has a rectangular array of PIXELS, or 'picture elements'. A pixel has both a location and a value; it is the array of values that constitutes an image. The location of a pixel is specified by its ROW and COLUMN. Rows and columns are numbered continuously from some starting value, usually ZERO. (Thus an image with 400 rows and 350 columns will have the rows numbered from 0 to 399 and the columns numbered from 0 to 349). The starting row or column may be non-zero however: for example, the numbering may run from 100 to 499, instead.

It is convenient to refer to images by a label or name, rather than having to refer to each of the pixels individually. For this reason, the VISTA program has several BUFFERS (reserved places in memory) for the storage of images. VISTA can hold up to ten images at once; an image is referred to by the number of the buffer that holds it. The buffer numbers run from 1 to 10.

Also associated with each image buffer is a HEADER. The header is a list of numbers or character strings containing properties of the image: number of rows and columns, date of observation, etc. These are always associated with an image and are automatically copied or adjusted if the image itself is copied or adjusted.

A SPECTRUM is a linear array of numbers representing the amount of light at each wavelength interval from an astronomical object. A CCD used for spectroscopy does not produce a spectrum directly; rather, it records a two-dimensional image of the spread-out light behind the slit and grating. This image is later converted ('mashed') into a spectrum. VISTA provides twenty buffers (numbered 1 through 20) for storing spectra. Like images, spectra have headers. Do not confuse the spectrum buffers with the image buffers -- they are different.

Various VISTA programs produce data sets that are neither images nor spectra. The buffers for the storing of these data are not refered to by number, usually because there is only one of each type. These data can be saved on the disk (with the SAVE command), retrieved (with GET) or displayed (PRINT).

A very important feature of VISTA is PROCEDURES. These are lists of commands that are exectued as a group. There is a special buffer for storing procedures, and they may be saved on the disk for

repeated use. VISTA also has VARIABLES, which are symbolic names for numerical values. Using variables in procedures greatly expands the power of VISTA, by allowing the user to write complex programs.

Commands VISTA COMMANDS

VISTA works by calling various subroutines from a set of libraries. You tell VISTA what to do by entering commands, as you do when operating the VAX.

Commands come in several types. The simplest commands, such as 'Q' (which halts VISTA) work on no data and only operate in one way. Commands that manipulate images or spectra have an 'object' that the command operates on. You should always specify the object immediately after the command. An example of this is 'ZAP 1' which performs the ZAP operation (q.v.) on image number 1. More complicated commands have 'parameters' that control their operation. Parameters allow you to tailor the operation of the command to suit your needs. An example of a command with parameters is 'BOX 2 CC=100 CR=123 NC=100 NR=25'. Parameters can appear in any order on the command line.

There can be only one command on each line, and the command itself must come first. You must specify the object(s) and any parameters on the same line as the command. Some programs will ask you for additional information if you forget something; others will immediately return without doing anything other than printing an error message which will tell you what you did wrong. As an example of this, consider the SC command, which subtracts a constant from a specified image. Suppose you wanted to subtract 40.3 from each pixel in image 1. The full command is

```
SC 1 40.3
```

If you type only 'SC 1', VISTA will respond with a message 'CONSTANT TO SUBTRACT:_ '. When you enter the number and hit RETURN, the command will be executed.

Any character can appear on the command line, but characters after the first occurance of ! in a command are ignored. The ! is the VISTA 'comment character', providing a way to label individual commands. Here is an example of a command with a comment:

```
SC 1 40.3    ! Subtract 40.3 from image 1.
```

Commenting commands comes in very handy in procedures.

There are several types of parameters that can appear in a command:

1) Integer constants. These are used to denote buffers that store the images and spectra. They must not have a decimal point:

for example, '1' is an integer constant, while '1.' is not. (See 'SYNTAX' for other ways of specifying buffers.)

2) Real constants. These are numbers containing a decimal point (e.g., '3.14'), and are used as input to VISTA programs that require a numeric value as a parameter.

3) Variables. These are symbolic names for real constants. Symbolic names can generally be used wherever a real constant is required. The value of a variable is set by the SET command: for example 'SET BIAS=40.0' assigns the value 40.0 to the name BIAS. In the above example with the 'SC' command, you could have typed 'SC 1 BIAS' to subtract 40.0 from image 1, provided that BIAS had been defined, and its value set to 40.0. Variables can also be used to denote integers in some programs; the real number assigned to the variable name is converted to the nearest integer before execution. Some VISTA programs automatically set variables as a way of saving their output for use later.

4) Keywords. Certain commands have words as parametes. These control the operation of the program. 'HARD', for example, is a keyword of the 'PLOT' command. If 'HARD' is included in the command, the output is sent to a plotter; if it is not included, the output is sent to the user's terminal.

5) Keyword values. These are combinations of keywords and arithmetic constants, and are used to pass values to VISTA subroutines, where needed. Keyword values have the form 'WORD=value': i.e., a keyword immediately followed by an equal sign, immediately followed by a constant. The constant can be either explicit or symbolic, for example

SHIFT 1 DR=2.3 and SHIFT 1 DR=INCR

do the same thing, provided INCR has the value 2.3. You can also use arithmetic expressions following the '=' sign, as in 'SHIFT 1 DR=INCR+3.2'

6) Character strings. These are sequences of alpha-numeric symbols not interpretable as a number. Some VISTA commands use character strings as input. If a character string consists of more than one word (i.e., if there is a 'space' character in the middle), the entire string must be contained in single quotes: 'THIS IS A CHARACTER STRING WITH EIGHT WORDS'

VISTA takes the command line, and splits it up into several lists: a list of integers, a list of floating-point numbers, and a list of character strings. The programs in VISTA examine these lists to decide what to do.

More information about command syntax can be found in section II of this manual.

Files VISTA AND VAX DISKFILES

Many VISTA commands read from or write to the disk, and therefore require filenames. The program has defined several default directories and extensions for the various types of files that VISTA uses: for example, images are stored in one directory, spectra in a second, procedures in a third, etc. Unless you specify otherwise, VISTA will automatically search in certain directories for images, spectra, etc., and will assume that the files have standard extensions. You can view these default directories with the VISTA command PRINT DIR. It might be nice to save a copy of these directories for future reference: Use the command PRINT DIR >LP:

An example of a default directory is this: The WD command writes an image to disk with a filename specified in the command.

The command

WD 2 M15

writes the image in buffer 2 to the file [CCD]M15.CCD. The program wrote to the default directory [CCD] and used the extension .CCD. You can override the default locations any time: for example

WD 2 [DEMO]M15

writes to [DEMO]M15.CCD; the command

WD 2 M15.XYZ

writes to [CCD]M15.XYZ.

The default directories are established when the VISTA program is run by a subroutine called INITIAL. That subroutine has a standard list of directories and extensions that it loads into a common block for use by other subroutines. There is a way to change the default directories before running VISTA without changing any of the subroutines. This allows you to control the storage and retrieval of data in a somewhat cleaner way then overriding the default directories with each command. To change the default directories, use the DCL command DEFINE (symbol name) (directoryname). The symbols you can use are:

Symbol	Directory for ...
V$CCDIR	Images

V$PRODIR	Procedures
V$SPECDIR	Spectra
V$FLUXDIR	Flux calibrations
V$LAMDIR	Wavelength calibrations
V$COLORDIR	TV color files
V$DATADIR	Data files.

An example is: DEFINE V$CCDIR [DEMO.CCD]
This command changes the default directory for storing and retrieving disk images to [DEMO.CCD]. Remember to do this BEFORE you begin running VISTA. (It you want the same default directories all the time, you can put the DEFINE statements in your login file LOGIN.COM.)

The default directores and extensions may be changed while running VISTA with the command SETDIR.

VISTA can also run a designated procedure as it starts up. Use the symbol V$STARTUP to specify the FILE containing the procedure you want executed as the program begins. Typically this procedure will define aliases or set the values of variables. Read the section on procedures ('HELP PROCEDURE' if you are on a terminal) for more information.

Run HOW TO START VISTA

To start the VISTA program, log in on the VAX, and issue the command

RUN [CCDEV.BASE]VISTA

VISTA will respond with a 'welcome' message and a prompt (GO:). The prompt tells you that the program is ready to execute a command. Type a command you want, and hit RETURN. When the command is completed, the prompt will reappear. If you type a command that VISTA does not recognize, it will say so.

To save typing the above command every time you start the program, put

VISTA :== RUN [CCDEV.BASE]VISTA

into your login file [LOGIN.COM]. Then, the command 'VISTA' will start the program.

VISTA is a very large program, and your account must be specially configured to run it. See the system programmer to make sure your account is so configured.

HELP GETTING HELP

Form: HELP [subjects] [ALL] [output redirection]

where:

subjects	are the subjects for which information is desired,
ALL	produces a manual.

HELP is used to get information about commands and topics, or to produce a hardcopy manual for reference. A command is, of course, a process that can be executed directly by VISTA. A topic is a set of information about some aspect of VISTA. For example, this information appears under the command 'HELP', while introductory material about procedures is found under the topic 'Procedures'.

To get information on a particular command or topic, type HELP followed by the name of the command or topic. If you want help on several subjects, type them all on the command line. The words should be separated with blanks, as is usual with VISTA commands. Three examples are:

HELP SC	information on the command SC.
HELP SC ZAP FITSTAR	information on SC, ZAP, AND FITSTAR.
HELP Photometry	information on photometry.

The first line of information about commands contains a line beginning with 'Form: ', which spells out the syntax of that command in detail.' Following this are more detailed explanations of the workings of that command. These paragraphs form an example of an entry readable by the HELP command. Important: Keywords listed in square brackets (as [ALL], above) are OPTIONAL; they modify the operation of the command, if desired, but need not be specified under all circumstances.

The output of this program can be redirected. (The output redirection mechanism is described in the section 'REDIRECT' -- type HELP REDIRECT if you are on a terminal.) You can receive a printed instruction manual for VISTA, complete with page numbers, a table of contents and an index by typing 'HELP ALL >LP:'

The HELP command reads its information from a disk file, which has the commands and topics organized in groups, called 'subjects.' Typing HELP by itself produces a list of the subjects, called a 'menu.' You can then, without returning to the VISTA command mode, list the commands and topics under any given subject, or display information on any given command, just as you would if you were doing it directly with the HELP command in VISTA. When seeking information in the menu-mode, you do not need to type HELP: just list the

commands or topics that you want, separated by spaces (i.e., the HELP command without the word 'HELP').

Examples of the help command:

1)	HELP MASH	Sends the information on 'MASH' to your terminal.
2)	HELP MASH >MASH.XXX	Sends the information on 'MASH' to the file MASH.XXX
3)	HELP AI >LP:	Prints a copy of the AI helpfile on the line printer.

Q STOP VISTA

Form: Q

This command stops VISTA, returning the user to DCL command level. This command is executed immediately and can not be used in a procedure. The contents of all image and spectrum buffers are lost.

Syntax

This section reviews mechanism in the VISTA syntax which modify the operations of commands or allow repetition of commands with variation, and describes the very important procedure for using VISTA variables to specify image or spectrum buffers.

Output from various commands can be 'redirected', which means that the output can be sent somewhere else than your terminal. Type HELP REDIRECT for a description.

You can repeat previouly executed commands by prefixing the first (few) characters of that command with %. You can use the editor on the last command with the command EDIT. You can also define new commands in terms of old ones with ALIAS.

There are several ways to substitute or insert new keywords into a command line. The # construction is used to insert previously defined STRINGS into the command line. These strings are obtained from a list which is loaded with the command AF. New parameters may be added with % or on the end of synonym commands defined by ALIAS.

A very important substitution mechanism is the $ construct, which is used to specify IMAGE OR SPECTRUM buffers with variables. The rule for using $ is this: Any time you you would use an integer to specify a buffer, you can use a variable in the same place by prefixing the variable name with $. For example:

```
RD 3 [MYDIR]MYIMAGE
```

and

```
RD $IM'[MYDIR]MYIMAGE
```

does the same thing, provided that IM has the value 3.

This construction replaces the old formula for specifying buffers, which was I=name for images and S=name for spectra. Most programs require images or spectra but not both. For those few programs that handle both, you have to use S=name to specify a spectrum buffer and distinguish it from an image. An example of this is:

```
SET Q=3
PLOT %Q R=5              plots row 5 of IMAGE Q (= 3)
PLOT S=Q                 plots SPECTRUM Q (=3)
```

Redirect THE OUTPUT REDIRECTION MECHANISM

Many (but not all!) programs that produce large amounts of information may have their output REDIRECTED by the user. The output from these programs normally goes to the terminal, but instead can be written to a file or to the line-printer.

To redirect the output, you must use the '>' or '>>' constructions at the end of a valid command. They work like this:

(command) >filename	writes the output to the specified file, creating a new version of that file.
(command) >>filename	appends the output to the specified file. If that file does not exits, it is created.
(command) >LP:	sends the output to the printer.

Examples:

PRINT PHOT >FIRST.LIS	Prints the contents of a photometry file into the file 'FIRST.LIS'. The file will be located in your current default directory.
HELP MASH >LP:	Sends the help file for MASH to the lineprinter.
HELP MASH >>HELP.XXX	Appends the help information for MASH to the end of file HELP.XXX, if it exists. If it does not, the file is created.

HISTORY SHOW LAST SEVERAL COMMANDS

Form: HISTORY

The command HISTORY will show on your terminal the last several (typically 20) commands that you have executed. Use this to find commands that you want to repeat with the % substitution character.

% REPEAT A PREVIOUS COMMAND (AND MODIFY IT)

You can repeat a command in VISTA without typing it over with the % character. The % character comes before the command you want to repeat. It is also possible to modify a command while you repeat it.

The % comes in several forms:

--> Typing '%' by itself repeats the last command that you executed.
--> Typing '%n' where 'n' is a number repeats command n. The numbers of the command can be found by the HISTORY command.
--> Typing '%string' repeats the last command beginning with 'string'.

Suppose the HISTORY command gives the output

```
10      RD 1 [MYDIR]HD183143
11      MN 1
12      AI 2 1
13      PLOT 1 R=342 MIN=100.0 MAX=300.0
14      MASH 4 1 SP=(100,103)
15      PLOT S=4
```

```
Then        %       repeats command 15:   PLOT S=4
    %10     repeats command 10:   RD 1 [MYDIR]HD183143
    %MASH   repeats command 14:   MASH 4 1 SP=(100,103)
    %M      repeats command 14:   MASH 4 1 SP=(100,103)
```

Note the last example: The history mechanism will repeat the last command which begins with 'M'. If there are several commands which begin with the same letter, you have to supply enough of the command name after the % to uniquely specify the command to be repeated.

It is possible to modify a previous command using the % construction before executing the command again. There are several ways in which this modification can be done:

1) You can ADD words to the command by typing them after you enter '%command', provided that these words are not keyword values (EXPRESSION=VALUE) which are already present in the previous command.

2) You can modify keyword values which already exist in the previous command by simply repeating the keyword with a new value. For instance, if the old command was "PLOT S=1 PIXEL NOERASE HARD", you could repeat it for a different spectrum by just entering "%P S=2".

3) You can force a keyword value to be added to or subtracted from the previous command (and not substituted as described above) by preceding the keyword with a plus (+) or minus (-) sign.

Examples of command modification with %: Assume you wanted to repeat in various ways the command

```
PLOT S=1
```

which, we will assume is number 10 in the history list.

```
%10 HARD              does          PLOT S=1 HARD
%10 S=5                     does          PLOT S=5
%10 S=5 HARD                does          PLOT S=5 HARD
%10 XS=100 XE=200     does          PLOT S=1 XS=100 XE=100
```

Now suppose that command number 58 was

```
PLOT S=3 PIXEL HARD XS=100 XE=300 MAX=4096.0
```

Then

```
%58 -PIXEL -HARD      does          PLOT X=3 XS=100 XE=300 MAX=4096.0
```

```
ALIAS          DEFINING SYNONYMS FOR COMMANDS
UNALIAS        REMOVING SYNONYMS FOR COMMANDS
```

Form: , ALIAS [synonym] [command] [output redirection]
UNALIAS synonym

The ALIAS and UNALIAS commands are used to define and remove synonyms for commands. This can save you typing if you have several commands that have to be used repeatedly.

The command ALIAS (with no arguments) shows a list of synonyms of your terminal. The list can be loaded into a file or sent to the printer with the output redirection mechanism.

To define a command, use the full syntax ALIAS SYNONYM COMMAND. An example is

```
ALIAS T 'TV 1 1234.0 CF=NEWTHREE'
```

This defines a new command, T, which executes 'TV 1 ...' .
As is usual, if the command for which you are defining synonym is composed of more than one word, the entire command must be enclosed in quotes.

If you give only the new synonym and no not give the command, the program will ask for it. You can redefine a synonym at any time.

To execute the command that you have defined with ALIAS, type the synonym just as you would type any other command. You can add keywords to the command at the time of execution by typing them after the synonym. Using the example above, the command

```
T BOX=1
```

executes TV 1 1234.0 CF=NEWTHREE BOX=1. The command 'T' goes on the history list.

To remove a synonym, use the UNALIAS command. Type UNALIAS followed by the name of the synonym that you want to remove from the list. As an example, UNALIAS T will remove the definition of T that we have done above.

It is not possible to define a synonym in terms of another. For example, the sequence

```
ALIAS A  'TV 1 1234.0 CF=NEWTHREE'
ALIAS AA 'A BOX=1'
```

defines properly the command 'A' but NOT the command AA. (If you tried to execute AA the program will say 'AA is not a command.')

You can have synonyms defined automatically by putting ALIAS commands in the startup procedure. See the sections FILES and PROCEDURE for information about the startup procedure.

EDIT — EDIT LAST COMMAND

Form: EDIT

The command EDIT loads the last command (whatever it was) into a temporary file in your current directory, then runs a process which allows you to edit it with the EDIT/EDT editor. If you leave the editor with EXIT, the command is immediately executed. If you leave with QUIT, the command is not executed.

AF ASK FOR A LIST OF FILE NAMES OR OTHER VALUES

Form: AF file_buf [N=number] [CNT=var_name] ['PROMPT'] [IF=file] [OF=file]

where:

file_buf	is an integer (1 - 5) specifying which list is being defined.
N=number	tells VISTA that there will be 'number' entries in the list.
CNT=var_name	counts the number of entries in the list, and sets the value of the variable 'var_name' to be that count.

This command constructs lists of file names, defined variable names, or numeric constants which can be used by other commands. The list is read with the #character in a command line (see the section on '#'). AF can define up to five lists.

The normal procedure for making one of these lists is to first tell VISTA how many names are going to be entered (really the minimum number VISTA should expect) followed by the actual names. To make the command specific to your particular job you can supply a prompt which VISTA will type on the terminal before it asks for the number of names you are about to enter. If you don't supply the prompt VISTA will use its own. As described below, the N= and IF= optional keywords can be used to modify this normal procedure for entering file names.

The optional 'N=number' keyword allows you to specify on the command line the minimum number of file names VISTA should expect you to enter. The 'number' can be either a numeric (integer) constant, a defined symbol or the question mark. If a numeric value is supplied then VISTA will print the prompt (which would typically be something like 'ENTER OBSERVATION FILE NAMES') and then wait for you to enter the file names. At least 'number' names or constants must be entered. If N=? is supplied then the prompt will be issued and VISTA will wait for a single line of file names to be entered. Note that in this case the 'CNT=' keyword (see below) will be especially useful since VISTA will count the number of file names for you.

The optional keyword 'IF=file' can be used to direct VISTA to use the file 'file' as the source of the file names instead of your terminal. If the number of names in 'file' is less than the minimum number of names you told VISTA to expect, then VISTA will revert back to the terminal and wait for you to enter additional names. If you gave the 'N=?' keyword then VISTA will read 'file' until it encounters the end-of-file. Here again, the 'CNT=' keyword may be especially useful. VISTA uses [CCD.DATA]file.ASK unless you specify a different directory or file extension.

The optional keyword 'OF=file' can be used to direct VISTA to write its list of file names out into file 'file'. In a subsequent AF command you can use the 'IF=file' keyword to re-use the list of names. Also, the file can be edited directly with EDT.

The optional keyword value CNT=var_name provides a way for a procedure to find out how many file names were actually entered. The AF command will count the number of file names given and will set the variable var_name to that value. One popular use of var_name is to control a procedure DO loop, as in 'DO I=1,var_name'.

HOW TO SPECIFY FILE NAMES TO 'AF'

The AF command accepts a series of file names which are later doled out by the '#' pseudo-command. When the AF command asks for file names, they can be given in a number of formats. The simplest format is to enter each file name on a separate line. Alternately, multiple file names can be entered on the same line if the names are separated by at least one space.

More complex formats make it easy to specify a series of file names which have similar names. For instance, a series of files all of which have the same directory can be specified as: [DIR]FILE1,FILE2,FILE3,... where 'DIR' represents a directory name and 'FILE1,FILE2,FILE3...' represents a series of file names. Note that the files are separated (or connected, if you like) by commas. A space character or end-of-line will terminate the series. When the file names are extracted by the '#' pseudo-command the first is [DIR]FILE1 , the second is [DIR]FILE2 , and so on.

The directory in the previous example can be replaced by a physical device name. For instance, MTA0:FILE1,FILE2,... would be extracted as MTA0:FILE1, MTA0:FILE2, etc. If needed, a physical device name and a directory can be used together, as in: DRA0:[DIR]FILE1,FILE2,... .

A final file format is provided which makes it easy to specify a series of similar file names which have a numerical component in their names. A simple example of this format is: A##B,1-10 where 'A' and 'B' represent arbitrary (including possibly non-existent) character strings and the #'s represent place-holders which will be replaced by decimal digits when the actual file names are extracted. The numeric field, '1-10', specifies the range of values to be substituted for the #'s. This example produces ten file names: A01B, A02B, ..., A10B. Actually, any number of #'s (within reason) can be specified in the file name, as long as there are enough to satisfy the range of the numeric field.

Several additional parameters can be added to the last example. For instance, A##B,0-10/2 produces six file names: A00B, A02B, A04B, ... A10B. The number following the '/' specifies an increment for the numeric field. Individual files can also be excluded from the series as shown in A##B,0-99X5;43,52 . This represents 97 file names. The missing names are A05B, A43B, and A52B. Both the increment and exclusion list can be supplied, but the increment must be given first, as in A##B,20-40/4X36 which represents five names.

FILE NAME SUBSTITUTION CHARACTER

Form: #file_buf

where file_buf is an integer, 1 though 5.

This is a pseudo-command, used in conjunction with the AF command to obtain file names or numeric constants from a previously defined list, or to generate permuations in a pattern of filenames.

The first step in constructing and using a filename list is the AF command (q.v.), which can store up to five lists. The #file_buf command is then used to extract the names or constants from the stored lists. The index 'file_buf' specifies from which of the five sets the name or constant is to be extracted. Some typical uses of the #file_buf command are:

1)	RD 1 #1	reads a an image stored on the disk into buffer 1. The name of the file is at the top of list 1.
2)	AC 2 #3	adds a constant to image 2. The constant is at the top of list 3.
3)	SET Y=#5+X	sets the value of Y to be X added to the variable at the top of list 5.

Before VISTA executes each command it scans it to look for the #file_buf construction. If it is found then VISTA replaces it with the next file name or constant in the specified list. Only then is the command

executed. Any place that a file name, arithmetic constant, or symbol name can be used the #file_buf constuct can be used.

Input GETTING IMAGES OR SPECTRA
Output SAVING IMAGES OR SPECTRA

VISTA operates on images or spectra stored in internal buffers. Therefore, before you are able to analyze an image or a spectrum, you have to get it into one of the buffers from either a tape or the disk. You can with a VISTA command move an image from a VISTA buffer to tape or to the disk, or move one from tape or disk to a buffer. It is not possible to do a direct transfer between tape and disk. Spectra can be moved around in the same way.

Note the repeated use of the phrase 'with a VISTA command.' The commands for data transfer in VISTA are not quite as flexible as the VAX commands COPY and RENAME. You must use VISTA commands to read CCD data tapes produced at the observatory, or to enter the images into the buffers. Once an image or spectrum is produced, and after it is stored on the disk (again, with a VISTA command), you can use VAX commands to save these images on tape. Read the information in 'TAPES' for details and instructions.

The command-syntax for moving images around is simple: The transfer commands are all two-letters long. Commands beginning with 'R' READ images, while commands beginning with 'W' WRITE images. (Always remember that the reader or writer is the VISTA program!). If the second letter of the command is 'T', the command works with the tape drive, while if it is 'D', it works with the disk. Thus:

RD	reads an image from disk,
WD	writes an image to disk,
RT	reads an image from tape, and
WT	writes an image to tape.

The commands for moving spectra around are:

RS	reads a spectrum from disk,
WS	writes a spectrum to disk,
RTS	reads a spectrum from tape, and
WTS	writes a spectrum to tape.

Tape VISTA AND MAGNETIC TAPES

CCD data on magnetic tapes are generated either from the Mt. Hamilton data-taking systems, or by VISTA itself. There are two computers on the Mountain that control the CCD: the LSI-11 and the PDP-8. These write the images in different formats, but VISTA can read either. When running VISTA, you do not need to specify any tape formats: the program can tell automatically which way the tape was written.

If you plan to use a magnetic tape in a session with VISTA, you may mount it BEFORE you begin running VISTA. Use the VAX command 'MOUNT/FOREIGN MTO' or 'MOUNT/FOREIGN MT1' to mount the tape on one of your two drives. Or you can mount the tape while running VISTA by using the VISTA 'MOUNT' command.

The VISTA command WT (WTS) will write an image (spectrum) to tape in the same format as the mountain computers. This allows you to save images on tape for later use, and to read those images with VISTA. There is another way to save images on tape: if an image or a spectrum is written to a disk file, you can transfer it to tape, preserving its VAX format. To do this, mount the tape with the MOUNT command (not MOUNT/FOREIGN). Then use the COPY command just as you would to save any other VAX file to tape:

```
COPY filename MTO:filename                (or MT1:filename)
```

To bring it back to disk, mount the tape and use the VAX command

```
COPY MTO:filename filename                (or MT1:filename)
```

These commands must be executed BEFORE you begin running VISTA. When starting a new tape, you must use the VAX command INITIALIZE.

Be careful! Remember what format your tapes are written! Do not mix formats on a single tape!

Another reason to be careful is this: The data on FITS format tapes are stored as 16-bit signed INTEGERS, so data values outside the range -32767 to 32768 will not be stored properly. If you have data values outside this range, you will have to scale the image as you write it. The tape writing commands have a keyword 'AUTO' which will determine the best scaling for you.

MOUNT MOUNTNG MAGNETIC TAPES

Form: MOUNT [UNIT=1]

MOUNT mounts a magnetic tape on unit 0 (if no keywords are given) or on unit 1 (if the UNIT=1 keyword is included). This is equivalent to a MOUNT/FOREIGN in DCL, and allows you to mount tapes while running VISTA. Also see DISMOUNT.

At Santa Cruz, unit 0 is MTA0: and unit 1 is MSA0:
Examples:

```
MOUNT           mounts a tape on unit 0.
MOUNT UNIT=1    mounts a tape on unit 1.
```

DISMOUNT DISMOUNT A MAGNETIC TAPE

Form: DISMOUNT [UNIT=n]

DISMOUNT is used to dismount a magnetic tape that was mounted with the VISTA MOUNT command. It cannot be used to dismount tapes that were mounted with the DCL command MOUNT/FOREIGN before VISTA was run.

DISMOUNT can be used with no arguments if you have only one tape mounted. It dismounts that tape regardless of the tape drive it is on. If more that one tape is mounted, you must specify the unit number of the tape with the UNIT keyword.

TDIR PRINT A DIRECTORY OF IMAGES STORED ON A TAPE

Form: TDIR [comment] [UNIT=n] [output redirection]

where:

comment (character string) is a comment or label to appear at the top of each page in the list, and

UNIT=n tells VISTA which tape drive to use. UNIT=0 reads from MT0 and UNIT=1 reads from MT1. (MT0 = MTA0:, MT1 = MSA0: at Lick)

This command will print out a paged listing of the images stored on a tape. The object names and observing parameters are printed. The optional comment will be printed at the top of each page. If the comment has more than one word in it, the entire comment must be enclosed in single quotation marks.

Examples:

1) TDIR — print a directory of the images on the currently mounted tape, assuming that only one is mounted. The output is sent to your terminal.

2) TDIR >LP: — does the same thing as the above, but sends the listing to the lineprinter.

3) TDIR UNIT=1 — print a directory of the images on the tape that is mounted on MT1. You need to do this only if you mounted two tapes

4) TDIR 'TAPE FIVE' — print a directory of images on the currently mounted tape, with the label TAPE FIVE at the top of each page.

Use the commands PRINT IM or PRINT SPEC to give a list of the images or spectra on the disk.

RT READ AN IMAGE FROM TAPE

Form: RT dest image_number [UNIT=n]

where:

dest	(integer or $ constuct) is the buffer where the image will be stored,
image_number	(integer or variable or $ construct) is the number of the image on the magnetic tape, and
UNIT=n	tells VISTA to read from drive n (integer).

RT will read an image from either a standard PDP-8 style tape or a FITS style tape as produced by the LSI-11's. You do not need to specify the style of the tape: VISTA reads either style automatically. Image numbers on tape start with image 1. If 'image_number' is not included on the command line then VISTA will ask for the image number when the command is executed. If you are using more than one tape, you can specify the unit with the 'UNIT=' keyword. Unit 0 is MTA0: and unit 1 is MSA0: here at Lick Observatory.

Examples:

1)	RT 1 3	reads the third image on the magnetic tape into buffer 1.
2)	RT 1 $TNUM	reads image number TNUM on the magnetic tape (where TNUM is a variable) into buffer 1.
3)	RT $N 5	reads the fifth image on the magnetic tape into buffer N, where N is a variable.
4)	RT 2 5 UNIT=1	reads the fifth image on the magnetic tape that is mounted on MSA0: into buffer 2

WT WRITE AN IMAGE TO TAPE

Form: WT source [PDP8] [ZERO=z] [SCALE=s] [AUTO] [UNIT=n]

where:

source	(integer or $ construct) is the number of the buffer holding the image to be written,
PDP8	tells VISTA to write in PDP-8 format,
ZERO=z	adjusts the zero level of the image to z

SCALE=s	scales the image by s (const. or variable), and
AUTO	computes best ZERO and SCALE to preserve the best precision on the data tape,
UNIT=n	writes to tape unit n (integer).

This command will write an image to tape in the standard LSI-11 FITS style. Images are always written at the end of the tape. (To initialize a tape, use the 'INT' command.) When the image has been written out VISTA will print on the terminal the tape image number into which it was written. VISTA will write your tape in the old PDP-8 style if you give the 'PDP8' keyword. This should be done only if the tape already has PDP-8 style images, since you should not mix the two styles.

The image pixel values written to the tape are scaled from the values contained in the image buffer by the relation:

(tape value) = ((buffer value) + z) * s

where initially z=0.0 and s=1.0. These values can be changed with the 'ZERO' and 'SCALE' keywords or will be computed for you if you use the 'AUTO' keyword. If you write a FITS style tape then these values are recorded in the image header, and your image will be restored to the original values when read back with the RT command. (Some integer truncation will always occur.)

Examples:

1)	WT 1	writes the image in buffer 1 to tape.
2)	WT $N	writes the image in buffer N to tape, where N is a variable.
3)	WT 1 PDP8 AUTO	writes the image in buffer 1 to tape in the PDP8 format. The data on tape is scale to span the full range from 0 to 4095.
4)	WT 1 UNIT=1	writes the image in buffer 1 to the tape that is mounted on unit 1.
5)	WT 1 SCALE=2.0 ZERO=0.0	writes the image in buffer 1 to tape, scaling the pixel values on the tape by a factor of 2.

RD READ AN IMAGE FROM A DISK FILE

Form: RD dest filename

where:

dest	(integer or $contruct) is the buffer in which the image will be stored, and
filename	(character string) is the name of the diskfile holding the image.

The image will be read from the default directory for images (which is either [CCD] or a directory specified by the DCL symbol V$CCDIR) unless you specify otherwise on the command line. The file is assumed to have extension .CCD unless you specify otherwise.

Some examples of RD are:

1)	RD 1 M92	reads [CCD]M92.CCD to buffer 1.
2)	RD 7 [DEMO]M92	reads [DEMO]M92.CCD to buffer 7.
3)	RD 3 M92.XYZ	reads [CCD]M92.XYZ to buffer 3.
4)	RD $N M92	reads [CCD]M92.CCD to buffer N, where N is a variable.

WD WRITE AN IMAGE TO A DISK FILE

Form: WD source filename

where:

source	(integer or $ construct) is the buffer holding the image to be written, and
filename	(character string) is the name of the file into which the image will be written.

WD writes the image contained in buffer 'dest' to the specified file. The image will be written to the default directory for images (which is either [CCD] or a directory specified by the DCL symbol V$CCDIR) unless you specify otherwise on the command line. If you do not give a filename, the program will ask you for it. The file will be written with extension .CCD unless you specify otherwise.

The files stored by the WD command count against the quota of the account you are using when running VISTA.

Some examples of WD are:

1) WD 1 M92 writes buffer 1 to [CCD]M92.CCD.

2) WD 7 [DEMO]M92 writes buffer 7 to [DEMO]M92.CCD.

3) WD 3 M92.XYZ writes buffer 3 to [CCD]M92.XYZ.

4) WD $N M92 writes buffer N to [CCD]M92.CCD, where N is a variable.

RTS READ A SPECTRUM FROM TAPE

Form: RTS dest spec_number [UNIT=n]

where:

dest (integer or $ constuct) is the spectrum buffer where the image will be stored,
spec_number (integer or variable or $ construct) is the number of the FITS file on the magnetic tape, and
UNIT=n tells VISTA to read from drive n (integer).

RTS will read a spectrum from a FITS style tape as produced by the LSI-11's and VISTA. A spectrum is defined as a FITS file in which there is only one axis coordiante associated with the data. (In the FITS header NAXIS is 1). FITS file numbers on tape start with image 1. If 'spec_number' is not included on the command line then VISTA will ask for the file number when the command is executed. If you are using more than one tape, you can specify the unit with the 'UNIT=' keyword. Unit 0 is MTA0: and unit 1 is MSA0: here at Lick Observatory.

Examples:

1) RTS 1 3 reads the third file on the magnetic tape into spectrum buffer 1.

2) RTS 1 $TNUM reads file number TNUM on the magnetic tape (where TNUM is a variable) into spectrum buffer 1.

3) RTS $N 5 reads the fifth file on the magnetic tape into spectrum buffer N, where N is a variable.

4) RTS 2 5 UNIT=1 reads the fifth file on the magnetic tape that is mounted on MSA0: into spectrum buffer 2.

WTS WRITE A SPECTRUM TO TAPE

Form: WTS source [ZERO=z] [SCALE=s] [AUTO] [UNIT=n]

where:

source	(integer or $ construct) is the number of the buffer holding the spectrum to be written,
ZERO=z	adjusts the zero level of the image to z
SCALE=s	scales the image by s (const. or variable), and
AUTO	computes best ZERO and SCALE to preserve the best precision on the data tape,
UNIT=n	writes to tape unit n (integer).

This command will write a spectrum to tape in the standard LSI-11 FITS style. Spectra are always written at the end of the tape. (To initialize a tape, use the 'INT' command.) When the spectrum has been written out VISTA will print on the terminal the tape file number into which it was written.

The spectrum pixel values written to the tape are scaled from the values contained in the image buffer by the relation:

(tape value) = ((buffer value) + z) * s

where initially z=0.0 and s=1.0. These values can be changed with the 'ZERO' and 'SCALE' keywords or will be computed for you if you use the 'AUTO' keyword. These values are recorded in the image header, and your image will be restored to the original values when read back with the RT command. (Some integer truncation will always occur.) If you use the 'AUTO' keyword the values on tape will be scaled to have the range +32767 to -32767.

Examples:

1)	WTS 1	writes the spectrum in buffer 1 to tape.
2)	WTS $N	writes the spectrum in buffer N to tape, where N is a variable.
3)	WTS 1 AUTO	writes the spectrum in buffer 1 to tape, scaling the data on tape to span the full range from -32767 to +32767.
4)	WTS 1 UNIT=1	writes the spectrum in buffer 1 to the tape that is mounted on unit 1.

5) WTS 1 SCALE=2.0 ZERO=0.0 writes the spectrum in buffer 1 to tape, scaling the pixel values on the tape by a factor of 2.

RS READ A SPECTRUM FROM A DISK FILE

Form: RS dest filename

where:

dest (integer or $ contruct) is the buffer in which the spectrum will be stored.

filename (character string) is the name of the diskfile holding the spectrum.

RS reads the spectrum contained in the specified file into buffer 'dest'. The image will be read from the default directory for spectra (which is either [CCD.SPEC] or a directory specified by the DCL symbol V$SPECDIR) unless you specify otherwise on the command line. The filename is assumed to have extension .SPC, unless you say otherwise.

Some examples of RS are:

1) RS 1 HD183143 reads [CCD.SPEC]HD183143.SPC to buffer 1.

2) RS 7 [DEMO]HD183143 reads [DEMO]HD183143.SPC to buffer 7.

3) RS 3 HD183143.XYZ reads [CCD.SPEC]HD183143.XYZ to buffer 3.

4) RS $N HD183143 reads [CCD.SPEC]HD183143.SPC to buffer N, where N is a variable.

WS WRITE A SPECTRUM TO A DISK FILE

Form: WS source filename

where:

source (integer or $ construct) is the buffer holding the spectrum to be written.

filename (character string) is the name of the file into which the spectrum will be written.

WS writes the spectrum contained in buffer 'dest' to the specified file. The image will be written to the default directory for images (which is either [CCD.SPEC] or a directory specified by the DCL symbol V$SPECDIR)

unless you specify otherwise on the command line. The filename is written with extension .SPC, unless you say otherwise.

The files stored by the WS command count against the quota of the account you are using when running VISTA.

Some examples of WS are:

1)	WS 1 HD183143	writes buffer 1 to [CCD.SPEC]HD183143.SPC.
2)	WS 7 [DEMO]HD183143	writes buffer 7 to [DEMO]HD183143.SPC.
3)	WS 3 HD183143.XYZ	writes buffer 3 to [CCD.SPEC]HD183143.XYZ.
4)	WS $N HD183143	writes buffer N to [CCD.SPEC]HD183183.SPC, where N is a variable.

COP COPY AN IMAGE FROM ONE BUFFER TO ANOTHER

Form: COP dest source [BOX=n]

where:

dest	(integer or $ construct) is the buffer where the new image will be stored,
source	(integer or $ contsruct) is the original copy, and
BOX=n	tells the program to copy only the part of the image that is in box 'n'.

COP copies the image in buffer 'source' to the buffer 'dest'. The header associated with the source image is also copied. The syntax of the copy command is:

COPY into (dest) the image (source).

Examples:

1)	COP 2 1	copies the image in buffer 1 to buffer 2, along with image 2's header and label.
2)	COP $B $A	copies the image in buffer A to buffer B, where A are B are variables.
3)	COP 1 2 BOX=7	copies the segment of image 2 that is

with in box 7 into image buffer 1. The size of the new image will be the size of the box. See BOX for more details.

COPS COPY A SPECTRUM FROM ONE BUFFER TO ANOTHER

Form: COPS dest source

where:

dest (integer or $ construct) is the buffer where the new spectrum will be stored.

source (integer or $ contstruct) is the original copy

COPS copies the spectrum in buffer 'source' to buffer 'dest'. The header of spectrum 'source' is also copied. Any spectrum already in buffer 'dest' is destroyed. Note that the destination spectrum comes FIRST. The syntax of the command is

COPY into spectrum (dest) the spectrum (source).

Examples:

1) COPS 2 1 copies the spectrum in buffer 1 to buffer 2, along with spectrum 1's header

2) COPS $B $A copies the spectrum in buffer A to buffer B, where A are B are symbolic constants.

BUF DISPLAY IMAGE HEADER INFORMATION

Form: BUF [buffers] [FULL] [FITS[=param]] [output redirection]

This command shows a brief summary of the header information for the images contained in the image buffers. If specific buffers are specified, the program lists information for only those buffers. Otherwise, all buffers with images in them are listed. If the keyword FULL is specified, a longer listing is given for each buffer. If the keyword FITS=param is given then the literal text of the FITS header is printed for the FITS header parameter 'param'. If you use just the keyword FITS (no specified parameter) and you use the FULL keyword then all of the FITS header parameters will be shown.

Examples:

1) BUF Give a brief list of all buffers containing images on the terminal.

2) BUF >LP: Do the same, but send the output to the lineprinter.

3) BUF FULL Give a long listing of the image buffers.

4) BUF 3 4 8 FULL Give a long listing of buffers 3, 4, and 8.

5) BUF 3 FULL FITS List all the individual FITS header parameters and values for buffer 3.

BUFS DISPLAY SPECTRUM HEADER INFORMATION

Form: BUFS [buffers] [FULL] [FITS[=param]] [output redirection]

This command shows a brief summary of the header information for the spectra contained in the spectrum buffers. If specific buffers are specified, the program lists information for only those buffers. Otherwise, all buffers with spectra in them are listed. If the keyword FULL is specified, a longer listing is given for each buffer. If the keyword FITS=param is given then the literal text of the FITS header is printed for the FITS header parameter 'param'. If you use just the keyword FITS (no specified parameter) and you use the FULL keyword then all of the FITS header parameters will be shown.

Examples:

1) BUFS Give a brief list of all buffers containing spectra on the terminal.

2) BUFS >LP: Do the same, but send the output to the lineprinter.

3) BUFS FULL Give a long listing of the spectrum buffers.

4) BUFS 3 4 8 FULL Give a long listing of buffers 3, 4, and 8.

5) BUFS 3 FULL FITS List all the individual FITS header parameters and values for buffer 3.

INT START OR EDIT AN OUTPUT DATA TAPE

Form: INT image [UNIT=n]

where:

image (integer or $ construct) is the number where the next image written on tape will have.

UNIT — tells VISTA which tape drive to use. UNIT=0 writes on MT0 and UNIT=1 writes on MT1. (MT0 = MTA0: and MT1 = MSA0: at Lick)

This command will write an end of volume mark in front of the specified 'image' file number, hence deleting THAT IMAGE AND ALL PAST IT on the tape. If you want to write an image at the end of an old tape, you do not need to use the INT command. If an image number is not specified, it is assumed that the tape is new and is to be initialized for writing. In all cases INT will ask you to confirm your command after it has positioned the tape, but before it writes to it.

Examples: Suppose there are 20 images on the tape. If the next image written on that tape will be number 21, then you do not need to use the INT command.

1) INT 20 — deletes the last image on tape. The next image written will be number 20.

2) INT 10 — deletes the last eleven images on tape (10 - 20). The next image written will be number 10.

3) INT — deletes ALL images on the tape.

CLOSE — CLEAR AND DISCONNECT AN IMAGE BUFFER

Form: CLOSE dest

where: dest (integer or $ construct) is the image to be deleted.

This command will clear and disconnect the image in buffer number 'dest' from VISTA. The memory needed to hold an image was not assigned when the VISTA program was compiled; rather, it is requested from the VAX when a new image is created. This run-time memory is termed 'virtual memory.' The CLOSE command is used for freeing this memory when the image is no longer needed or when VISTA has run out of memory for setting up other image buffers. If you want to save the image, it should be written to disk or tape first with the WD or WT commands before closing out the buffer.

Examples:

1) CLOSE 7 — deletes image 7

2) CLOSE $Q — deletes image Q, where Q is a variable. (This form is helpful in procedures.)

Suggestions:

If you CLOSE images that you will no longer need, the VISTA program may run faster, especially if there are many users on the VAX. If you wish to close several files at once, it is helpful (but not necesary) to close them in the opposite order from the order in which they were created.

The VISTA may sometimes respond with the message 'No virtual memory available' when trying to create an image. To correct this, close an image or two. This frees up more run-time memory, allowing you to proceed.

CH CHANGE IMAGE NAME

Form: CH dest new_name

where:

dest — (integer or $ construct) is the number of the image that is having its label changed, and

new_name — (character string) is the new label.

This command allows you the change the image label or name of the image held in buffer number 'dest'. The new object name can be given on the command line as shown or it can be supplied when the command executes. If it is supplied on the command line then it must be enclosed in quotes if it is more than one word long. If the new name is not given on the command line then VISTA will type the current object name and ask for the new name. If you respond with only a carriage return then the old name is left unchanged. To force a blank object name you must enter at least one space character before the carriage return. The image name is stored in upper case letters only.

Examples:

1) CH 1 HD183143 — changes the name of image 1 to 'HD183143'.

2) CH 1 'TEST IMAGE 2' — changes the name of image 1 to 'TEST IMAGE 2'.

3) CH $R 'NEW LABEL' — changes the name of image R (where R is a variable) to 'NEW LABEL'.

4) CH 1 — changes the name of image 1. The old name is printed, and VISTA asks you for

the new name.

CHS CHANGE SPECTRUM NAME

Form: CHS dest new_name

where:

dest (integer or $ construct) is the number of the spectrum that is having its label changed, and

new_name (character string) is the new label.

This command allows you the change the spectrum label or name of the spectrum held in buffer number 'dest'. The new object name can be given on the command line as shown or it can be supplied when the command executes. If it is supplied on the command line then it must be enclosed in quotes if it is more than one word long. If the new name is not given on the command line then VISTA will type the current object name and ask for the new name. If you respond with only a carriage return then the old name is left unchanged. To force a blank object name you must enter at least one space character before the carriage return. The spectrum name is stored in upper case letters only.

Examples:

1) CHS 1 HD183143 changes the name of spectrum 1 to 'HD183143'.

2) CHS 1 'TEST SPEC 2' changes the name of spectrum 1 to 'TEST SPEC 2'.

3) CHS $R 'NEW LABEL' changes the name of spectrum R (where R is a variable) to 'NEW LABEL'.

4) CHS 1 changes the name of the spectrum in buffer 1. The old name is printed, and VISTA asks you for the new name.

GET GET DATA FILE
SAVE SAVE DATA FILE

Form: SAVE [data_keyword=[filename]] [other data keywords]
GET [data_keyword=[filename]] [other data keywords]

where:

data_keyword	is a word specifying the type of data being saved or get (APER, MASK, PHOT, or PROFILE),
filename	is the name of the file that is holding the data (GET) or will hold the data (SAVE), and
other data keywords	are used if you are getting or saving more than one data file at a time.

The SAVE and GET commands are used for writing out results produced by other VISTA routines to disk for future use by VISTA or other programs, and getting them back again. Various VISTA commands will generate reduced data which you may want to save for later analysis. To SAVE or GET results from a specific program, specify that program's keyword and a file name. If no file name is given, VISTA will ask you for one. More than one keyword can be specified at a time. By default, all files are kept in [CCD.DATA] (or the directory specified by the DCL symbol V$DATADIR) with the file extensions given below. These can be overriden if desired. To examine the current results that VISTA is storing, use the PRINT command.

At this time VISTA provides for the storage of the following data types:

Keyword:		
	APER=name	Aperture photometry file 'name(.APR)'
	MASK=name	Photometry mask file 'name(.MSK)'
	PHOT=name	Stellar photometry file 'name(.PHO)'
	PROFILE=name	Surface photometry file 'name(.PRF)'

Examples:

1)	GET PHOT=ORION	loads the photometry file [CCD.DATA]ORION.PHO to VISTA

2) SAVE MASK=MASK5 — writes the VISTA mask file to [CCD.DATA]MASK5.MSK

3) GET PHOT=ORION MASK=MASK5.ABC — loads [CCD.DATA]ORION.PHO and [CCD.DATA]MASK5.ABC

PRINT PRINT DATA FILES, SPECTRA, OR IMAGE SUBSECTIONS

Form: PRINT [data keywords] [output redirection]

where:

data keywords — specify which information is printed

This command will print out a formatted listing of reduced data files, spectra or image subsections. The output from PRINT appears on your terminal; to send it to the printer use the '>LP:' construct. PRINT recognizes the following keywords:

1) 'image' BOX=n — Print out the pixel values of 'image' in the subsection specified by box 'n' and generate basic statistics on the pixels.

Examples: PRINT 1 BOX=2
PRINT 2 BOX=7 >LP:
PRINT $Q BOX=4 >IMAGESEC.DAT

2) BOX — Print out the sizes, centers, and origins of all boxes defined.

Forms: PRINT BOX
PRINT BOX >LP:

3) S=n [COLUMN] — Print out the pixels in spectrum 'n'. Wavelengths are also printed if the spectrum is on a wavelength scale. The COLUMN keyword produces a single column output which is useful for MONGO plotting package input.

EXAMPLES: PRINT S=5
PRINT S=4 >LP:
PRINT S=2 COLUMN >SPEC.DAT

4) PROFILE — Print out the surface photometry profile.

Examples: PRINT PROFILE

PRINT PROFILE >LP:

5) PHOT — Print out the stellar photometry results.

Examples: PRINT PHOT
PRINT PHOT >PHOTLIST.DAT

6) APER — Print out the aperture photometry results.

Examples: PRINT APER
PRINT APER >>APERLIST.DAT

7) DIR — Print the default directories.

Examples: PRINT DIR
PRINT DIR >LP:

8) IM — Print headers for images in the default image directory. (Use PRINT DIR) to see which directory this is)

Examples: PRINT IM
PRINT IM >IMLIST.DAT

9) SPEC — Print headers for spectra in the default spectrum directory. (Use PRINT DIR) to see which directory this is)

Example: PRINT SPEC

10) LINEID — Print wavelength v. pixel identifications obtained with the LINEID command.

Example: PRINT LINEID
PRINT LINEID >LP:

11) VAR — Print list of all defined variables.

Examples: PRINT VAR
PRINT VAR >VARLIST.DAT

SETDIR SET DEFAULT DIRECTORIES AND FILE EXTENSIONS

Form: SETDIR code [DIR=directory_name] [EXT=extension]

where: code specifies which directory is being set or changed
DIR= specifies a directory for the type of object indicated by the code.
EXT= gives the extension for files in the default directory

SETDIR sets the default directories and extensions of files storing images, spectra, color maps, etc. You can see the default values with the command PRINT DIR. See the section FILES (type HELP FILES if you're on a terminal) for information about default directories and extensions.

The DIR word gives the default directory for the type of object specified by the code, and the EXT word gives the extension for that type of object. An example of a default extension is that for '.CCD' for images. An example of a default directory is '[CCD.SPEC]' for spectra. You must specify either the directory or the extension or both with SETDIR. If the extension is not blank, it must include a period as its first character: For example, '.XYZ' is a valid extension, while 'FLK' is not.

The 'code' gives the directory which is being set or changed. The code is derived from the type of object in the directory you are specifying. You must type at least the first two letters of the code:

Object	Code	Abbreviation
Images	IMAGES	IM
Spectra	SPECTRA	SP
Procedures	PROCEDURES	PR
Flux calibration files	FLUX	FL
Wavelength files	WAVE	WA
Color maps	COLOR	CO
Data files	DATA	DA

Examples:

Suppose you see with PRINT DIR that the default directory for CCD spectra is [CCD.SPEC] and the default extension is '.SPC'

1) SETDIR SP DIR=[MYDIR.SPEC] changes the default directory to [MYDIR.SPEC]

2) SETDIR SP EXT=.XYZ changes the default extension to '.XYZ'

3) SETDIR SP EXT=.XYZ DIR=[MYDIR.SPEC] changes both the directory and extension at one time.

Television INTRODUCTION TO THE AED TELEVISION SYSTEM

The principle method for displaying images is the AED television system. The AED (the large, flat box with a keyboard in the VAX room) has three components:

1) memory large enough to hold a 512 by 512 image,
2) a set of instructions for translating pixel value into color (a 'color map'), and
3) a cursor, allowing the user to visually select segments of an image.

An image is loaded into the AED with the 'TV' command. The color map may be changed with the COLOR command.

There are several commands which automatically operate with the image stored in the television. Such commands have no image specifier on the command line. Probably the most important of these is the 'ITV' command, which allows you to interactively examine an image in various ways. When the AED is used with these commands, several keys on its keyboard are defined so that they perform functions when struck. Any AED-interactive command (including TV itself) has the following keys defined:

I	zooms in, showing a smaller segment of the picture. The center of the new picture is the location of the cursor when the 'I' key was struck.
O	zooms out, showing a larger segment of the picture. The center of the new picture is the location of the cursor when the 'O' key was struck.
P	pans (i.e., moves) to another part of the picture. The center of the new picture is the location of the cursor when the 'P' key was struck.
R	restores the picture to the configuration it was in when it was loaded.
E	exits the interactive program, returning the user to the VISTA command mode.

The various AED interactive routines also define other keys, which perform functions unique to that program. Normally, the AED 'H' key will list on your terminal the definitions for that program. Examples of these interactive routines are ITV, MARKSTAR, PSF, and FITSTAR. Besides the joystick, you never need to touch any of the AED keys, except the letters A-Z and 0-9.

To operate the TV, (1) turn on the AED: the switch is in the far left of the back pannel. (2) turn on the television monitor: the switch is to the right of the screen. Remember to turn off both systems when finished.

TV WRITE AN IMAGE TO THE AED VIDEO DISPLAY

Form: TV sourc [span] [zero] [L=span] [Z=zero] [BOX=n] [CF=xxx] [NOLABEL] [LEFT] [RIGHT] [NOERASE] [OLD] [BW]

where:

sourc	(integer or $ contruct) is the image to be displayed,
span or L=	set the span level for the color map,
zero or Z=	set the zero level,
BOX=n	displays the part of the image in box 'n',
CF	specifies the color map,
NOLABEL	suppresses labeling of the image,
LEFT, RIGHT	are used for blink comparison,
NOERASE	prevents the TV from erasing the previous image when displaying a new one (used for blinking),
OLD	displays using previosly defined parameters,
BW	displays the image in black and white.

This routine will display the image 'sourc' on the AED video display. The pixels are translated for display by first subtracting the 'zero' level and loading the result modulo the intensity 'span' range. By default, 'zero' is '0' and 'span' is four times the image mean. (Important: TV does not calculate the mean, so you must use the MN command first if you want to use the default span.) These are default values good for low background astronomical images, but you should experiment with the zero and span, since careful choice of these parameters can enhance or suppress various features present in the image. The 'span' and 'zero' values can be specified either as an ordered pair of real numbers (notice that 'span' precedes 'zero') or with the L and Z keywords. Pixels less than the zero level are displayed as zeros.

CF=filename loads the color map that translates the pixel values into color. 'BW' loads a black and white color map. Once the map is loaded, it remains in use until changed by the COLOR or ITV commands, or is reloaded with the CF keyword. The available color maps are:

CF=WRMB	(thought by many to be the most useful)
CF=RAIN	(the color distribution in a rainbow)
CF=IBW	(inverse black and white)

You can define new color maps with the COLOR command.

The keyword 'OLD' tells the routine to re-use the last set of parameters. The 'BOX=n' keyword specifies that only the image subsection in box 'n' (see BOX command) is to be displayed.

The displayed image is shown with its header, a color bar showing the translation from values to color, and a set of labeled tick

marks to mark the image's rows and columns. This annotation can be suppressed with the NOLABEL keyword. The routine automatically zooms into the image center so that the it fills as much of the screen as possible. Once the image is displayed, its parameters are held in a common block so that other routines can be called from VISTA to interact with it.

The AED display can be used as a digital blink comparitor, and the LEFT and RIGHT keywords have been provided for this purpose. When either are given the AED screen is divided into equal left and right halves and the image is placed in the specified half. When the LEFT keyword is given the entire screen is cleared (unless the NOERASE keyword is given) before the image is put up on the screen. An example of the normal sequence for loading two images for blink comparison is shown below. This option is very useful for examining fields for variable stars as well as comparing processed and unprocessed images.

The AED can display a maximum of 512 rows and 512 columns, or 2 sets of 512 rows and 256 columns if the blink option is used. Images larger than this will be compressed before being displayed.

Examples:

1)	MN 1 TV 1 CF=WRMB	loads image 1 into the AED, setting the span to be 4 times the image mean.
2)	TV 3 100. 30.0	loads image 3 into the AED, setting the span to be 100 and the zero to be 30.
3)	TV 3 L=100. Z=30.	does the same thing as example 2
4)	TV 7 NOLABEL	loads image 7, suppressing the labeling
5)	TV 7 NOLABEL BW	does the same as example 6, but showing the image in black and white.
6)	MN 1 TV 1 CF=WRMB LEFT TV 2 RIGHT OLD	is an example of the normal sequence for loading images 1 and 2 for blink comparison. The first command sets the span. The second loads the left half of the image, while the third loads the right half, using the old span and zero.

ITV INTERACT WITH AN IMAGE DISPLAYED ON THE AED

Form: ITV

This routine enables you to interactively examine an image displayed on the AED video display. Upon calling this routine, the AED cursor is activated, and can be moved around the image with the joystick. Several keys on the AED are activated to perform functions when struck:

B	Blink between two images with the joystick
C	Use the joystick to roll through the color map
D	Type out selected pixel row, column, and value
E	Exit to VISTA
H	Print this list
I	Zoom in to cursor
O	Zoom out of cursor
P	Pan over to cursor
R	Restore image to center

The D key prints pixel values taken from the displayed image's original data buffer; therefore, so be careful to note if the image has been modified after being displayed. The D key also loads the coordinates of the most recently selected pixel into VISTA variables R and C. If the image had to be compressed to fit in the AED, the 'D' key will not show every pixel. If you want to look at individual pixels in a compressed image, use the BOX word in the TV command to display smaller sections of the image.

The B key will cause the AED to rapidly blink between two images which have been displayed using the TV command and the LEFT and RIGHT keywords. Note that the D function does not work with such images. Move the joy stick up and down to control the rate of the blink or left and right to just display the left or right image. Hit any key to stop the blinking.

The C key enables you to roll the color map up or down through the displayed intensity range by moving the joystick up or down. This enables you to highlight subtle features or transitions in the image by quickly positioning sharp changes in color at any desired intensity. Hit any key to stop the color map roll.

COLOR LOAD A COLOR MAP INTO THE AED VIDEO DISPLAY

Form: COLOR [CF=filename] [BW] [INV]

where:

CF= loads an already-defined map into the AED,

BW loads a black and white map, and

INV inverts the ordering of the map.

This routine is used to change or define new color maps for translating displayed pixel intensities into color. The new map is entered without reloading the image.

The color map is a list of length 256, each entry of which holds three numbers. These entries are called 'levels', and code the gun intensity for the red, green, and blue guns in the TV: their values run from 0 to 255. The 256 entries show the proportions of these colors to use to display each of the 256 intervals of intensity between each 'span', or contour interval in the TV picture. COLOR defines these numbers, either by loading them from a pre-determined list, or by creating a new list.

If you wish to create a new list, type the command 'COLOR' with no options. The program will ask you to define the proportions of red, green, and blue intensity to use for each of the entries in the color map. The entries are numbered from 0 to 255. The proportions of each color are linear functions of entry number. You define a starting entry number and value, and an ending entry number and value. You may chain several linear segments together to produce various effects in the color map. You must define all 256 entries in the map.

The last thing the COLOR program will ask you, is to specify the color of the label and tick marks.

As an example of defining a color map, consider the black and white map. For each level of intensity, the proportions of red, green, and blue are equal. We want increasing brightness to correspond to higher intensity, so each color should have zero intensity at the bottom of the map, and full intensity (255) at the top of the map. The sequence of responses for defining this color map would be:

```
COLOR                                                  (begins the sequence)
Red: starting level 0 enter initial value:   0
Go to level                                255
with intensity                             255

Green: starting level 0 enter initial value: 0
Go to level                                255
with intensity                                   255

Blue: starting level 0 enter initial value:  0
Go to level                                255
with intensity                                   255

Define label intensities: Red:                   200
                          Green:           230
```

Blue: 50

Save color file: Y or N

In this example, all three colors have zero intensity at the bottom level of the color file, and full intensity at the top level. The label color is green with slightly less red, with a tinge of blue.

VTEC PRINT AN IMAGE ON THE VERSATEC

Form: VTEC sourc [span] [zero] [L=span] [Z=zero] [BOX=n] [OLD] [INV]

where:

sourc	(integer or $ contruct) is the image to be displayed,
span or L=	set the span level for the color map,
zero or Z=	set the zero level,
BOX=n	displays the part of the image in box 'n',
OLD	displays using previosly defined parameters, and
INV	inverts black and white in the output.

This routine will print the image 'sourc' on the Versatec printer. The pixels are translated for display by first subtracting the 'zero' level and loading the result modulo the intensity 'span' range. By default, 'zero' is '0' and 'span' is four times the image mean. (Important: VTEC does not calculate the mean, so you must use the MN command first if you want to use the default span.) These are default values are good for low-background astronomical images, but you should experiment with the zero and span, since careful choice of these parameters can enhance or suppress various features present in the image. The 'span' and 'zero' values can be specified either as an ordered pair of real numbers (notice that 'span' precedes 'zero') or with the L and Z keywords. Pixels less than the zero level are displayed as zeros.

The keyword 'OLD' tells the routine to re-use the last set of parameters. The 'BOX=n' keyword specifies that only the image subsection in box 'n' (see BOX command) is to be displayed.

VTEC usually prints objects as dark on a light background. To reverse this, use the INV keyword.

Examples:

1) MN 1
VTEC 1 prints image 1, setting the span to be 4 times the image mean.

2) VTEC 3 100. 30.0 prints image 3 into the AED, setting the

span to be 100 and the zero to be 30.

3) VTEC 3 L=100. Z=30. does the same thing as example 2

4) VTEC 3 BOX=4 INV prints the section of image 3 that is in box 4, showing light objects against a dark background.

PIC PRODUCE A GREYSCALE PICTURE ON A VT100

Form: PIC sourc [span] [zero] [L=span] [Z=zero] [BOX=n] [NOERASE] [OLD] [INV] [INT]

where:

sourc	(integer or $ contruct) is the image to be displayed,
span or L=	set the span level for the color map,
zero or Z=	set the zero level,
BOX=n	displays the part of the image in box 'n',
NOERASE	prevents the program from erasing the previous image when displaying a new one,
OLD	displays using previosly defined parameters,
INV	produces light objects against a dark background, and
INT	allows you to examine pixel values with a cursor.

This routine will display the image 'sourc' on a VT100 'Retrographics' terminal. The pixels are translated for display by first subtracting the 'zero' level and loading the result modulo the intensity 'span' range. By default, 'zero' is '0' and 'span' is four times the image mean. (Important: PIC does not calculate the mean, so you must use the MN command first if you want to use the default span.) These are default values good for low background astronomical images, but you should experiment with the zero and span, since careful choice of these parameters can enhance or suppress various features present in the image. The 'span' and 'zero' values can be specified either as an ordered pair of real numbers (notice that 'span' precedes 'zero') or with the L and Z keywords. Pixels less than the zero level are displayed as zeros.

The keyword 'OLD' tells the routine to re-use the last set of parameters. The 'BOX=n' keyword specifies that only the image subsection in box 'n' (see BOX command) is to be displayed. If you type INT, PIC will give you a cursor. You can move this cursor around with the 'arrow' keys on the terminal. If you hit any key (except 'Q'), the row, column, and value of the pixel closest to the cursor is displayed at the bottom of the plot. 'Q' exits from PIC. If you hit the keys 0 to 9, the row and column position at the cursor are loaded into the variables Rn and Cn, where 'n' is the key struck.

Examples:

1)	MN 1 PIC 1	displays image 1, setting the span to be 4 times the image mean.
2)	PIC 3 100. 30.0	displays image 3, setting the span to be 100 and the zero to be 30.
3)	PIC 3 L=100. Z=30.	does the same thing as example 2
4)	PIC 3 BOX=4 INV	displays the section of image 3 that is in box 4, showing light objects against a dark background.

LINE PLOT A SELECTED ROW, COLUMN, OR SPECTRUM (INTERACTIVE)

Form: LINE sourc [BOX=b] [R=n] [C=n] [S=n] [MIN=f] [MAX=f] [XS=f] [XE=f] [OLD]

where:

sourc	(integer or $ contruct) is the image from which rows or columns will be selected,
BOX=b	limits the image 'sourc' to box 'b',
R=n, C=n	plots the selected row or column,
S=n	plots spectrum n,
MIN, MAX	select limits for y-axis of plot,
XS, XE	select limits for x-axis of plot, and
OLD	uses parameters from previous plot.

This command will produce a plot of a selected row, column, or spectrum. It is similar to the PLOT command, but this command will produce a plot which you can interact with. On the other hand, the PLOT command will produce a labeled plot more suitable for hard output. The index 'sourc' specifies which of the image buffers contains the image for plotting a row or column. No 'sourc' is needed to plot a spectrum. The optional index 'box' specifies one of the 10 available boxes (see BOX command) to be used to select a sub-section of an image. Row 'n' is displayed by using the option R=n, column 'n' by using the C=n option, and spectrum buffer 'n' by using the S=n option.

Normally the endpoints of the x- and y-axes are selected in this way: The x-axis runs over the entire row, column, or spectrum selected. The y-axis runs from the minimum to the maximum value of the row, column, or spectrum being plotted. This scaling can be overridden by the MAX and MIN keywords for the vertical axis and XS and XE for the horizontal. XS and XE refer to the starting and ending X values in pixels or angstroms. The OLD keyword will produce a plot using the last used set of parameters.

This command continues in interactive mode until the RETURN key is hit. While the command is still active the cursor can be moved about using the left- and right-arrow keys. The up- and down-arrow keys cause the plot to be redrawn with the cursor centered on the screen. The plot can also be magnified with the following keys: PF1= increase Y mag, PF2=decrease Y mag, PF3=increase X mag, PF4=decrease X mag. (Note: Increasing Y mag too much produces bad results.)

Examples:

1) LINE 1 R=100 — Plots row 100 of image 1.

2) LINE 1 R=100 MIN=0. MAX=100. — Plots row 100 of image 1, with y-axis running from 0. to 100.

3) LINE 4 XS=20 XE=45 R=4 — Plots row 4 of image 4, where the x- axis runs over columns 20 through 45.

4) LINE S=4 — Plot spectrum 4.

PLOT — PLOT A SECIFIED ROW, COLUMN OR SPECTRUM (NON-INTERACTIVE)

Form: PLOT sourc [S=n] [BOX=b] [R=n] [C=n] [RS=(n1,n2)] [CS=(n1,n2)] [MIN=f] [MAX=f] [XS=f] [XE=f] [GRID] [INFO] [SEMILOG] [LOG] [OLD] [HARD]

where:

sourc	(integer or $ contruct) is the image from which rows or columns will be selected,
BOX=b	limits the image 'sourc' to box 'b',
R=n, C=n	plots the selected row or column,
RS=(n1,n2)	plots sum of selected rows,
CS=(n1,n2)	plots sum of selected columns
S=n	plots spectrum n instead of an image,
MIN, MAX	select limits for y-axis of plot,
XS, XE	select limits for x-axis of plot,
GRID	produces a full plot grid intead of just tick marks around the edges of the plot,
INFO	puts additional about the data on the plot
SEMILOG	plots log(y) against x,
LOG	plots log(y) against log(x),
OLD	uses parameters from previous plot, and
HARD	sends the output to the VERSATEC.

This command will produce a plot of a selected row, column, or spectrum. It is similar to the LINE command, but this command will produce a labeled plot more suitable for hard output, and cannot be used to interactively examine the plot. On the other hand,

the LINE command is more suited to interactive line examination, since PLOT has no interactive cursor or scale control.

The index 'sourc' specifies which of the image buffers contains the image for plotting a row or column. No 'sourc' is needed to plot a spectrum. The optional index 'box' specifies one of the 10 available boxes (see BOX command) to be used to select a sub- section of an image. Row 'n' is displayed by using the option R=n, column 'n' by using the C=n option, and spectrum buffer 'n' by using the S=n option. The RS and CS keywords are used to display the sum of the selected rows or columns in an image.

Normally the endpoints of the x- and y-axes are selected in this way: The x-axis runs over the entire row, column, or spectrum selected. The minimum and maximum y-values define the corresponding limtis for the y-axis. This scaling can be overidden by the MAX and MIN keywords for the vertical axis and XS and XE for the horizontal. XS and XE refer to the starting and ending X values in pixels (or Angstroms for a spectrum that has had its wavelength scale determined.)

The keyword GRID will produce a coordinate grid on the plot. This is very useful if you are trying to make measurements from a hard copy plot. The keyword INFO will put some additional information onto the plot. The keywords LOG and SEMILOG will produce log vs. log or log vs. linear plots as desired. The OLD keyword will produce a plot using the last used set of parameters. HARD will send the output to the VERSATEC.

Examples:

1) PLOT 1 R=100 Plots row 100 of image 1.

2) PLOT 1 RS=(100,120) MIN=0. MAX=100. Plots sum of rows 100 to 120 of image 1, with y-axis running from 0. to 100.

3) PLOT 4 XS=20 XE=45 R=4 Plots row 4 of image 4, where the x- axis runs over columns 20 through 45.

4) PLOT 4 OLD SEMILOG Plot the old graph (whatever it was) in semi-log format.

5) PLOT S=4 Plot spectrum 4.

CONTOUR MAKE CONTOUR PLOT OF IMAGE OR IMAGE SUB-SECTION

Form: CONTOUR source [BOX=b] [SCALE=s]
[LEVELS=(L1,L2,L3,...)] [HARD]

where:

source	(integer or $ contruct) is the image to be plotted.
BOX=b	tells the program to plot only the part of the image in box 'b'.
SCALE=s	sets the plot scale to 's' units/pixel (typically arcsec/pixel).
LEVELS=	sets the levels of the plot.
HARD	sends the output to the Versatec printer.

This program produces a contour plot of an image or an image sub-section. The program is still under development: more options will be added later.

If the levels are not specified, the program computes its own. The lowest contour is the mean of the image in the area being plotted; higher contours are 0.5 magnitude (a factor of 1.583) apart. This turns our to be useful for a wide variety of astronomical applications. Use the LEVELS keyword to define contour levels, if you wish. You can have up to 20 levels.

The SCALE keyword sets the scale of the axes in units/pixel. If you leave this out, the scale is in row and column numbers.

The contours produced by MONGO are unlabeled, and there is no way to differentiate peaks from valleys. The plot is made as large as will fit on a VT100 screen if sent to a terminal; if sent to the printer, the plot is made as large as will conveniently fit on an 8.5 by 11 inch sheet.

Examples:

1)	CONTOUR 1	Plots all of image 1. The contour levels are set in the program.
2)	CONTOUR 2 HARD	Does the same as example 1, but sends the output to the Versatec.
3)	CONTOUR 1 BOX=3	Does the same as example 1, but plots only the part of

the image in box 3.

4) CONTOUR 1 LEVELS=(10,20,30,40,50) An example of setting the levels.

CL CLEAN VT100 SCREEN OF TEXT AND GRAPHICS

Form: CL

The VT100 screen is cleared of all text and graphics.

```
AI              ADD TWO IMAGES
SI              SUBTRACT TWO IMAGES
DI              DIVIDE TWO IMAGES
MI              MULTIPLY TWO IMAGES
```

The commands that perform arithmetic between two images are:

```
AI dest source [BOX=n] [DR=n] [DC=n]          (dest=dest+source)
SI dest source [BOX=n] [DR=n] [DC=n] [DARK]   (dest=dest-[Ts/Td]*source)
DI dest source [BOX=n] [DR=n] [DC=n] [FLAT]   (dest=dest/source[*source mean])
MI dest source [BOX=n] [DR=n] [DC=n]          (dest=dest*source)
```

where:

dest	(integer or $ contruct) is the buffer holding one image and also specifying the buffer where the result will be stored.
source	(integer or $ contruct) is the other image used in the arithmetic.
BOX=n	uses only that portion of the source image that is in box 'n'.
DR, DC	shifts the 'source' image before doing the arithmetic.

Note that the result of the arithmetic operation is stored in the first location mentioned on the command line! The syntax of these commands is

OPERATE ON (image 1) WITH (image 2)

The operations are done on a pixel-by-pixel basis so that pixel (I,J) of the 'dest' image is combined with pixel (I,J) of the 'source' image. 'DR' and 'DC' can be used to specify an optional offset in number of rows or columns of the source image when it operates on the destination. 'DR' and 'DC' can be negative as well as positive, but will be rounded to the nearest integer. The result is that row 'I' and column 'J' of the source operates on row 'I+DR' and column 'J+DC' of the destination.

IMPORTANT: If the images do not overlap exactly, only those pixels in the destination image that are also in the source image are effected by the operation. The other pixels are not changed!

If the optional keyword FLAT is included in the DI command, the resulting image is multiplied by the mean of the image in the 'source' buffer. The mean is NOT computed by the DI command, but must be computed with the MN command ahead of time. If the optional keyword DARK is included on SI command line, then the source image is scaled by its exposure time relative to the destination image before the subtraction. These two operations preserve the mean of the destination image.

You can make an image which contains nothing but zeroes by subtracting it from itself.

Examples:

AI 2 1	adds image 1 to image 2 and stores the result in image 2.
DI 2 3 FLAT	divides image 2 by image 3, then scales the result by the mean of image 3. This preserves the mean of image 2.
AI 4 7 DR=4 DC=-10	add image 7 to image 4, but first shift image 7 by 4 rows and by -10 columns. The pixels in image 4 that are also in image 7 will contain the sum. The pixels in image 4 that are NOT in image 7 will not be changed.
AI $IM1 $IM2	add image IM2 to IM1, where IM1 and IM2 are variables. (This is helpful in procedures.)

AC	ADD A CONSTANT TO AN IMAGE
SC	SUBTRACT A CONSTANT FROM AN IMAGE
MC	MULTIPLY AN IMAGE BY A CONSTANT
DC	DIVIDE AN IMAGE BY A CONSTANT

The commands that perform arithmetic between an image and a constant are:

AC dest number	(dest=dest+number)
SC dest number	(dest=dest-number)
DC dest number	(dest=dest/number)
MC dest number	(dest=dest*number)

'dest' specifies the buffer that contains the image to be processed and 'number' is the value to be used. The operation is applied to

each pixel in the image.

'number' can either be an explicit numeric value or a defined symbol (see SET command). If 'number' is not included on the command line then VISTA will request a numeric value (not a symbol) when the command is executed. The program will also ask you for a number if the value of the variable constant (if you use one) is zero. If you want to make an image zero, you cannot multiply it by the constant 0.0: you have to subtract the image from itself.

Examples:

1)	AC 7 4.03	adds 4.03 to image 7
2)	AC 7 BIAS	adds the value of the variable BIAS to image 7. This does the same thing as example 1 if BIAS equals 4.03.
3)	DC $J 2.0	divides image J (J a variable) by 2.0

AS	ADD TWO SPECTRA
SS	SUBTRACT TWO SPECTRA
MS	MULTIPLY TWO SPECTRA
DS	DIVIDE TWO SPECTRA

The commands that perform arithmetic between two spectra are:

```
AS dest source [DC=n] [FLAT]    (dest=dest+source -[source mean])
SS dest source [DC=n] [FLAT]    (dest=dest-source +[source mean])
MS dest source [DC=n] [FLAT]    (dest=dest*source /[source mean])
DS dest source [DC=n] [FLAT]    (dest=dest/source *[source mean])
```

'dest' and 'source' specify the spectra (1 to 20) to be used in the operation. 'DC' can be used to specify an optional positive or negative offset in number of pixels in the source spectrum when it operates on the destination spectrum. The result is that pixel 'J' of the source operates on pixel 'J+DC' of the destination. If the two spectra do not overlap exactly, only those pixels common to both spectra are added: the rest are not changed!

If the optional keyword [FLAT] is specified, the arithmetic operation will preserve the destination spectrum's mean.

Examples:

1)	AS 3 6	adds spectrum 6 to spectrum 3, storing the result in spectrum 3.
2)	AS 3 6 DR=-12	does the same as above, but first shifting spectrum 6 left by 12 pixels. Only the pixels common to both spectra are added.
3)	MS $S1 $S2	multiplies spectrum S1 by S2, where S1 and S2 are variables. (This is helpful in procedures).
4)	SS 1 2 FLAT	subtracts spectrum 2 from spectrum 1, adding the mean of spectrum 2 to the result, thereby preserving the mean of spectrum 1.

ACS	ADD CONSTANT TO SPECTRUM
SCS	SUBTRACT CONSTANT FROM SPECTRUM
MCS	MULTIPLY SPECTRUM BY CONSTANT
DCS	DIVIDE SPECTRUM BY CONSTANT

The commands that perform arithmetic between a spectrum and a number are:

ACS dest number	(dest=dest+number)
SCS dest number	(dest=dest-number)
MCS dest number	(dest=dest*number)
DCS dest number	(dest=dest/number)

'dest' specifies the spectrum to be processed and 'number' is the value to be used in the operation.

'number' can either be an explicit numeric value or a defined symbol (see SET command). If 'number' is not included on the command line then VISTA will request a numeric value (not a symbol) when the command is executed.

Examples:

1)	ACS 7 4.03	adds 4.03 to spectrum 7

2)	ACS 7 BIAS	adds the value of the variable BIAS to spectrum 7. This does the same thing as example 1 if BIAS equals 4.0
3)	DCS $J 2.0	divides spectrum J (J a variable) by 2.0

BOX DEFINE A BOX OR IMAGE SUBSECTION

Form: BOX box_num [NC=n] [NR=n] [CR=n] [CC=n] [SR=n] [SC=n]

where:

box_num	(integer) is the number of the box being defined,
NC	defines the number of columns in the box,
NR	defines the number of rows in the box,
CR	defines the center row,
CC	defines the center column,
SR	defines the starting row, and
SC	defines the starting column.

To permit analysis on sub-sections of an image VISTA can store up to 10 sets of specifications for image sub-sections (or boxes). These stored parameters can be used by other commands (such as TV, WIND and PRINT) by including 'BOX=' in the command line, modifying these commands so they operate only on the designated subsection.

When a box is initially defined, the size of the box in both dimensions and its origin or center must be specified. Subsequently, however, if any of the box parameters are to be changed, only the modifications need to be entered. The locations of the defined boxes can be found with the PRINT BOX command.

Examples:

1)	BOX 1 SC=100 NC=100 SR=0 NR=100	defines box 1 as columns 100 to 199 and rows 0 to 99
2	BOX 2 CC=100 CR=100 NR=13 NC=13	defines a box having 13 rows and columns, centered on row=100 and column=100.

MN COMPUTE MEAN OF THE PIXEL VALUES IN AN IMAGE

Form: MN source [NOBL]

The mean of all the pixels in the image contained in buffer 'source' is computed. If the keyword NOBL is included then the baseline column (the last column of the image) is not included in the computation. The computed mean value is printed at the terminal and is also loaded into the VISTA variable Mbn, where 'bn' is the image buffer number. (ex: the command MN 1 loads the variable M1).

The mean is used by other commands such as the TV command for

a default display range, or the DI command for rescaling images after flat-field divisions. The VISTA mean variables are always defined, but are set to zero if the image mean has not been calculated.

FLIP CHANGE THE ORINETATION OF AN IMAGE

Form: FLIP source [ROWS] [COLS]

This command 'flips' an image in either rows or columns. Use this command to change the orientation of an image so that it matches the way it is viewed on the sky or on a finding chart, or to get the wavelength dispersion running the way you want before using MASH.

A flip in ROWS reverses the image top to bottom as seen on the television. A flip in columns reversed the image left to right.

Examples:

FLIP 3 ROWS	Inverts image 3 top to bottom.
FLIP 3 COLS	Inverts image 3 left to right.

SKY MEASURE THE 'SKY' OR BACKGROUND LEVEL OF AN IMAGE

Form: SKY source [BOX=n]

where:

source	(integer or $ contruct) is the number of the image that SKY works on.
BOX=n	tells SKY to work only box=n.

This routine finds the sky background level of an image under the assumption that the most common pixel intensity in the image is the level of the 'sky' or background. This is a nonlinear algorithm which is largely insensitive to bright objects in the image. The routine calculates the mean of the image 'source', and builds a histogram of pixel intensities about the mean. The region of the peak value is located in the histogram, and is fit with a parabola to find its precise location. This intensity value is defined to be the sky value, and is loaded into the VISTA variable 'SKY' for access by other commands.

The use of a box may be helpful if a large fraction of the image is occupied by stars or an extended object, causing the routine to measure a sky level systematically higher than the true level. The box can in this case be used to select a region of the image that does not have bright objects in it.

Examples:

1) SKY 5 finds the background in image 5, loading

its value into the variable SKY.

2) SKY $W BOX=5 does the same as example 1, but finds the background only within box number W.

ABX ANALYZE THE IMAGE WITHIN A BOX

Form: ABX source boxes [TOTAL=var] [MEAN=var] [HIGH=var] [LOW=var] [HIGH_ROW=var] [HIGH_COL=var] [LOW_ROW=var] [LOW_COL=var] [SIGMA=var] (redirection)

Where:

source	(integer of $ contruct) specifies the image.
boxes	list boxes to be used in the analysis
var	the name of a variable.

ABX finds the properties of an image in a given box or a set of boxes. If you do not specify a box, all defined boxes are used. The boxes are specified by integers, and NOT with the BOX= word as in other commands. Thus, for example

ABX 3	Finds the properties of image 3 in all boxes.
ABX 3 1	Finds the properties of image 3 in box 1
ABX 2 3 4 5 6	Ananyzes image 2 in boxes 3, 4, 5, and 6

ABX finds the image mean, total count over all pixels, values of the highest and lowest pixels, location of the highest and lowest pixels, and the standard deviation of the counts in the pixels about the mean. A table of the results is printed on the output device.

ABX will store the values for the various properties of the image in variables if certain keywords are included on the command line.

TOTAL=var	Stores the total count of all the pixels in 'var'
MEAN=var	Stores the average of the image
HIGH=var	Stores the VALUE of the pixel with the highest count
LOW=var	Stores the VALUE of the pixel with the lowest count
HIGH_ROW=var	Stores the row number in which the highest-valued pixel is located
HIGH_COL=var	Stores the column number in which the highest-valued pixel is located
LOW_ROW=var	Stores the row number in which the lowest-valued pixel is located.
LOW_COL=var	Stores the column number in which the lowest-valued pixel is located.
SIGMA=var	Stores the standard deviation of the pixel values

about the mean.

Examples of storing information in variables are:

ABX 1 3 MEAN=M3 SIGMA=SIG3	Analyzes image 1 in box 3, storing the mean in variable M3 and the standard deviation in SIG3.
ABX 2 7 HIGH_ROW=HR HIGH_COL=HC	Analyzes image 2 in box 7, storing the location of the highest-valued pixel in HR and HC

The use of the variable-setting keywords should be used only when you analyze an image one box at a time, as the values will be loaded into the variables only for the last box analyzed.

HIST DISPLAY HISTOGRAM OF IMAGE VALUES

Form: HIST source [BOX=b] [BIN=n] [XMIN=x1] [XMAX=x2] [YMIN=y1] [YMAX=y2] [HARD] [NOLOG]

where:

source	(integer of $ construct) is the image
BOX=b	limits the computation to those pixels in box 'b'.
BIN=n	bins the image values by the specified factor
XMIN, XMAX	limits the computation to those pixels with values between x1 and x2, inclusive.
YMIN, YMAX	limits the display of the histogram on the Y-axis to be from y1 to y2.
HARD	sends the plot to the hardcopy device.
NOLOG	displays the number of pixels at each intensity, rather than the logarithm.

This program displays a histogram of an image, plotting the logarithm of the number of pixels at each value for the image 'source'.

Use the BIN word to specify how wide the intensity intervals show in the plot is to be. Normally the binning factor is 1, meaning that the plot displayed is the logarithm of the number of pixel values at each intensity (the image values are converted to integers). If the bin factor is non-zero, the display is the log of the number in larger bins. For example, if the bin was 5, then the plot shows the number of pixels with intensity 0 - 4, 5 - 9, 10 - 14, 15 - 19, etc. If there is a large range in intensities, the BIN word should be used to keep the plot from having so many points that it looks like hash.

The BOX word limits the calcuation to those pixels in the specified box. The XMIN and XMAX words limit the calculation to those

pixels in the specified intensity range. If XMIN is not given, the lower limit will be the minimum pixel value in the image. If XMAX is not given, the upper limit will be the maximum pixel value in the image.

The YMIN and YMAX words, by contrast, limit the DISPLAY of the histogram so that the Y-axis runs over the given range. These words do not effect the calculation in any way. If YMIN is not used, the lower limit of the display will be the smallest number of pixels in the image that have a given value (often this is zero pixels at many intensities). If YMAX is not used, the upper limit of the display will be the largest number of pixels which have a certain intensity.

NOLOG makes the plot show the actual number of pixels at each intensity, rather than the logarithm. When the logarithm is computed in the default option, intensities with no pixels are given the value 0, so you cannot distinguish an intensity with 1 pixel and an intensity with 0 pixels unless you use the LOLOG word.

Examples:

HIST 4	shows the histogram for image 4
HIST $Q BOX=3	shows the histogram for image Q (where Q is a variable) using only the pixels in box 3
HIST 2 XMIN=1000 XMAX=1999	shows the log of number of pixels with values between 1000 and 1999.
HIST 4 NOLOG	shows the number of pixels (not the logarithm) at each value in image 4.

AXES FIND THE CENTROID OF AN OBJECT IN AN IMAGE

Form: AXES source BOX=n

where:

source	(integer or $ contruct) is the image that AXES uses, and
BOX	specifies the section of the image used.

This command will find the centroid of an object within the

specified box in the source image. The centroid coordinates are loaded into VISTA variables AXR and AXC, and are also held in a common block for use by other routines.

The routine uses the highest pixel value on the box perimeter as a threshold value for the centroid. The threshold is subtracted from each pixel in the box during the calculation.

PROFILE FIND THE SURFACE BRIGHTNESS PROFILE OF AN EXTENDED OBJECT

Form: PROFILE dest source [N=n] [ITER=(n1,n2)] [SCALE=f] [CENTER] [PA=f] [INT]

where:

dest	(integer or $ construct) is a spectrum which will hold the calculated profile,
source	(integer or $ contruct) is the image that contains the object whose profile is being measured,
N=n	sets the number of steps in the iteration,
ITER=(n1,n2)	sets number of iterations for (n1) fast bilinear interpolation and (n2), slower sinc interpolation,
CENTER	solves for the contour centers,
PA=f	position angle for the top of the image, and
INT	interactively iterate contour solution.

This command is used to find the surface-brightness profile of an object by describing it as a set of elliptical contours. The center of the object must first be calculated with the AXES command. PROFILE uses this center as the starting point for its calculations. The profile is found by sampling the image with a set of circles with radius 1, 2, 3, ... pixels. The average value of the pixels along a circle is the mean surface brightness of that contour. Low order sine and cosine transforms are taken along the contour to derive its center, position angle, and ellipticity. After these are found for the entire image in the radius specified, the countours are adjusted to more exactly fit the isophotes. The first iteration usually turns the original circles into ellipses with varying positoin angles and eccentricities as a function of major-axis length.

High accuracy sinc interpolation is used to find the values of the pixels along the inner 15 contours. Outside of this, either a lower accuracy (but faster) interpolation can be used, or an even faster bilinear interpolation scheme. The kind of iteration used is set by the ITER keyword.

The result of the profile calculation is stored in a common block for later use. Print the contents of this block with PRINT, or save it in a diskfile with SAVE. The results are also written into the specified spectrum. Use PLOT or LINE to show that spectrum.

Position angles are calculated assuming that the position

angle of the 'top' of the image (as seen on the TV) is 0. This can be changed with the PA keyword. Use PA < 0 to indicate that the image has been reversed right-left (the 'normal' arrangement for an image is north at top, east at left).

APER PERFORM APERTURE PHOTOMETRY ON AN EXTENDED OBJECT

Form: APER source [RAD=(r1,r2,...,r10)] [MAG=(M1,M2,...MN)] [STEP=(size,n)] [SCALE=f] [C=(r,c)] [SCALE=f] [OLD] [INT] [REFF]

This command will sum up the intensities of pixels falling within circular apertures centered on an object in the 'source' image. Using apertures of differing radii will enable you to characterize the radial intensity distribution of an object. From 1 to 10 aperture radii in arc seconds can be specified with the RAD keyword. If more than one aperture is specified, the list must be enclosed within parentheses. It does not have to be in order, however. Alternatively, a linearly increasing squence of apertures can be specified with the STEP keyword, where 'incr' is the radius increment in arc seconds, and 'n' is the number of apertures. The pixel scale in arcsec/pixel is specified with the SCALE keyword. Specifying a list with the MAG word will give observed magnitudes in the listed apertures. The centers of the apertures must be calculated ahead of time with the AXES centroiding command, or loaded with the C keyword. Lastly, the OLD keyword tells the routine to use any unspecified parameters from the last time.

The routine will produce a list of the apertures, the sum of pixels interior to them, the average interior surface brightness, and the same quantities for the rings defined by sequential apertures. The summations are done to the nearest pixel at the edges of the aperture, that is, no fractional pixel interpolation is done at the edges. For this reason, results obtained with very small apertures may be inaccurate. The results are stored in a common block, and can be examined with the PRINT command, and stored or retrieved with the GET and SAVE commands.

WIND WINDOW AN IMAGE TO A SMALLER SIZE

Form: WIND source BOX=n

where:

source (integer or $ construct) is the image being made smaller, and

BOX tells VISTA what part of the old image to save.

This command can be used to select part of an image for future reduction or analysis. The image specified by 'source' is redefined to be the part of it enclosed by BOX 'n'. If the specified box extends over the edges of the image, an error results. Set the size and position of the box with BOX. Note: This command cannot be used to expand the size of an image.

Example: Let image 7 have rows numbered 0 - 500 and columns numbered 0 - 500. We want to chop out a 100 by 100 image, with the first row of the new image at 100, and the first column at 200.

```
BOX 1 SR=100 SC=200 NR=100 NC=100
WIND 1 BOX=1
```

The WIND command is similar to COPY. To copy and window an image at the same time, use COPY im2 im1 BOX=n.

SHIFT SHIFT AN IMAGE IN ROWS OR COLUMNS

Form: SHIFT source [DC=f] [DR=f]

where:

source (integer or $ contruct) tells SHIFT what image to work on,

DC=f shifts the image by f columns, and

DR=f shifts the image by f rows.

SHIFT will move the 'source' image by any desired amount. Specify the desired shift in pixels in the row or column directions with the DR or DC keywords. The routine first rounds the desired shift to the nearest pixel, and adjusts the variables specifying the image's origin. The routine then uses sinc-interpolation to move the image the remaining fraction of a pixel in the row and column directions.

Examples:

1) SHIFT 1 DR=0.5 shifts image 1 by 0.5 rows.

2) SHIFT $IM DR=-0.2 shifts image IM (IM a variable) by -0.2 rows.

Warning: The program can be used to shift an image so that its starting row or column is negative. This can lead to errors or FORTRAN crashes in other routines in a way that is not easily fixable. After shifting an image, IMMEDIATELY window the image (see WIND) so that the starting row and column are greater than or equal to zero!

CLIP REPLACE PIXELS OUTSIDE AN INTENSITY RANGE

Form: CLIP source [MAX=f] [MIN=f] [VMAX=f] [VMIN=f] [BOX=n]

where:

source (integer or $ contruct) is the image that CLIP works on.

MAX=n sets the level above which pixels are adjusted.

MIN=n sets the level below which pixels are adjusted.

VMAX=n replaces all pixels above MAX by 'n', and

VMIN=n replaces all pixels below MIN by 'n'.

This routine searches through the image 'source', and replaces pixels with intensities above MAX with VMAX and pixels below MIN with VMIN. If VMIN and VMAX are not specified, they default to zero.

If VMIN or VMAX are specified, but no thresholds are entered with MAX or MIN, the thresholds are taken to be the replacement values. If neither MAX nor VMAX are specified, no upper level clipping will be performed. If neither MIN nor VMIN is specified, but MAX or VMAX is, no lower level clipping will be performed. If no keywords are given, the routine by default will set all negative pixels to zero. Specify 'BOX=n' only to clip the portion of the image within box 'n'.

Examples:

1) CLIP 1 MAX=110. VMAX=100. replace all pixels in image 1 that are above 110 by 100.

2) CLIP 1 MIN=SKY VMIN=0.0 — replace all pixels below SKY by 0.0. SKY is a previously-defined variable.

3) CLIP 1 MAX=110. VMAX=100. MIN=SKY VMIN=0.0 — does the same as examples 1 and 2 simultaneiosly.

4) CLIP 1 VMAX=100. — replaces all pixels above 100 by 100.

5) CLIP 1 VMAX=100. BOX=6 — does the same as example 4, but only in box 6.

6) CLIP 1 — sets all negative pixels in image 1 to zero.

MASK TELL PROGRAMS TO IGNORE SPECIFIED PIXELS
UNMASK TELL PROGRAMS TO STOP IGNORING SPECIFIED PIXELS

Form: MASK [ROW=n] [COL=N] [BOX=N] [PIX=(ROW,COL)]
UNMASK [ROW=n] [COL=N] [BOX=N] [PIX=(ROW,COL)]

MASK tells VISTA programs to ignore the specified pixels when doing computations.

Options:

ROW=r	Ignore all pixels in row r.
COL=c	Ignore all pixels in column c.
BOX=b	Ignore all pixels in box b.
PIX=(r,c)	Ignore the pixel at row 'r' and col 'c'.

The command UNMASK is the opposite of MASK; it tells the programs to stop ignoring the specified pixels. UNMASK can be used without options to unmask all pixels.

You can type UNMASK without any options to tell the routines to use all the pixels. Remember that rows and columns are often numbered from ZERO. The row or column numbers specified in MASK are the same as those given by the 'D' option in the ITV command, but remember that if the TV image is compressed, the 'D' option will not give the exact location of certain features in the image. Use the BOX option in the TV command to show smaller sections of an image when finding features you want to mask.

The only routines that test for bad columns now are MARKSTAR, FITSTAR, and SURFACE. The mask may be saved with 'SAVE MASK' and recalled with 'GET MASK'. The mask is stored in [CCD.DATA] with the extension .MSK.

Examples:

1)	MASK COL=234	Masks column 234.
2)	UNMASK COL=234	Removes the mask from column 234.
3)	MASK PIX=(120,100)	Masks the single pixel at column 100 and row 125.
4)	MASK COL=0	Masks column zero.
5)	UNMASK BOX=5	Unmasks the pixels in box 5.

LOG COMPUTE LOGARITHM OF AN IMAGE

Form: LOG source

This command replaces the image pixel values in the image 'source' (integer or $ contruct) by their base-10 logarithm values. Any pixels whose original values were less than or equal to 0 are replace by 0 in the resulting image.

BL CORRECT AN IMAGE FOR BASELINE SUBTRACTION NOISE

Form: BL source [JUMP]

where:

source	(integer or $ construct) tells VISTA what image to work on, and
JUMP	implements detection of jumps in the baseline column.

This command removes the noise introduced through the digital baseline-restoration procedure used by the data collection program. The last column of the CCD, called the 'baseline column' is not illuminated during an exposure. Each pixel in this column determines the zero-level for the corresponding row in the rest of the chip, which is subtracted from each pixel in that row as the image is read off the chip into the data-taking system.

The baseline measurement is rather noisy, so when the zero-levels are subtracted, significant variations in level from one row to another are introduced. The BL program corrects for this by fitting a straight line by least-squares to the values in the baseline column. The original baseline measurements are then added back into the image data and the mean baseline values evaluated from the linear least-squares fit are subtracted.

The baseline column can sometimes exhibit several jumps in level along its length. BL detects these jumps, fitting a series of linear functions to the baseline column between these jumps. To detect the jumps, thus fitting the entire column with more than one linear function, use the JUMP keyword.

The BL procedure should be applied to every raw image before analysis or processing.

SMOOTH GAUSSIAN SMOOTHING OF AN IMAGE

Form: SMOOTH source [FW=f] [FWC=f] [FWR=f]

where:

source — (integer or $ contruct) is the image being smoothed,

FW=f — sets the full width of the gaussian smoothing function to be f pixels, and

FWC and FWR — set the full width for the gaussian smoothing function to have a different width in rows or columns.

This routine will smooth or convolve the 'source' image with a 2D or 1D gaussian. This is useful for reducing noise, enhancing low surface brightness features, or as the first step for looking for sharp features in an image (by subtracing the smoothed image from the original). Specify the full-width-half-maximum (FWHM) of the gaussian in pixels with the FW keyword. If desired, you can specify differing widths in the column or row directions with the FWC or FWR keywords. To smooth the image in the column direction only, (each image row is convolved along its extent seperately) just specify the width with the FWC keyword. To smooth in the row direction only, (down the columns) just specify the FWR keyword.

The convolutions are done in the image domain and may be slow for large widths of the gaussian. Edges are handled properly. At this time, the FWHM's of the gaussians are limited to 21 pixels.

Examples:

1) SMOOTH 2 FW=8.5 — Smooths image 2 with a gaussian having full width 8.5 pixels. The width is the same in the row and column directions.

2) SMOOTH 2 FWR=8.5 FWC=8.0 — Smooths image 2 with a gaussian having full width = 8.5 rows and 8.0 columns.

3) SMOOTH 5 FWR=6.2 — Smooths each row individually with a gaussian having full width 6.2 rows.

ZAP IMAGE MEDIAN FILTER AND PIXEL ZAPPER

Form: ZAP source [SIG=f] [RAD=f] [BOX=n]

where:

source	(integer or $ contruct) specifies the image being worked on,
RAD=f	sets the size of the region around each pixel used in the median filter,
SIG=f	specifies the rejection level in terms of the standard deviation in each circle, and
BOX	tells ZAP to work only in the specified box.

This command will remove high or low pixels or median-filter an image. If called without any keywords, the routine moves a 2 pixel radius circle through the image, and finds the median of the pixels within the circle. If the central pixel exceeds the local pixels' deviation about the median by 5 sigma, it is replaced by the median.

To change the radius of the circle, use the RAD=f keyword. 'f' must be >= 1 and <= 10 pixels. To change the zap detection level, use the SIG=f keyword, where 'f' is the detection level in sigmas. If 'f' is set to zero, this routine will median-filter the image: all pixels will be replaced by the local median within the circle. Specify 'BOX=n' only to process the portion of the image within box 'n'.

Examples:

1)	ZAP 1	Do the filtering with a circle of radius 2, adjusting pixels that are 5 sigma away from the median, where sigma is the standard deviation in each circle.
2)	ZAP 1 BOX=4	does the same as example 1, but only in the region defined by box 4.
3)	ZAP 1 RAD=7	considers a circle of radius 7 at each pixel.

SURFACE FIT A PLANE OR SECOND-ORDER SURFACE TO AN IMAGE

Form: SURFACE source [BOX=n] [PLANE] [SUB]
[DIV] [MASK] [NOZERO] [PIX=N] (redirection)

where:

source	(integer or $ contruct) is the image to which the surface is being fit,
BOX=n	tells VISTA to do the fit only in box 'n',
PLANE	fits a plane, rather than a second order surface,
SUB	has the program subtract the best-fit surface from the original image,
DIV	replaces the original image with itself divided by the best-fit surface,
MASK	tells VISTA to ignore masked pixels, and
NOZERO	suppresses rejection of pixels with zero value.
PIX=n	uses every n'th pixel for speed

This routine fits a second-order polynomial surface to the specified image, or to the subset of that image designated by the BOX keyword. It fits to all pixels in the image or box, except those that have the value zero, or those masked with the MASK command. These two features allow you to mark out sections of an image that you do not want included in the fit. Use the command CLIP to set pixels to zero.

For speed, you can have the program find the best-fitting surface using every n-th pixel. Use the PIX keyword for this. For example, if you say PIX=3, the surface is fit to the pixels in columns 0, 3, 6, 9, ... in rows 0, 3, 6, 9...

A polynomial expression for the fit is printed on the output device.

To make the program ignore masked pixels, you must use the MASK keyword in the command. If you do not, it will use all pixels except those that have value zero. To fit ALL pixels, including those that have value zero, use the NOZERO word. (This word is short for NO ZERO CHECKING). To fit a planar (instead of a second-order) surface, include the word PLANE.

In its normal operation, the program replaces the image with the best-fitting surface. To subtract the surface from the image, use the word SUB. To divide the image by the surface, use the word DIV. The best-fit surface is applied in the manner you specify to EVERY

pixel, regardless what the PIX word says.

Examples:

1)	SURFACE 1	replaces image 1 by the best-fitting polynomial. Pixels with value zero are not included in the fit.
2)	SURFACE 1 NOZERO	does the same as example 1, but this time ALL pixels are included in the fit.
3)	SURFACE 1 BOX=2	does the fit only in BOX 2.
4)	SURFACE 1 MASK	fits the best second-order surface, ignoring masked pixels.
5)	SURFACE 1 SUB	subtracts the best fitting surface from image 1.
6)	SURFACE 1 PIX=5	does the fit using every 25th pixel.

MASH MASH AN IMAGE INTO A SPECTRUM

Form: MASH dest source SP=(i1,i2) [BK=(b1,b2)] [NORM] [COL=(c1,c2)] [SKY=s] [REFLAT] [SUB]

where:

dest	(integer or $ construct) is the buffer which will hold the resulting spectrum,
source	(integer or $ construct) is the image from which the spectrum is being made,
SP=	delimits the rows in the image 'source' which are used to make the spectrum,
BK=	delimits rows used to determine the background,
NORM	averages the added rows,
COL=	takes the spectrum from the specified columns,
SKY=	Saves the sky rows (from BK= keyword) in spectrum buffer s.
REFLAT	fits each column with a parabola, and uses this as the background, and
SUB	subtracts the average background spectrum from the original image.

This command will collapse selected portions of an image into a single row, thus producing a spectrum.

The spectrum is produced under the assumption that the dispersion runs parallel to the image rows: i.e. the pixels in any given column were struck with light from only one wavelength.

The rows indicated by the SP, COL, and BK keywords are PAIRS of numbers enclosed in parentheses. Rows 'i1' to 'i2' are co-added into the final 1-row spectrum. The COL word selects the columns used to construct the spectrum. Background rows, specified with the BK keyword, are co-added to produce another spectrum: this is subtracted from the final 1-row spectrum and saved into another spectrum buffer if the SKY keyword was supplied. Up to six SP or BK keywords can be specified. Also, if an image or background section consists of only one row (i1=i2 or b1=b2) then the second number and the parentheses can be omitted. Rows lying in both spectrum and backgound rows are taken to be spectum rows: they are not used in the determination of the background.

If the keyword 'NORM' is specified, then the sum of the image rows is divided by the number of image rows. If the 'REFLAT' word is used, the program fits a polynomial to each column of the background

rows, and uses the polynomial as the background. In this case, the BK rows select the rows used in the fit. 'SUB' will subtract the background spectrum from the original IMAGE.

Examples:

1) MASH 3 1 SP=(100,150) BK=(90,99) BK=(151,170)

This takes rows 100 through 150, inclusive, in image 1, and produces a spectrum out of them. Rows 90 through 99 AND 151 through 170 are used to determine the background.

2) MASH $SNUM $INUM SP=(100,150) BK=(90,99) BK=(151,170)

Does the same as example 1, but this time producing the spectrum from image INUM and storing it in spectrum SNUM, where INUM and SNUM are variables.

3) MASH 18 7 BK=(0,16) SP=(17,99) NORM

Mashes rows 17 through 99 of image 7 into spectrum 18. Rows 0 though 16 in the image determine the backgound. The spectrum is the average (rather than the sum) of the rows in the image.

4) MASH 18 7 BK=(0,16) SP=(17,99) REFLAT SUB

Does the same as example three, but subtracts a polynomial expression for the background, rather than the background spectrum itself. It also subtracts the background spectrum from the SP rows in the original image.

5) MASK 18 7 BK=(0,16) SP=(17,99) COL=(100,200) SKY=20

Does the same as example 3, but extracts only columns 100 through 200, inclusive, and saves the background rows in spectrum buffer 20.

LAMBDA CALIBRATE WAVELENGTH SCALE FROM COMPARISON SPECTRUM

Form: LAMBDA source [FILE=xxx] [ORD=n] [TTY] [INT] [redirection]

where:

source	(integer or $ construct) selects the spectrum used in the calibration,
FILE=	is a file of line identifications,
ORD	determines the order of the polynomial

expressing the relation between channel number and wavelength.

TTY — sends more extensive output to the terminal.

INT — allows interactive selection and weighting of lines.

This command is a concatination of the LINEID and WSCALE commands. It will calculate the wavelength calibration of a comparison spectrum specified by 'source'. The program looks for peaks and attempts to identify them based on an initial estimate of the reciprocal dispersion in Angstroms/pixel and a list of line identifications read in from file in [CCD.SPEC] with name 'xxx.WAV'. Accurate line centers are calculated, and if specified in the data file, partial line blends deconvolved. The line centers are then fit to a polynomial of order specified by 'ORD=n'. If the keyword 'INT' is specified, interactive line identifications can be made and weights can be assigned to the identified lines. The program will ask for all unspecified parameters.

The resulting wavelength scale becomes associated with the original comparison spectrum. You then use the COPW command to associate the wavelength scale with you program spectra.

Examples of command:

1) LAMBDA 4 ORD=3 FILE=NEON

Computes the wavelength scale for spectrum 4, doing a third-order fit to wavelength versus channel number. The wavelengths for the lines are in [CCD.SPEC]NEON.WAV. The resulting wavelength scale is printed on the terminal.

2) LAMBDA 4 ORD=3 FILE=NEON >LP:

Does the same as example 1, but this time sending the output to the line printer.

3) LAMBDA 4 TTY ORD=1 FILE=NEON INT

Does a first order fit to wavelength versus channel number for spectrum 4. The user can interact with the routine as it proceeds.

See the help message on the LINEID command for a description of the wavelength data file.

LINEID IDENTIFY LINES IN A WAVELENGTH CALIBRATION SPECTRUM

Form: LINEID source [FILE=xxx] [ADD] [TTY] [INT] [redirection]

where:

source	(integer or $ construct) selects the spectrum used in the calibration,
FILE=	is a file of line identifications,
ADD	will add newly identified lines to a list of lines from a previous execution of LINEID,
TTY	sends more extensive output to the terminal,
INT	allows interactive selection of lines.

This command will match emission peaks in a wavelength calibration spectrum with their wavelengths supplied in a file. The program looks for peaks and attempts to identify them based on an initial estimate of the reciprocal dispersion in Angstroms/pixel and a list of line identifications read from a file in [CCD.SPEC] with name 'xxx.WAV'. Accurate line centers are calculated, and if specified in the data file, partial line blends deconvolved. The matched lines are saved into a common block in VISTA. The contents of the common block can be examined using the command 'PRINT LINEID'. By default, the previous contents of the common block are replaced with the current identifications. However, if you use the 'ADD' keyword the new identifications will be appended to the older results. This allows you to combine the identifications from several independent wavelength calibration spectra. If the keyword 'INT' is specified, interactive line identifications can be made. Once your line list has been created with LINEID you use the WSCALE command to fit a polynomial to the wavelenghts as a function of pixel number.

Examples of command:

1) LINEID 4 FILE=NEON

Matches lines in spectrum 4 with the wavelenghts supplied in [CCD.SPEC]NEON.WAV. The resulting identifications replace any previously saved identifications in the common block.

2) LINEID 4 FILE=NEON >LP:

Does the same as example 1, but this time sending the output to the line printer.

3) LINEID 4 INT FILE=NEON
LINEID 5 TTY FILE=MERCURY ADD

The first command does the same as example 1, but allows the user to make additional line identifications after the command has made its best attempt. The second command works on a separate mercury lamp spectrum in spectrum buffer 5. The identifications are added to those for the neon spectrum.

The wavelength file has the following format: The file is formatted, with the first line showing the estimated dispersion, followed by any second-order term. The format is free, but the second-order term must be 0.0 if it is to be ignored. Subsequent lines contain an ordered set of line wavelengths in Angstroms. One spectral line is given per file line. The wavelength if followed by a two-letter ID code. If the code is CO, that line is ignored. The code is followed by the wavelengths of any blue-side or red-side satellites which might be blended with the primary line. The wavelengths must be set to 0.0 if there are no lines blended with the primary.

Example file:

```
  7.86       1.0E-03
5881.4900   NE  5852.4900      0.0
5944.8300   NE     0.0       5975.2800
6030.0000   NE     0.0          0.0
6096.1600   NE  6074.3400    6143.0600
```

The file should be stored in the [CCD.SPEC] directory with the extension .WAV.

WSCALE CALIBRATE WAVELENGTH SCALE FROM 'LINEID' OUTPUT

Form: WSCALE dest [ORD=n] [TTY] [INT] [redirection]

where:

dest	(integer or $ construct) selects the spectrum for which the wavelength scale is to apply.
ORD=	determines the order of the polynomial expressing the relation between channel number and wavelength.
TTY	sends more extensive output to the terminal.

INT allows interactive selection of lines. INT implies TTY.

This command will calculate the wavelength calibration using the output generated with the LINEID command. The resulting calibration is associated with the spectrum specified by 'dest'. The line centers determined with the LINEID command are fit to a polynomial of order specified by 'ORD=n'. If the keyword 'INT' is specified, interactive line weighting can be done. The program will ask for all unspecified parameters.

Although the resulting wavelength scale becomes associated with the 'dest' spectrum, you can then use the COPW command to associate the wavelength scale with any other spectrum you wish.

Examples of command:

1) WSCALE 4 ORD=2

Computes the wavelength scale for spectrum 4, doing a second-order fit to wavelength versus channel number. The resulting wavelength scale is printed on the terminal.

2) WSCALE 4 ORD=2 >LP:

Does the same as example 1, but this time sending the output to the line printer.

3) WSCALE 4 ORD=1 INT

Does a first order fit to wavelength versus channel number for spectrum 4. The user can interactively give weights to the identified lines being fit (0 weight discards a line).

COPW COPY A WAVELENGTH SCALE FROM ONE SPECTRUM TO ANOTHER

Form: COPW dest source [source2]

where:

dest is the destination spectrum to which a wavelength scale is to be copied.

source is the source spectrum from which the wavelength scale is to be copied

source2 is a second source spectrum to be used

for wavelength interpolation

This command will copy a wavelength calibration from one spectrum to another. This is typically done to copy the wavelength scale computed with the LAMBDA command from the comparison spectrum to a program spectrum. The actual spectrum data are not changed by this command, only the wavelength parameters. If a second source spectrum is given then the command assumes that you have two comparison spectra which bracket the program spectrum in time, and it will use the times of observations of the three spectra to do a linear interpolation of the wavelength parameters of the comparisons to the time of the program spectrum. This procedure will provide an improved wavelength calibration by removing most of the effects of instrumental flexure.

ALIGN TRANSFER A SPECTRUM TO A NEW WAVELENGTH SCALE

Form: ALIGN source DSP=f W=(w,p) [LOG] [LGI] [MS=n] [FLIP] [V=f] [Z=z]

where:

source	(integer or $ construct) is the spectrum that the program works on.
DSP=f	converts the dispersion to f Angstroms/pixel,
W=(l,p)	sets wavelength l to occur at pixel p,
LOG	transforms to a logarithmic wavelength scale,
LGI	uses 4-pt Lagrangian instead of sinc interpolation
MS=n	sets wavelength scale to be that of spectrum 'n',
FLIP	reverses the direction of the dispersion, and
V=f	removes velocity shift of f km/sec.
Z=z	removes redshift z

This command will transform a spectrum from its original wavelength scale to either a linear wavelength scale or a logarithmic wavelength scale by sinc interpolation. The original spectrum can be on any of the three wavelength scales supported by VISTA, i.e. linear, logarithmic, or polynomial. If the spectrum is on a polynomial wavelength scale then the observed intensities will be scaled to an uncalibrated F-Lambda scale. Before calling this command, a lambda calibration should be done on the appropriate arc spectrum. The results of the calibration are then transfered to the program spectrum using the COPW command. ALIGN is then called to transform the spectrum onto the desired final wavelength scale. You must supply ALIGN with the desired dispersion and wavelength reference point with the keywords 'DSP=f', where 'f' is the dispersion in Angstroms/pixel, and 'W=(w,p)', where 'w' is the desired wavelength at pixel 'p'.

Other keywords provide for options as follows. The word 'MS=n' will force the dispersion to match that of previously aligned spectrum 'n'. 'FLIP' will reverse the order of the dispersion, and 'V=f' will perform a velocity shift of 'f' kilometers/second.

SKYLINE RECALIBRATE WAVELENGTH SCALE USING NIGHT SKY LINES

Form: SKYLINE s1 [s2] [s3] [s4] [s5] [s6] ... [s15]

where:

s1	is the spectrum buffer number of the sky spectrum.
s2 ... s15	are spectrum buffer numbers whose wavelength scales are to be recalibrated using the night sky spectrum s1.

This command examines the night sky spectrum 's1' for suitable night sky emission lines. If it finds at least two lines in the spectrum it will use them to compute a corrected zero-point for the wavelength scale (the dispersion term is not changed). This new zero-point is then applied to all of the other spectra supplied on the command line. All of the spectra should be on the same linear wavelength scale (using the ALIGN command) to begin with.

Example:

```
MASH 1 8 SP=(30,40) BK=(10,20) BK=(50,60) SKY=2
COPW 1 20                  ! Copy wavelength parameters
COPW 2 20
ALIGN 1 DSP=7.0            ! Xform to linear wavelength
ALIGN 2 DSP=7.0
SKYLINE 2 1                ! Correct zero-point
```

MASH is used to operate on image buffer 8, producing a sky-subtracted program spectrum in spectrum buffer 1 and the sky spectrum in spectrum buffer 2. Wavelength parameters are copied from a previously calibrated comparison spectrum (in buffer 20). The two spectra are transformed to a linear wavelength scale using ALIGN. Then the zero-point of the wavelength scale is corrected using SKYLINE. This procedure can remove instrumental flexure if your spectra contain usable night-sky lines.

EXTINCT CORRECT A SPECTRUM FOR ATMOSPHERIC EXTINCTION

Form: EXTINCT source

This command will correct the 'aligned' spectrum 'source' for atmospheric extinction. The zenith angle is measured from the header information, and the air mass calculated with a polynomial in secant(z) to account for the finite thickness of the atmosphere. A set of extinction values are calculated at several wavelengths, and a spline

drawn through them. The spectrum is multiplied by the spline to correct it for extinction.

The longitude and latitude of the observatory at which the image was taken can be entered be defining the DCL symbols V$LONGITUDE and V$LATITUDE before you begin running VISTA. You will need to do this to obtain correct determinations of air mass or extinction when reducing CCD images from other observatories. The longitude and latitude must be in decimal degrees: For example

```
DEFINE V$LONGITUDE "121.64554"
DEFINE V$LATITUDE "37.43029"
```

If these symbols are not defined, the longitude and latitude of Lick Observatory will be used.

FLUXSTAR DEFINE A FLUX CURVE WITH A STANDARD STAR SPECTRUM

Form: FLUXSTAR source [standard] [AVE] [WT=w] [SYSA] [SYSC]

where:

source	(integer or $ construct) is the spectrum used to determine the flux curve,
standard	(character string) is a file containing the flux calibration for the 'source' spectrum,
AVE	averages the current spectrum with previous results, and
WT=w	specifies weighting for the averaging.
SYSA	produces a "System A" (IDS nomenclature) response curve,
SYSC	produces a "System C" (Compromise between A and B) reponse curve.

The FLUXSTAR command generates a flux curve from a standard star spectrum and a file containing the correct flux levels as a function of wavelength. The input spectrum is assumed to be on a linear wavelength scale, corrected for atmospheric extinction, and to have its intensities on an uncalibrated F-lambda scale. The routine locates the flux points given in the input file, and finds the star's average flux over the specified wavelength bin of the flux point. A set of correction points are thus defined, which consist of the correct fluxes, reduced to the Hayes-Latham Vega calibration, divided by the observed intensity of the standard star. A spline is drawn through these points and replaces the standard star spectrum to give the flux

curve.

The FLUX command takes the correction points defined above, and uses a spline to define a flux calibration buffer for the input spectrum. The spectrum is then calibrated by multiplication by this buffer. The separate FLUXSTAR and FLUX commands permit the calibration of spectra on different wavelength scales than the standard star spectrum.

The standard star's flux measurements are read in from a file, which is assumed to be in the [CCD.SPEC] directory, with extension .FLX unless specified otherwise. The file is headed by the stars apparent V magnitude, and then its magnitude at 5445 A. Each line of the file will contain a flux point specified by its wavelength, magnitude per unit frequency, and bin width in angstroms. The points must be in order, but there is no strict format that must be observed.

The keywords 'AVE' and 'WT=' allow the averaging with weights of multiple flux curves. The default weighting, used for the first flux curve as well as for those using 'AVE', is 1. Anytime neither keyword is specified a fresh flux curve is started (again, with a weight of 1.). Note that the averaging can handle two flux curves which overlap (but do not necessarily match perfectly) in wavelength, and can even create a flux curve from two curves with completely disjoint wavelength scales, but it cannot insert flux points in the midst of an existing flux curve.

The SYSA keyword produces a point-by-point flux curve instead of a smooth, spline fitted flux curve. This is done by removing the known stellar absorption lines from the observed standard star spectrum. The atmospheric bands, however, are not removed. The result is that you end up with a flux curve which can correct for the atmospheric bands. A drawback for SYSA is that any "anomalous" absorption lines not known to the program will appear in the response curve and can lead to extraneous features in the data.

The SYSC keyword produces a compromise response curve with the best features of both "system A" and [the default] "system B". In this option the system A curve is smoothly fitted by a spline (at the usual knot points) while the B and A atmospheric bands (+- 65 Angstroms) are retained. The curve is piecewise continuous. It is free from glitches introduced by spurious absorption features in the stellar spectra, retains a higher accuracy outside the extreme knot points, and compensates for the atmospheric bands in the data.

FLUX FLUX CALIBRATE A SPECTRUM

Form: FLUX source

where:

source (integer of $ construct) is the spectrum being worked on.

FLUX replaces a spectrum, which is in some arbitrary units of intensity, with the flux in that spectrum. The spectrum must be on a linear wavelength scale. The flux calibration is taken from a standard star flux curve loaded with the FLUXSTAR command.

Photometry INTRODUCTION TO STELLAR PHOTOMETRY ROUTINES

VISTA has a package of programs designed to produce stellar magnitudes from CCD pictures. These programs are best suited for work in moderately crowded fields where there are photometric standards on each frame, but they may also be used for photometry of single stars.

This section of the helpfile is designed as a reminder to those who already know how to use the photometry routines. If you are a beginner, read this section, and note which commands are mentioned. Study the helpfiles for those commands, then read this section again, particularly the examples below. If things are not clear, please see (or mail) Don Terndrup (at Santa Cruz).

There are three steps in finding the magnitude of a star. It is assumed that all the stars on the CCD frame have the same shape, and differ only in brightness. The first step of the process is the determination of the shape of the stellar images. This is accomplished by the routine PSF. In the second step, the brightness of the stars on the frame relative to the point-spread function is found by a least-squares method (the command for this is FITSTAR). This produces a list of relative magnitudes which are converted to apparent magnitudes in the third step, which is performed by the command MAGSTAR.

There are other commands which make the fitting easier or produce other data: see MASK, MARKSTAR, FITMARK, MODPHOT, and COORDS.

Let's assume that you want to do the whole number on an image: locate all the stars, find their brightnesses, convert them to magnitudes, find the coordinates, and print a neat table of the results. Assume the image you are working on is stored on the disk with name 'IMAGE'. Then you would probably do the following:

```
RD 1 IMAGE                  (read into buffer 1)
MN 1                        (find the mean for the TV)
TV 1 CF=WRMB                (display the image)
MARKSTAR NEW                (create a photometry list;
                             enter standard magnitudes
                             and coordinates)
PSF 2 SIZE=20               (find the PSF and store it
                             in buffer 2)
FITSTAR 2 FILE              (find the stellar brightnesses)
MAGSTAR                     (find the magnitudes)
COORDS                      (compute coordinates)
PRINT PHOT HARD             (print the results)
SAVE PHOT=filename          (save the results)
```

Now you examine image 1, which is the original image with the stars removed. You notice that there is a star or two that you missed marking the first time around. Then you might do the following

sequence.

```
MARKSTAR                          (append to the current list)
FITSTAR 2 FILE INTER              (select only those new stars
                                   in the determination of
                                   brightness)
COORDS                            (do the coordinates again)
MAGSTAR                           (do the magnitudes again)
PRINT PHOT HARD                   (make a new table)
SAVE PHOT=filename                (save the results)
```

Or, suppose you found that the programs failed to find the proper brightness for a star on the frame you just worked on because column 30 was bad.

```
RD 1 IMAGE                        (restore the original image)
TV 1                              (display again)
MASK COL=30                       (tell the program to ignore
                                   column 30)
FITSTAR 2 FILE INTER              (select the star and try again)
```

Another example: You found the brightnesses of the stars two days ago, but did not enter the magnitudes of the standards. You want to do this now, and compute the magnitudes for all the stars in the photometry file.

```
GET PHOT=filename                 (get the old version)
MODPHOT                           (enter the magnitudes)
MAGSTAR                           (compute magnitudes for all)
SAVE PHOT=filename                (save the results)
```

Some experimentation will no doubt be necessary. Don Terndrup will be glad to receive comments about these programs and help you understand them.

MARKSTAR LOCATE STARS

Form: MARKSTAR [NEW] [AUTO] [DR=dr] [DC=dc] [RADIUS=r] [NOBOX]

MARKSTAR creates a 'photometry file', which is a record of the positions, coordinates, and magnitudes of stars on an image. (See below for a complete list of entries in the photometry file). Typing 'new' starts a new list. If you don't type 'new', any stars you mark will be appended to the current list (if there is one); in this case the program will show the positions on the TV of the stars that have already been marked.

MARKSTAR is an AED interactive routine, which automatically operates on the image currently stored in the TV. You mark with the TV cursor those stars you want to include in the photometry file. The

TV draws a box around the stars you have marked so you won't include a star more than once. The position is stored in the photometry file, along with other information which you may enter.

The photometry file is stored internally in the VISTA program as a common block. It is NOT automatically written to the disk. You have to save the results you make with the photometry routines using the SAVE command. Similarly, you can connect a photometry file to the program with the GET command. Read 'PHOTOMETRY' for examples. The photometry files are stored in [CCD.DATA] with the extension .PHO.

There are two ways to operate this program. The first mode lets you interactively mark the positions of the stars:

1) Load an image into the AED with the TV command.
2) Type MARKSTAR or MARKSTAR NEW (with a RADIUS specifier).
3) Move the cursor near a star.
4) Hit 'C' or 'X' on the AED to mark the star.
   The 'X' key defines the star's positon to be exactly the location of the cursor. The 'C' computes an exact position by finding the centroid of the stellar image. You will probably use the 'C' key most of the time. The 'X' key can be used in very crowded fields.
5) Striking the 'N' key after marking a star allows you to enter data about that star (see below).
6) Repeat steps 3 and 4 until all the desired stars are found. At any time you may type 'H' on the AED to get a list of the commands for this program.
7) Hit 'E' on the AED when you are finished.

Using the 'N' key on the AED allows you to enter:
1) a character label for the star
2) the magnitude, if the star is a standard
3) the right ascension and declination

The 'N' key only applies to the star most recently marked by the 'X' or 'C' keys on the AED. Any information you can enter with the 'N' key can be changed with the MODPHOT command.

The second mode is to have the program mark all the stars that are on an already-present photomety file. This saves you time when you have several exposures of the same field (say in several colors). The syntax for the second mode is MARKSTAR AUTO with the other options. The program takes each position on the current photometry photometry file, then looks at the current TV image for a star. If the star is found, the information is stored on a new photometry list, REPLACING the old list. (Save the old one first!) If the stars on the new image are not exactly in the same positions as on the old image, use DR and DC to specify the change that must be applied to the old

coordinates to match the new ones. When the program is finished marking the stars in this automatic mode, it switches to the interactive mode, allowing you to mark more stars.

You can print the contents of a photometry file with the PRINT command. Type 'PRINT PHOT' to see the results on your terminal; type 'PRINT PHOT HARD' to send them to the lineprinter.

It is best to create a new photometry file for each image you reduce. That way there is a one-to-one match between images and files. You do not have to include all the stars in a frame in a photometry file, if you do not so desire, but you are asking for trouble if you have information from several frames in the same file. This will be especially troublesome if you use FITSTAR.

The photometry file contains a series of records of the form:

```
LABEL, (DATA(I), I=1,30)
```

where

LABEL is a character*80 description of the star
DATA are numbers describing the star.

The current definitions of DATA are:

1) Hour angle at midpoint of exposure in seconds of time (always > 0)
2) Frame scale in arcseconds per pixel.
3) Row position of star.
4) Column position of star.
5) Epoch of coordinates.
6) Trial right ascension for star in seconds of time.
7) Calculated right ascension in seconds of time.
8) Calculated declination in seconds of arc.
9) Trial declination in seconds of arc.
10) Peak height of star.
11) Scale ratio: ratio of star to point-spread function.
12) Error in scale ratio.
13) Magnitude, if star is a standard.
14) Non-zero if star is a standard.
15) Error in standard magnitude.
16) Calculated magnitude.
17) Error in calculated magnitude.
18) Background level near star in counts.
19) Background expressed in magnitudes per arcsec.
20) Total counts under the star.
21) Air mass at midpoint of exposure (computed assuming taken at Lick)
22) Exposure time in seconds.
23) Universal time: month
24) UT day.
25) UT year.

26) Ut time at which exposure was taken in seconds.
27) Right ascension of center of frame in seconds of arc.
28) Declination of center of frame in seconds of time.
29)
30)

Note: If you are reducing data which was not taken at Lick, you will have to tell VISTA the longitude and latitude of the observatory where the images were produced. See the section EXINCT (type HELP EXTINCT if you are on a terminal) for instructions.

PSF FIND THE POINT SPREAD FUNCTION FOR AN IMAGE

Form: PSF dest [SIZE=n] [SKY=n] [average]

where:

dest	(integer or $ contruct) is the buffer where the PSF will be stored,
SIZE	specifies the size of the PSF image,
SKY	sets the level of the background subtracted from the PSF before it is stored, and
average	averages (rather then sums) the images of the stars making up the PSF.

This program finds the point spread function for an image. The PSF is the function which describes the image made by a point source, such as a star. This routine empirically finds the PSF by extracting pieces of the TV image and moving them to the image buffer 'dest'.

This is an AED INTERACTIVE routine, which automatically operates on the image currently stored in the TV. To run the program, place the TV cursor near a star you want included in the PSF, then hit 'C' on the AED. The program then finds an exact position, extracts a segment of the image near the star, then draws a box showing the section extracted. This process may be repeated for different stars, in which case the segments of the image are summed.

Comments on the keywords:

dest	This number must be different from the number of the image in the TV. If you select any stars for the PSF, any image previously in buffer 'dest' will be destroyed.

[SIZE=n] If you include this word, the PSF image will be square, having (2 * n + 1) rows and an equal number of columns. If you leave this word off, the program will try to find the size of the stellar images by seeing how the first image falls off from its peak. The algorithm for this is rather primitive, so it is best to use the SIZE keyword. To find the value of n, use the PLOT command to see how large the wings of the stellar images are.

[SKY=n] This optional word defines the sky level, which is subtracted from the PSF before it is written to the output buffer. If you do not define the sky level, the program chooses its own. The default value is taken to be the mean of the pixel values on the perimeter of the PSF.

[average] If you type this word in the command line, the PSF will be the average of the images of the selected stars. Otherwise, the PSF will be the sum of the selected stellar images.

Examples:

Assume the image in buffer 1 is loaded into the TV.

PSF 2 SIZE=15	cuts out a 31 by 31 section of the TV image, and stores it in buffer 2.
PSF 2 SIZE=15 AVERAGE	does the same as the above, but averages the selected stars.
PSF 2	computes the size of the region selected.

Suggestions:

1) The stars which define the PSF should be isolated from other stars, significantly above the noise level, and not too near the edge of the picture.

2) For accurate photometry, the stars which go into the PSF must all have the same shape. Examine the profiles of the stars with PLOT if you suspect that the stellar images are not all of the same shape.

FITSTAR FIND THE BRIGHTNESSES OF STARS

Form: FITSTAR psf [file] [flat] [plane] [inter] [local] [nosub] [radius=r]

FITSTAR finds the brightnesses of stars on an image using a point spread function generated by PSF. It assumes that a stellar image is composed of a background and the PSF multiplied by some constant. The program finds this constant, which is called the 'scaling ratio' and terms in a polynomial expression for the background by least-squares.

The program operates both with and without a photometry file. If you are interested in the integrated brightnesses of single stars and do not care about saving the results, then you do not need to use a photometry file. If, on the other hand, you want to compute magnitudes, fit stars in crowded fields, or save the results for the future, you will need to have a photometry file. These files are created by the program MARKSTAR; see the helpfile for that command for a description of photometry files.

After finding the scaling ratio, the program subtracts the best fit to the star from the input image. That way you can check that the fit was good, and try again if it was not. This feature allows you to remove unwanted stars from an image.

Definition of arguments:

psf	is the image buffer which contains the point spread function for the image being analyzed. See the helpfile for PSF for an explanation of the PSF and how to generate it.
file	tells FITSTAR to find the brightnesses of stars in the photometry file. If you do not put this word in the command line, the program will ask you to select the stars you want, and will not save the results on the disk.
[flat], [plane]	constrains the type of background used in the least-square fit for the brightness of the star. If the word 'flat' is specified, the background will be a constant. If the word 'plane' is used, the background will be a plane surface. If neither is specified, the background will be a second-degree polynomial surface.
[inter]	This word has meaning only if you are using a photomtry file. If this word is included in the command string, the program will ask

you to place a TV cursor on those stars or groups of stars whose brightnesses are being determined. If the word 'inter' is not in the command, all the stars on the photometry file will be fit.
If no file is specified, the program runs interactively.

[local] This word has meaning only if there is an input file. If this is specified, the fitting is done one star at a time. If the word is not specified, the program divides the input image into groups of stars, and finds the brightnesses of all the stars in a given group simultaneously.
If there is no input file, the program runs in the 'local' mode.

[nosub] If this is included, the images of the stars are not subtracted from the input image.

[RADIUS=r] Specifies the region of the image over which the stars will be fitted. The least-square fit will be done inside a circle of radius 'r' around a single star, or on all pixels within distance 'r' from any of several stars being fit simultaneously. If the radius is not specified, the radius is taken to the the size of the PSF image.

Examples of the use of FITSTAR:

Each of the examples below assumes that the PSF for the image has been found, and is stored in buffer 2.

1) Zap out stars. Use a planar background.

FITSTAR 2 PLANE

Move the AED cursor to the star you want removed. Hit 'C' on the AED to find the brightness. Examine the original image. The selected stars should be gone.

2) Find the brightnesses of all stars on an input file. Do not subtract the stellar images from the image. Use a constant background. Fit all the stars one by one.

FITSTAR 2 FILE NOSUB FLAT LOCAL

3) Same as #2, but use the AED cursor to select

the stars to be fitted.

FITSTAR 2 FILE NOSUB FLAT LOCAL INTER

4) Same as #2, but segment the picture into groups of close stars, and fit all the stars in each group simultaneously.

FITSTAR 2 FILE NOSUB FLAT

5) Fit all the stars in the input file. Use a second-degree polynomial background. Subtract the stars from the input image. Segment the picture into groups.

FITSTAR 2 FILE

6) Segment the picture into groups of stars. Select the groups to be fit with the cursor. Use a plane background.

FITSTAR 2 FILE INTER

FITMARK LOCATE STARS AND FIND THEIR BRIGHTNESSES

Form: FITMARK psf [new] [flat] [plane] [nosub]

This program combines the operation of the programs MARKSTAR and FITSTAR, and is used to interactively select stars on a TV image and find their brightnesses. The keywords of this command are defined in the same was as in MARKSTAR and FITSTAR. See the helpfiles for those commands for complete informaton. Briefly:

Use NEW to start a new photometry list. Otherwise the stars that you mark will be added to the current photometry list. Specify the type of backgound used with PLANE or FLAT. The stars that you mark should be isolated from one another. The program cannot fit several stars simultaneously.

COORDS COMPUTE COORDINATES FOR STARS

Form: COORDS

COORDS finds the right ascension and declination for each star in a photometry file. At least three of the stars must have had their positions entered in MARKSTAR or in MODPHOT. The coordinates for the standards must all be at the same epoch. (One day we will have a precession program.)

COORDS works by solving a least-square relation between the

rectangular coordinates on the image (i.e., row and column) and the spherical coordinates on the sky. You can see Smart's book for the formulae, if you dare. The program calculates the coordinates for every star in the photometry file based on the standard positions you have entered, including the standard stars. The input positions and calculated positions for the standards will be displayed on your terminal. The difference between them should be small and evenly distributed about zero. If not, one of your input coordinates is probably wrong. Use MODPHOT to change that coordinate, and try again.

MAGSTAR COMPUTE MAGNITUDES FOR STARS

Form: MAGSTAR

MAGSTAR finds the apparent magnitude of all the stars in a photometry file that have had their brightnesses determined with FITSTAR. Use MARKSTAR or MODPHOT to enter the magnitudes of the stars which you are using as standards.

The program relates the calculated integtrated brightnesses and the magnitudes of the standard stars stars using the relation:

MAG = -2.5 LOG(BRIGHTNESS) + CONSTANT

MAGSTAR solves for the value of the constant by finding a least-square relation between the magnitudes and brightnesses of the standards, and estimates the error in the constant by the scatter of the data points about the best value. The error in the magnitudes of the other stars is computed from the error in the constant and the error in the brightness.

The more standards you enter the better the result will be. The program will work with any number of standards, but the smaller the number, the larger will be the error in the magnitudes. If you enter only one standard magnitude, the program will print a warning message, telling you that the values of the error in the magnitudes are nonsense.

This assumes, of course, that the magnitudes of the standards (which are obtained from some catalog) and the calculated brightnesses of those stars are actually related by the above formula. The program PRINT PHOT lists both the calculated and standard magnitudes, so you can see whether or not the formula holds. Be careful that the standard magnitudes are in the same wavelength system as your observations.

MODPHOT MODIFY ENTRIES IN A PHOTOMETRY FILE

Form: MODPHOT

MODPHOT allows you to change records in a photometry file. The program asks for the number of the record being changed. You can enter or change coordinates, the magnitudes of standard stars, or identifying labels. You can also delete records. You should use the command 'PRINT PHOT HARD' to get a list of the record numbers before you call MODPHOT.

A word of warning: The program renumbers the photometry records if any are deleted. So if you, for example, delete record 17, number 18 becomes 17, 19 becomes 18, etc.

Variables VISTA VARIABLES

The following commands are used to define and display the values of variables:

SET	sets the value of a variable, either directly or in terms of arithmetic operations on other variables.
TYP	displays the value of a variable.
ASK	asks for information to be entered at the terminal.
PRINTF	formatted printing of variable values and character strings.
PRINT VAR	displays all the defined variables.

SET DEFINE A VISTA VARIABLE AND GIVE IT A VALUE

Form: SET var_name=value

where:

var_name	is the name of the variable being defined.
value	it its value.

SET defines VISTA variables in terms of numerical constants, other variables, or the result of arithmetic operations between other variables. The name of a VISTA variable is any alphanumeric string. The value of the variable is a floating point number. VISTA supports an internal variable table which holds variables defined by you or as the output of a program. These variables can be used to pass the results of arithmetic calculations to keywords, to control the flow of a procedure in IF tests or DO loops, or to store convenient numbers in symbolic form.

Each SET command can handle up to 15 definitions. Each definition must include an '=' sign with the name of the new variable to its left, and a defining expression to its right. The expressions can make use of any of 5 operations. The operations are simply read left to right as in reverse polish notation; no brackets can be used. The operations are:

+	Add the next variable to the running result.
-	Subtract the next variable from the running result or negate it.
*	Multiply the running result by the next variable.
/	Divide the running result by the next variable.
^	Raise the running result to the next variable.

Examples:

```
1)  SET X=1.07              Sets the value of X to 1.07
2)  SET Y=-X                Sets Y to -1.07
3)  SET Z=Y+X               Sets Z to -1.07 + 1.07 = 0.0
4)  SET Q2=X^Y+X/Y-Y^2      Q2 is set to:
                            (((((X^Y) +X) /Y) -Y)^ 2)
5)  SET Y=-2.71828E12       Sets the value of Y to -2.71828E12
```

NOTE: All operations are done in single precision floating point. There must be no spaces between the beginning of 'var_name' and the end of 'value'.

TYP TYPE A VARIABLE VALUE ON THE CONSOLE

Form: TYP var_name

This command can be used to print out variable values. Up to 15 variables can be printed out at one time.

Example:

```
SET SKY=1.0
TYP SKY          will show 'SKY =    1.00000' on your terminal.
```

You can get a list of all the defined variables by giving the command PRINT VAR.

ASK ASK FOR A VARIABLE VALUE ON THE CONSOLE

Form: ASK ['An optional prompt in quotes'] var_name

This command can be used to request the input of variable values during the execution of a procedure. When the ASK command is executed, the prompt will be displayed at the terminal until the requested value is typed in. If no prompt is given, the command will respond with 'ENTER var_name : '. Only one value can be requested per ASK command.

Examples:

```
1)  ASK BCKGND       will print 'ENTER BCKGND   : on your screen.
                     When you enter a number and hit RETURN, the
                     value of BCKGND will be set to the number you
                     specified.

2)  ASK 'Enter an estimate for the background >> '  BCKGND
                     will type the prompt 'Enter an estimate for the
                     background >> ' on your terminal, and wait for you
                     to enter a number; the value of BCKGND is set to
                     that number.
```

PRINTF FORMATTED DISPLAY OF VARIABLE VALUES AND STRINGS

Form: PRINTF 'Format string' [VARIABLES] [output redirection]

This command displays character strings and variables in specified formats, thus producing tables of results.

The simplest form of PRINTF is PRINTF 'string'. This prints the specified string. Examples are:

```
PRINTF HELLO              prints          HELLO
PRINTF 'Hello, world'     prints          Hello, world
```

You can print the values of variables by specifying in the character string (1) that a variable is to be printed, and (2) the format for the printing of the variable. The character % in the string does the job. It tells that a variable is to be printed where the % appears. The rest of the word following the % is used to specify the format of the string. The format specifiers are the same as they are in FORTRAN. ANY valid FORTRAN specifier appropriate for displaying numeric values may be used.

Examples:

Suppose we have the variables A with value 1.0 and PI with value 3.14159. Then

```
    PRINTF '%F4.1 %F9.4' A PI
prints
         ' 1.0    3.1416'
     1234 123456789       <-- length and arrangement

    PRINTF '%I6 and %F9.5' A PI
prints
     '     1 and   3.14159'
      123456     123456789   <-- length and arrangement

    PRINTF 'The value of pi is %F9.7' A PI
prints
     'The value of pi is 3.1415900'
                         123456789   <-- arrangement
```

Note that spaces between % specifiers are printed. The output of PRINTF can be redirected.

Procedure INTRODUCTION TO PROCEDURES

VISTA can store several commands in a list and execute them as a program. A list of such commands is termed a PROCEDURE. The list is stored in a special buffer, called the 'procedure buffer'.

Almost any VISTA command that has proper syntax can be used in procedure. The basic commands for creating, storing, and modifying procedures are these:

DEF	defines a procedure.
SAME, END	end the definition.
PEDIT	edits the current procedure buffer.
WP	stores the procedure buffer on the disk.
RP	reads the procedure from disk.
SHOW	displays the procedure buffer
GO	begins execution of the procedure.
IDEF, RDEF	are used to edit procedures.

There are several 'control commands' that effect the operation of a procedure.

VER	executes a procedure line by line, to aid in debugging.
PA	pauses during execution of a procedure.
CALL	runs a procedure as a subroutine.
RETURN	returns from a procedure used as a subroutine.
DO, END_DO	define a loop in a program for execution a given number of times,
GOTO	jumps to another place in the procedure.
:	defines a place to jump to in the procedure.
IF, END_IF	define a block of commands that are executed only under certain conditions.
ELSE, ELSE_IF	control branching for branching that has many options.

Finally, there is the important symbol

#	which is used to generate patterns of characters, subsituting them sequentially in lists, thus allowing permutations of filenames.

This list serves not only as an introduction to those not familiar with procedures, but illustrates the flexibility that procedures give to VISTA programming. A defined procedure eliminates the drudgery of typing repetative commands over and over, but is does much more than that: it greatly expands the functions of VISTA so that new applications do not always require new subroutines. A good familiarity with procedures will make your data reductions more efficient and quicker.

The VISTA program executes the procedure stored in the file defined by the DCL symbol V$STARTUP. For example, if you had defined V$STARTUP through

DEFINE V$STARTUP [.MYPROCS]MYPROC.PRO

before running VISTA, then [.MYPROCS]MYPROC.PRO will be executed as the program begins. Typically, the startup procedure will contain definitions of aliases, the setting of symbol values, or the reading into buffers of repeatedly-used images. This procedure is not saved in the procedure buffer as it is executed.

DEF DEFINE A PROCEDURE

Form: DEF [line_number]

where:

line_number	is the [optional] line number of the beginning of the new definition.

This command signals VISTA to begin a procedure definition. DEF will prompt you with a series of line numbers, beginning with the number specified in the command line. If no number was given, the first command will be on line 1. You are to specify a command for each line. Type the command you want on that line, then hit RETURN. VISTA will check the command for proper syntax, and store it in the procedure buffer. Then VISTA will prompt you with the number of the next line. Continue typing commands in until the entire procedure has been entered. To tell VISTA that you have entered the entire procedure, type END or SAME (q.v.).

Examples:

1)	DEF	begins a procedure definition on line 1.
2)	DEF 10	begins a procedure definition on line 10. Commands on lines 1 through 9, if there are any, are preserved.

END END A PROCEDURE DEFINITION OR EXECUTION

Form: END

When this command is entered during a procedure definition it tells VISTA to leave the procedure-definition mode and to return to the command-execution mode. The command is also saved in the procedure buffer and signals the end of the current procedure when it is executed. If the procedure is executed as a subroutine, the END command, like RETURN (q.v.), tells VISTA to return to the calling procedure.

SAME END A PROCEDURE INSERTION AND KEEP TRAILING LINES

Form: SAME

When this command is entered during a procedure definition it tells VISTA to leave the procedure-definition mode and to return to the command-execution mode. Unlike the END command, however, the SAME command tells VISTA to keep any lines that may have been defined after the insertion point, that is, the commands in the buffer following the insertion point are to be left the same as they were before. The SAME command makes sense only when you are modifying previously defined procedures, and is not saved in the procedure buffer.

SHOW SHOW PROCEDURE BUFFER

Form: SHOW [output redirection]

This command lists out the lines or commands held in the procedure buffer.

WP WRITE A PROCEDURE TO DISK

Form: WP filename

where:

filename is the name of the file that will store the current procedure.

Unless otherwise specified in 'filename', RP will write to the directory [CCD.PROCEDURE] and puts the .PRO extension on the new file.

Examples:

1) WP MEDFLY writes the current procedure to the file [CCD.PROCEDURE]MEDFLY.PRO

2) WP [DEMO]MEDFLY writes to [DEMO]MEDFLY.PRO

3) WP MEDFLY.XYZ writes to [CCD.PROCEDURE]MEDFLY.XYZ

RP READ A PROCEDURE FROM DISK

Form: RP filename

where:

filename is the name of the file that holds the

desired procedure.

The specified file is read into the procedure buffer. Unless otherwise included in 'filename', RP will read from the directory [CCD.PROCEDURE] and assumes the file has the extension .PRO.

Examples:

1)	RP MEDFLY	reads the contents of [CCD.PROCEDURE]MEDFLY.PRO into the procedure buffer.
2)	RP [DEMO]MEDFLY	reads from [DEMO]MEDFLY.PRO
3)	RP MEDFLY.XYZ	reads from [CCD.PROCEDURE]MEDFLY.XYZ

GO START PROCEDURE EXECUTION

Form: GO [loop_count] [filename]

where:

loop_count	(integer) tells VISTA to execute the procedure this many times.
filename	loads the specified procedure into the procedure buffer and begins execution.

GO tells VISTA to start executing the procedure held in its procedure buffer. You may also simultaneously load a procedure from the disk and begin execution. In this case, any procedure in the original buffer will be written over. To call in and execute a procedure without writing over the main buffer, see the CALL command. The GO command makes sense only when starting a procedure from the VISTA command-execution mode.

Examples:

GO	executes the procedure in the buffer.
GO 3	executes the procedure 3 times.
GO FIXIT	loads the procedurea FIXIT from the disk, and begins executing it.

PEDIT EDIT THE PROCEDURE BUFFER

Form: PEDIT

The command EDIT loads the proceudre buffer into a temporary file in your current directory, then runs a process which allows you to edit it with the EDIT/EDT editor. If you leave the

editor with EXIT, the modified procedure is loaded back into the procedure buffer, but not executed. If you leave the editor with QUIT, the procedure buffer is not altered.

As the procedure is loaded back into the procedure buffer, the lines are not checked for proper syntax as they are in DEF or IDEF.

RDEF REMOVE LINES FROM A DEFINED PROCEDURE

Form: RDEF line_number

where:

line_number is the line number you want removed.

You can specify as many as fifteen lines to be removed on the RDEF command line. After the desired lines are removed, the remaining procedure lines are renumbered.

Examples:

1) RDEF 3 removes line number 3.
2) RDEF 7 8 9 10 11 12 removes lines 7 through 12, inclusive.

IDEF INSERT LINES INTO A DEFINED PROCEDURE

Form: IDEF [line_number]

where:

line_number is the procedure line BEFORE WHICH the new lines will go

This command allows you to insert new lines into a command procedure. Use the 'SAME' command to terminate the new procedure definition if you do not want to write over the procedure lines following the insertion. Use the 'END' command if you want to delete all the remaining lines in the procedure. If you do not specify a line number, the new lines will be inserted in front of line 1.

Example: Suppose the procedure in the buffer is

```
1   RT 1 4
2   BL 1
3   SC 1 0.75
4   TV 1 CF=WRMB
5   END
```

Typing 'IDEF 4' and entering 'MN 1' then 'SAME' would change the procedure to.

```
1    RT 1 4
2    BL 1
3    SC 1 0.75
4    MN 1
5    TV 1 CF=WRMB
6    END
```

If 'END' had been entered instead of 'SAME', the last command would be number 4.

CALL CALL IN AND EXECUTE A PROCEDURE AS A SUBROUTINE

Form: CALL procedure_filename

where:

procedure_filename is the name of a file holding a procedure.

The CALL command tells VISTA to save the contents of its current procedure buffer, read in the desired procedure file, and begin execution at its first line. The CALL command can be executed directly in the immediate input mode, or be used inside procedures to call other procedures. In both cases, at the completion of the called procedure, VISTA will return properly to either the input mode or calling procedure. VISTA will support up to 10 levels or subroutine calls. If an error occurs while a called procedure is executing, VISTA will unwind and display the complete subroutine stack.

RETURN RETURN FROM AN EXECUTING PROCEDURE

Form: RETURN

This command tells VISTA that the execution of the current sub-procedure is complete and to return to any calling procedure or to immediate input mode as is appropriate. This command is intended to allow a return from the procedure as a result of an IF test. In cases where no condition testing is needed, the final END command in the procedure buffer will suffice to tell VISTA that the procedure has completed.

VER VERIFY AN EXECUTING PROCEDURE

Form: VER Y or VER N

VER causes each line of the procedure to be shown on the terminal just prior to its execution, allowing you to watch the procedure work line by line. The keyword 'Y' turns the display on and the keyword 'N' turns the display off.

PA WAYS TO PAUSE DURING A PROCEDURE EXECUTION

Forms: PA 'prompt message'
ctrl-C

When the PA command is encountered in a procedure, VISTA prints the prompt 'PAUSE ' followed by the rest of the (optional) prompt which appears on the command line. The execution of the procedure is then stopped. While paused, you can enter any commands in the normal immediate mode of execution. To resume the procedure where it paused type the single letter command C. You do not need to type a command to permanently halt the procedure. If you give the command GO any previous pause state will be canceled, and VISTA will start the program from the beginning.

You may also pause an executing procedure by typing 'crtl-c'. (Hold down the 'control' and the 'C' keys simultaneously.) The procedure will then pause after it completes the operation it is currently working on. VISTA will also print the next procedure line to be executed so that you can easily determine where in the procedure the pause occurred. To resume the procedure where it paused type the single letter command 'C'.

GOTO JUMP TO A SPECIFIED PLACE IN A PROCEDURE

Form: GOTO label_name

where:

label_name is a label defined somewhere else in the procedure.

GOTO tells VISTA to jump to a line in the procedure buffer beginning with label_name:. The label_name can be any alphanumeric string. You can jump out of but not into a procedure DO loop or IF block. In the latter cases, VISTA will raise an error conditon and stop the procedure execution. The jump can be in either direction in the procedure buffer, that is to both higher and lower line numbers.

Example 1: GOTO WHEREVER
Any number of procedure lines...
WHEREVER:
The next commands to be executed...

Example 2: NOWHERE:
Any number of procedure lines...
GOTO NOWHERE

: LABEL A LINE AS A GOTO JUMPING POINT

Form: label_name:

To label a procedure line as a place for the GOTO command to jump to, the line must start with the label_name string and a ':' immediatly following the label. No other commands can appear following the label on the same line.

Examples are

1)
```
LOOP:
GOTO LOOP
```

2)
```
DO_IT_AGAIN:
GOTO DO_IT_AGAIN
```

```
DO              BEGIN 'DO' LOOP IN PROCEDURE
END_DO          END 'DO' LOOP IN PROCEDURE
```

Form:
```
DO var=N1 N2 [N3]
   {any vista commands}
END_DO
```

where:

var	is a variable name,
N1	is the initial value of the variable,
N2	is the final value,
N3	is the increment by which N1 is adjusted in each pass through the DO loop.

The DO commands enable you set set up repeatable groups of commands within the procedure buffer. The VISTA DO-LOOP is very similar to the FORTRAN-77 DO-LOOP.

The variable 'var' is initially set equal to the starting value N1. When the END_DO statement is encountered the value is changed by an amount equal to N3. The value of N3 can be either positive or negative. If N3 is positive then the looping terminates when N1 becomes greater than N2. If N3 is negative then looping terminates when N1 becomes less than N2. If N3 is not specified then it defaults to +1.0 if N2 is greater than N1 or to -1.0 if N2 is less than N1.

N1, N2, and N3 can all be arithmetic expressions as described in the SET command. The value of 'var' can be changed within the loop without affecting the do-loop operation. However, VISTA will reset 'var' to its appropriate loop value at the beginning of each loop.

The underline is required in END_DO because VISTA requires commands to be one word long. Up to 20 do-loops can be nested. Do-loops are recognized only within procedures. The GOTO command can be used to jump out of a DO loop, but VISTA will not permit jumping into one. Further, DO loops must contain or be contained completly within any IF blocks.

```
Example 1    DO I=1,3
                    Any number of procedure lines.  These lines
                    are executed 3 times.
             END_DO

Example 2    DO Q=1,N
                    Any number of procedure lines.  These lines
                    are exectued N times.  Here N is a variable
                    that has had its value set by the SET command.

Example 3    DO B=D+I,N-J,-1
                    Any number of procedure lines.  The counter
                    decrements from D+I to N-J.
             END_DO
```

```
IF              'IF' CONDITION TESTING AND BRANCHING IN PROCEDURES
ELSE_IF         'IF' CONDITION TESTING AND BRANCHING IN PROCEDURES
ELSE            'IF' CONDITION TESTING AND BRANCHING IN PROCEDURES
END_IF          'IF' CONDITION TESTING AND BRANCHING IN PROCEDURES
```

VISTA procedures allow testing of variables and branching based on the results of those tests. This capability greatly expands the usefulness of procedures.

The simplest use of IF is to mark a section of a procedure that is executed only if come condition is true. It has the form:

```
IF condition
        Procedure lines (any number) that are executed if the
        specified condition is met.
END_IF
```

You can also have two level branching:

```
IF condition
        Procedure lines to be executed if the condition
        is true.
ELSE
        Procedure lines to be executed if the condition
        is false.
END_IF
```

These may be strung together:

```
IF contition_1
        Procedure lines to be executed if condition_1 is true.
ELSE_IF condition_2
        Procedure lines to be executed when condition_1 is
        false and condition_2 true.
.
.
.

ELSE_IF condition_N
        Procedure lines to be executed when all conditions
        are false except condition_N.
ELSE
        Procedure lines to be executed if and only if
        all other conditions are false.
END_IF
```

The tests made by the IF and ELSE_IF statements are relations between VISTA variables. The possible IF tests are listed below. The ELSE_IF tests are identical to these. A and B can represent single VISTA variables or algebraic expressions involving several.

```
IF A>B                  Test A greater than B
IF A>=B                 Test A greater than or equal to B
IF A=B                  Test A equal to B
IF A~=B                 Test A not equal to B
IF A<=B                 Test A less than or equal to B
IF A<B                  Test A less than B
```

There are two logical conjunctions & (and) and ! (or) which can be used to join several of the above tests. VISTA simply reads the tests from left to right. There are no parentheses and no spaces permitted in the expressions. Examples of the conjunctions are below:

```
IF A>B&A=C              Test A > B and A = C
IF A=B!C<D&C=B          Test (A = B or C < D) and C = B
```

The syntax of the IF statements is designed to look similar to the FORTRAN-77 IF block structures. Each IF block must begin with an IF command and end with the END_IF command. An algebraic relation between VISTA variables to be tested must follow the IF on the same line. If the relation is true, then the procedure commands following the IF command are executed. If the relation is false, VISTA looks for any ELSE_IF tests, any final ELSE statement, or jumps to the procedure lines following the END_IF statement.

The ELSE_IF command also must have a condition to be tested on the same line. ELSE_IF's are optional, but permit you to test other conditions and execute other blocks of the procedure buffer in the event

that the initial IF or any preceding ELSE_IF's are false. In this way you can allow VISTA to 'trickle' down through several tests looking for one that is true.

The ELSE statement is also optional and marks a set of procedure lines for VISTA to execute if and only if the inital IF and any following ELSE_IF's all test out false. Basically, the IF, any ELSE_IF's, or any ELSE statements all mark out various blocks of the procedure to be executed under different conditions. After the execution of any block, VISTA transfers control to the procedure lines following the END_IF statement.

IF blocks can be nested within other IF blocks up to 15 levels deep. IF blocks can be jumped out of, but not into, by the GOTO command. IF blocks must contain or be contained within DO loops completely. Some examples of IF blocks are given below:

```
Example 1:   IF X>Y
                     Do these procedure lines if X is greater than Y
             END_IF

Example 2:   IF X>Y&X<Z
                     Do these procedure lines if X is greater than Y
                     but less than Z.
             ELSE
                     Otherwise jump to these procedure lines.
             END_IF

Example 3:   IF SKY-LIMIT>BACKGRND
                     Do these procedure lines if IF test true.
             ELSE_IF BACKGRND=0
                     Do these procedure lines if IF test false,
                     but ELSE_IF condition true.
             END_IF
```

$ EXECUTE A VAX 'DIGITAL COMMAND LANGUAGE' INSTRUCTION
VAX_DCL EXECUTE A VAX 'DIGITAL COMMAND LANGUAGE' INSTRUCTION

Form: $ Any valid DCL command

You may exectute most VAX commands directly from VISTA by prefixing them with a '$' sign or with the command 'VAX_DCL'. These commands can also be included as part of a procedure and can be modified or updated with the VISTA '#' substitution command.

If the command line is '$' only, you go into DCL command mode and stay there until you type 'logout'. It is dangerous to run large programs while you are in this mode. When you type 'logout', you will receive a message 'PROCESS LOGIN_1 logged out at (time of day)', where LOGIN is the name of the account you are currently using.

Examples:

1)	$SHOW USERS	Exectutes the command 'SHOW USERS'.
2)	VAX_DCL SHOW USERS	Does the same thing.
3)	$DIR [CCD]*.CCD	Gives a list of the CCD images on disk.
4)	$	Go to DCL mode and stay there until 'logout'

See 'dcl' in the section 'Examples and applications' for a list of DCL commands that might be helpful.

BELL TURN THE BELL PROMPT ON OR OFF OR RING THE BELL

Forms: BELL Y, BELL N, or BELL R

The VISTA prompt can be accompanied by a bell, if desired.

N prevents the bell from ringing
Y restores the bell with the prompt
R rings the bell

When you begin running VISTA, the bell does not ring with the prompt. The 'BELL R' command is helpful in procedures for signalling the completion of some process.

Sessions SAMPLE SESSIONS WITH VISTA

Shown below are several sequences of VAX and VISTA commands designed to familiarize you with the operation of the VISTA program. VAX commands are indicated by a $, while VISTA commands are indicated by GO:.

To run VISTA, type the VAX command

RUN [CCDEV.BASE]VISTA

Here is a session examining VISTA images and display. Type these commands in the order presented. After each command, note what has happened. IMPORTANT: Read the help-file for each command before you use it.

--- Turn on the AED and run VISTA ---

```
$DIR [CCD].CCD                      This command looks for all CCD images
                                  on the disk.  The $ is part of the
                                  VISTA command.  As the list of images
                                  scrolls off, pick one of the names.
                                  Suppose you picked M15.CCD.  Let's
                                  read it into the buffer.
GO: RD 1 M15                            Read into buffer 1.
GO: BUF                                 Show list of buffers
GO: COP 2 1                       Copy into buffer 2.
GO: BUF                                 Show buffer list.
GO: CLOSE 1                       Delete image 1.
GO: BUF                                 Show list of buffers.
GO: CH 2 'NEW NAME'               Change the name of the command.
GO: BUF                                 Note the changed name.
GO: COP 1 2                       Copy back into buffer 1.
GO: MN 1                          Show mean of that image.
GO: TV 1 CF=WRMB                  Display on the TV.
GO: ITV                                 Interact with the image on the TV.
    Use the AED keys (I, O, P, R, C, D, and E)
GO: Q                                   End of session.
```

Here we illustrate boxes and the use of the WIND command. It is assumed that the image we are using has at least 200 rows and 200 colums. We'll play with the image M15, but you can use any image you like.

```
GO: RD 1 M15                            Load into buffer 1.
GO: COP 2 1                       Preserve original copy.
GO: BOX 1 SR=100 SC=100 NC=50 NR=50 Define box 1.
GO: PRINT BOX                           Show the box parameters.
GO: MN 1                          Take the mean.
GO: TV 1 CF=WRMB                  Display the original image.
```

```
GO:  TV 1 BOX=1                          Show only the image subsection.
GO:  WIND 1 BOX=1                  Cut down the size of image 1, leaving
                                   only the part within box 1 intact.
GO:  TV 1                          Show the cut-down image.  This is
                                   the same as the previous image
GO:  Q                                   End of session.
```

Here is a session exploring VISTA variables and arithmetic between them. The commands illustrated here are SET and TYP. The ! are used as comments in the VISTA line itself.

```
GO:  SET A = 1.0
GO:  TYP A
GO:  SET B = 2.0
GO:  TYP B
GO:  SET SUM=A+B                         ! Result = 2.0
GO:  SET DIFF=A-B                  ! Result = 0.0
GO:  SET PROD=A*B                  ! Result = 2.0
GO:  SET QUOT=A/B                  ! Result = 1.0
GO:  TYP SUM DIFF PROD QUOT        ! Show results
GO:  SET C=3.0
GO:  SET X=A+B+C                         ! Result = 6.0
GO:  TYP X
GO:  SET X=A+B*C                         ! Result = (1.0 + 2.0) * 3.0 = 6.0
GO:  TYP X
GO:  SET X=A-B*C                         ! Result = (1.0 - 2.0) * 3.0 = -3.0
GO:  TYP X
GO:  SET X=A^B+C                         ! Result = (1.0 ^ 2.0) + 3.0 = 5.0
GO:  TYP X
GO:  PRINT VAR                           ! Show all defined variables
GO:  Q                                   ! Halt
```

The following is an example of the defining, running, and saving a procedure. The procedure here prints the squares and cubes of the first N integers, where N is input from the procedure. The commands illustrated here are ASK, DO, END_DO, SET, TYP, GO, and SHOW.

```
$  RUN [CCDEV.BASE]VISTA
GO:DEF
1  ASK 'Print the squares and cubes of the first N integers, where N = ' N
2  DO J=1,N                        ! Begin DO loop
3  SET SQR=N^2                     ! Compute the square
4  SET CUB=N^3                     ! Compute the cube
5  TYP N SQR CUB          ! Show the result
6  END_DO                 ! End DO loop
7  END                             ! End of definition
GO:  SHOW                          ! Display entire procedure
GO:  GO                            ! Run the procedure
GO:  WP TESTPROC                   ! Write procedure to disk
GO:  GO TESTPROC                   ! Load procedure and GO
GO:  GO 5 TESTPROC     ! Run the procedure 5 times
```

GO: Q

Flat FLATTENING AN IMAGE

The pixels in a CCD do not all have the same quantum efficiency: if an image were exposed using uniform ('flat') illumnination, the response would not be uniform. The first step in the analysis of an image is to 'flatten' it, to correct for the non-uniformity of the response of the chip. This is done, as with all detectors, by taking an exposure with uniform illumnation. This is termed the 'flat-field' exposure. Dividing any other image by the flat-field exposure will correct for the non-uniformities.

CCD's, although linear, usually exhibit what is called a 'zero-offset'. This means that the first few photons striking a pixel give no response. Thus, the function representing the response as a function on illumination does not have its intercept at zero. To correct for this zero-offset, you add a constant to each image before dividing by the flat-field image. The zero offset can be found by recording several flat-field images that have different illumination, and noting the intercept of the response vs. exposure time curve.

Here is a procedure for flattening images. Note the use of comments in the procedure lines. It reads an image and a flat-field image from a tape, does the baseline procedure on both, adds the zero-offset, and divides the image by the flat-field image. The result is stored in buffer 1.

```
ASK 'What is the zero offset ?' ZEROFF              ! Get zero offset
ASK 'Which image do you want ?' IMNUM               ! Get image number
ASK 'Which number is the flat?'     FLNUM           ! Get flat number
RT 1 IMNUM                                  ! Read image
RT 2 FLNUM                                  ! Read flat image
BL 1                                                ! Do baseline corrections
BL 2
AC 1 ZEROFF                                 ! Add zero offset
AC 2 ZEROFF
MN 2                                                ! Compute mean
DI 1 2 FLAT                                 ! Perform scaled division
END                                         ! End of procedure
```

Now, image flattening is not really so well-determined. The above procedure usually produces moderately good results, but you may have to experiment, or talk to an experienced CCD observer to learn all the lore about flattening images: it's a bit of an art.

dcl HELPFUL VAX COMMANDS TO RUN FROM VISTA

Here are some commands that may be of service:

1) $DIR — use the DIRECTORY command to list images, procedures and data files.

2) $EDIT — use the EDIT command to change (or create!) procedures and formatted data files.

3) $SHOW QUOTA — tells you how much room you have left on your account.

4) $DELETE/CONFIRM — use this to delete unwanted files from the disk. BE CAREFUL!!! to delete only your own files, and only those you do not want.

5) $SET TERM/VT100 — can be used, if not already done, to set your terminal to VT100 status. This is necessary for screen-editing of files, and for the PIC, PLOT, and LINE commands.

Index of Commands and Topics

Commands are listed in UPPER CASE.
Topics are listed in capitals and lower case.