A Beginner's Guide to Using IRAF
IRAF Version 2.10

Jeannette Barnes

Central Computer Services
National Optical Astronomy Observatories††
Tucson, Arizona 85726
August 1993

ABSTRACT

IRAF is a general Image Reduction and Analysis Facility providing a wide range of image processing tools for the user. IRAF is a product of the National Optical Astronomy Observatories and was developed for the astronomical community although researchers in other scientific fields have found IRAF to be useful for general image processing. This document is intended to provide the novice user with a brief introduction to IRAF and its Command Language (CL) user interface.

The author would like to thank the members of the IRAF group for comments and suggestions during the preparation of this document.

DRAFT

††Operated by the Association of Universities for Research in Astronomy, Inc. under cooperative agreement with the National Science Foundation.

CONTENTS

A Beginner's Guide to Using IRAF
IRAF Version 2.10

Jeannette Barnes

Central Computer Services
National Optical Astronomy Observatories††
Tucson, Arizona 85726
August 1993

1. Introduction

1.1. Overview

IRAF is a general Image Reduction and Analysis Facility providing a wide range of image processing tools for the user. IRAF is a product of the National Optical Astronomy Observatories (NOAO) and was developed for the astronomical community although researchers in other scientific fields have found IRAF to be useful for general image processing. This document is intended to provide the novice user with a brief introduction to IRAF and its Command Language (CL) user interface.

Only the IRAF "core" system will be discussed in this document. The NOAO science software (general spectroscopic and photometric reduction and analysis tools) is included in all IRAF tributions, but a discussion of this software is better covered elsewhere.

The IRAF core system provides the user with a wide selection of image processing tools using a command line interface. Commands, called tasks, are executed to perform various functions; each task has a parameter file that the user can modify to affect the output of the task. Highlights of the system include:

- similar environment regardless of host system
- query and command line interface
- pipes and redirection
- background job queue
- extensive online help
- interactive graphics and image display
- text and graphics hardcopy
- tape I/O
- FITS image I/O
- several user programming environments
- world coordinate systems

Additional IRAF software packages, called layered packages, are available from a variety of sources and can be installed to run under the IRAF environment. See Appendix F for more information.

---

††Operated by the Association of Universities for Research in Astronomy, Inc. under cooperative agreement with the National Science Foundation.

1.2. Getting IRAF

IRAF is distributed by NOAO and is available over the Internet free of charge to any interested party. Taped distributions are available for a small handling fee. The IRAF system runs on a wide selection of UNIX-based and VMS platforms. See Appendix E for details about acquiring the IRAF distribution.

1.3. Documentation

A large selection of IRAF documentation is available. A selected list of documents is provided in Appendix G. All IRAF documentation (excluding the online help/manual pages, see §3.3.2.), is available in the IRAF network archive in the *iraf/docs* directory. A *README* cross-reference file lists the documents and their file names as they appear in the archive. These files are in either compressed PostScript or compressed ASCII format. See Appendix E for more information about the network archive. Printed copies of these documents are available upon request.

An *IRAF Newsletter* is published periodically, and it is a valuable source of information for even the casual IRAF user. Old newsletters are available in the IRAF network archive. Anyone can receive the *Newsletter*; simply send your name, address, and email address to *iraf@noao.edu* and request this service. The *IRAF Newsletter* is distributed by posted mail; an electronic news service will be implemented shortly as well.

1.4. IRAF support services

General questions concerning IRAF can be directed to *iraf@noao.edu* or *5355::IRAF*. For those of you wishing to talk to someone directly the IRAF Hotline number is (602)-325-4160. Posted inquiries should be sent to Jeannette Barnes, NOAO, P.O. Box 26732, Tucson, AZ 85726 USA.

1.5. Style conventions

The following conventions are used throughout this document.

- IRAF tasks and package names are in CAPITAL LETTERS in the running text of this document. Since IRAF is case sensitive the user should be aware that the capital letters are being used only as a *visual* aid to the reader.
- Terminal output is in `typewriter` font.
- Input parameters that require local values to be substituted in their place in order for the example to execute properly are in `<typewriter>` font but enclosed in "angle brackets", as shown.
- Text that you type in exactly as shown is in **`typewriter bold`** font.
- References, directories and file names, parameters and parameter values, and other important words and phrases are in *italics*.
- The `cl>` prompt is used in most IRAF examples. The prompt the user actually sees on the terminal will depend on the last IRAF package loaded.
- An image of the galaxy M51 is included with all IRAF distributions and is often used in the examples in this document. The image is called *dev$pix*; the same image but with a different header file is also used in some examples and is known as *dev$wpix*.
- The examples in this document assume that the needed packages have been loaded at IRAF startup time, as defined by the distributed IRAF system (see §3.2.).
- For the sake of brevity, the pronoun "he" is used in a generic sense, to include all users, be they female or male.

2. Getting started

2.1. Host startup files

The host level files necessary to run IRAF will vary depending on the host machine and window environment in which you plan to execute IRAF. A workstation environment (SunView, OpenWindows, DECwindows, X Windows) will require additional host level files than those used by the conventional graphics terminal environment. A discussion of the host level files used in the workstation environment is beyond the scope of this document.

All UNIX users need to be sure that either the *.login* or the *.cshrc* files in their UNIX home directories contain the "path" of the directory containing the IRAF startup commands as defined at IRAF installation time (see your local IRAF system manager if you have trouble starting up IRAF). A "path" statement might look like the following.

```
set path = (. /local/bin /usr/ucb /bin /usr/bin /usr/local/bin)
```

No special host level files are required by VMS/IRAF in a conventional graphics terminal environment.

2.2. IRAF startup files

Each user must execute the command **`mkiraf`** at the host level before logging into IRAF for the first time†. This task must be executed in the IRAF "login" or "home" directory which does not necessarily have to be the same as the host "login" directory. You do not have to do this each time you start an IRAF session - only the first time and after that only when it is suggested that you do so, usually after a new version of IRAF has been installed at your site.

The MKIRAF command creates a file called *login.cl* (see Figure 1) and a subdirectory called *uparm*. The *login.cl* file is executed at IRAF startup time, and the *uparm* subdirectory is used by IRAF to save your customized parameter files (see §3.5. for more information about parameter files). MKIRAF will prompt for two things.

1. If you have already done a MKIRAF at some previous session then you will be asked if you want to "Initialize uparm?" Generally answer "no" unless you are absolutely sure you know what you are doing. One of the useful features of IRAF is that it "learns" parameters by storing them in the *uparm* directory - initializing this directory literally wipes out all of these customized parameter files! We do recommend that you answer "yes" to this question if you are doing a MKIRAF after a major new installation of the system.
2. Now you will be asked to "Enter terminal type:" - enter here what your graphics terminal type is, i.e., **`vt640, gterm, xterm`**, and so forth. See §3.1.1. for more information on setting your terminal type.

MKIRAF saves the old *login.cl* file from a previous MKIRAF as *login.cl.OLD*. Any changes that were made to this old file by the user in order to customize the previous IRAF login must be merged back into the new one.

The user can also create a *loginuser.cl* file (see Figure 2) that will be executed by the *login.cl* file (see Figure 1) at startup time. Commands in the *loginuser.cl* file supersede those commands in the *login.cl* file that are executed prior to the statement that calls the *loginuser.cl* file itself. The *loginuser.cl* file is not affected by a new MKIRAF and so this file is the preferred way to customize your IRAF login. The important thing to note about this file is that the last statement in it must be `keep`.

---

†VMS users will need to type **`IRAF`** and then **`MKIRAF`**. Alternatively `$ IRAF` can be entered into the *login.com* file for execution at VMS login time. Otherwise it will be necessary to type **`IRAF`** at the beginning of each session.

```
# LOGIN.CL -- User login file for the IRAF command language.

# Identify login.cl version (checked in images.cl).
if (defpar ("logver"))
    logver = "IRAF V2.10EXPORT April 1992 revision 1"

set	home		= "/home/jbarnes/iraf/"
set	imdir		= "/ursa/scr3/iraf/jbarnes/"
set	uparm			= "home$uparm/"
set	userid		= "jbarnes"

# Set the terminal type.
if (envget("TERM") == "sun") {
    if (!access (".hushiraf"))
	print "setting terminal type to gterm..."
    stty gterm
} else if (envget("TERM") == "xterm") {
    if (!access (".hushiraf"))
	print "setting terminal type to xterm..."
    stty xterm nl=44
} else {
    if (!access (".hushiraf"))
	print "setting terminal type to gterm..."
    stty gterm
}

# Uncomment and edit to change the defaults.
#set	editor		= vi
#set	printer			= lw
#set	stdimage	= imt800
#set	stdimcur	= stdimage
#set	stdplot			= lw
#set	clobber			= no
#set	filewait	= yes
#set	cmbuflen	= 512000
#set	min_lenuserarea	= 24000
#set	imtype			= "imh"

# IMTOOL/XIMAGE stuff.  Set node to the name of your workstation to
# enable remote image display.
#set	node		= ""

# CL parameters you mighth want to change.
#ehinit	= "nostandout eol noverify"
#epinit	= "standout showall"
showtype = yes

# Default USER package; extend or modify as you wish.  Note that this can
# be used to call FORTRAN programs from IRAF.

package user
```

Figure 1. The default *login.cl* file.

```
task   $adb $bc $cal $cat $comm $cp $csh $date $df $diff      = "$foreign"
task   $du $find $finger $ftp $grep $lpq $ls $mail $make      = "$foreign"
task   $man $mon $mv $nm $od $ps $rcp $rlogin $rsh $ruptime   = "$foreign"
task   $rwho $sh $spell $sps $strings $su $telnet $tip $top   = "$foreign"
task   $touch $vi $emacs $w $wc $less $rusers $sync $pwd      = "$foreign"

task   $xc $mkpkg $generic $rtar $wtar $buglog                = "$foreign"
#task  $fc = "$xc -h $* -limfort -lsys -lvops -los"
task   $fc = ("$" // envget("iraf") // "unix/hlib/fc.csh" //
           " -h $* -limfort -lsys -lvops -los")
task   $nbugs = ("$(setenv EDITOR 'buglog -e';" //
           "less -Cqm +G " // envget ("iraf") // "local/bugs.*)")
task   $cls = "$clear;ls"

if (access ("home$loginuser.cl"))
    cl < "home$loginuser.cl"
;

keep;   clpackage

prcache directory
cache   directory page type help

# Print the message of the day.
if (access (".hushiraf"))
    menus = no
else {
    clear; type hlib$motd
}

# Delete any old MTIO lock (magtape position) files.
if (deftask ("mtclean"))
    mtclean
else
    delete uparm$mt?.lok,uparm$*.wcs verify-

# List any packages you want loaded at login time, ONE PER LINE.
images          # general image operators
plot            # graphics tasks
dataio          # data conversions, import export
lists           # list processing

# The if(deftask...) is needed for V2.9 compatibility.
if (deftask ("proto"))
    proto       # prototype or ad hoc tasks

tv              # image display
utilities       # miscellaneous utilities
noao            # optical astronomy packages

keep
```

Figure 1. The default *login.cl* file (cont.)

```
# customize IRAF login


# force the graphics buffer to be very large
        set cmbuflen = 256000
        gflush

# some useful routines outside of IRAF - declare a foreign task
        task $mongo = "$foreign"

# define the script task mt2ex.cl
        task mt2ex.cl = /u2/jbarnes/iraf/util/mt2ex.cl

# load some packages at startup time
        noao
        imred
        ccdred

# define pixel directory
        set imdir=/tmp3/jbarnes/pixels/

# set printer and plot device
        set printer=lw5
        set stdplot=lw5

# add other commands here but before keep


keep
```

Figure 2. An example of a *loginuser.cl* file for a UNIX host.

2.3. Logging in/out of IRAF

Once the host and IRAF level startup files are in place the user simply logs into IRAF by typing **`cl`**. It is important that you log into IRAF from your IRAF "login" or "home" directory as mentioned in §2.2. At this time, the commands in the *login.cl* file (and in the *loginuser.cl* file, if it exists) are executed.

If you are running IRAF from a workstation environment then you will want to first startup the window environment and then type **`cl`** in a terminal emulator window such as *gterm* or *xterm*.

Logging out of IRAF is done by typing **`logout`**. If you are running IRAF from a window environment it is important that you first **`logout`** of IRAF and then shut down the window environment. This provides a clean exit from IRAF and allows the system to shut down in an orderly manner.

3. The basics

3.1. Setting up your IRAF environment

Once the user has logged into IRAF he may want to customize his IRAF environment for that session. The *loginuser.cl* file can also be used for this purpose (see Figure 2) at IRAF startup time.

The IRAF CL maintains a table of "environment variables" which affects the operation of many IRAF tasks. A complete list of these variables can be viewed by typing the following (the output of SHOW is "piped" to the input of the PAGE command, a simple task for looking at one page of text at a time):

```
cl> show | page
```

The complete list is a little overwhelming but there is a small set of these variables that you may want to modify to customize your IRAF environment. The SET or RESET commands can be used for this purpose.

The SET command only modifies the variable temporarily, i.e., while the current package is loaded (see more about packages in §3.2.). Using RESET will normally change the value of the variable for the remainder of the current IRAF session.†

IRAF environment variables can also be used to define IRAF logical names for directories. These logical names can be strings denoting either host level pathnames or other IRAF logical names. For example, the test image, *dev$pix*, that is used in many of the examples in this document, can be traced to the following host directory with the SHOW task. The *iraf* logical directory will be a different host level pathname for each installation, but the *dev* logical directory will be the same on all hosts. IRAF logical directories are followed by a "$". In the last example, the task PATH is used to print the equivalent host pathname directly, including the node or machine name.

```
cl> show dev
iraf$dev/
cl> show iraf
/ursa/iraf/iraf/
cl> path dev$
ursa!/ursa/iraf/iraf/dev/
```

This use of IRAF environment variables as logical directories is discussed further in §3.4.

†Refer to the help page for RESET (see §3.3.3.) for details.

3.1.1. Defining your terminal type

Your terminal type will be set at login time by the execution of the "stty" statement in the *login.cl* file (see Figure 1). The terminal type was chosen by the user when MKIRAF was run initially (see §2.2.). At startup time, IRAF will print a message on the screen echoing the terminal type that it is setting.

```
% cl
    setting terminal type to gterm...
```

You can always check or reset your terminal type once IRAF is loaded.

```
cl> stty                        # check terminal type
cl> stty xtermjh nlines=24      # reset terminal type to xtermjh (24 lines)
```

The STTY command sets two environment variables, *terminal* (for text input and output) and *stdgraph* (for graphics output). If there is some question about what terminals and terminal emulators are supported by your IRAF distribution page through the *graphcap* file in the *dev* directory and look at the *STDGRAPH* entries.

```
cl> page dev$graphcap
```

Notice that in the default *login.cl* file (Figure 1) that some checking at the host level is done first to attempt a first guess at the correct terminal type. The user can do something similar in the *loginuser.cl* if he wants to modify this scheme.

3.1.2. Selecting hard copy devices

Two environment variables are used to define the names of printer and plot devices that IRAF will use for hard copies. These devices will vary from site to site.

```
cl> show printer                # show default printer
cl> show stdplot                # show default plotter
cl> reset printer=<lwa>         # change default printer for this session
cl> reset stdplot=<lwb>         # change default plot device for this session
```

Typing the command **devices** (if the appropriate file has been edited for your site by your IRAF site manager) will list your options for these variables.

There are three built-in options for *stdplot* that do not send the output to a hard copy device, but rather create Encapsulated PostScript files on disk with names similar to "sgi12345.eps". These "device" names are *eps* (produces a plot in the middle of the page), *epsl* (produces a landscaped plot), and *epsh* (produces a plot at the top of the page).

3.1.3. Choosing an editor

IRAF does not have its own editor but rather uses host level editors. You can set the IRAF environment variable *editor* to the host level editor that you want to use (it of course must exist) and then execute the IRAF task EDIT to invoke this editor. The editor that you choose must have a *.ed* entry in the IRAF *dev* directory to work properly with the tasks EDIT, EPARAM and EHISTORY. Currently there is support for *vi*, *edt*, *emacs*, and *memacs.*

```
cl> show editor                 # show default editor
cl> set editor=vi               # set the editor to vi
cl> edit <filename>             # invoke the editor
```

3.2. Packages and tasks

The programs or commands that the user executes to perform specific functions are called tasks. Tasks that perform similar functions are grouped together into packages. The packages for the IRAF core system are listed below.

```
     dataio - Data format conversion package (RFITS, etc.)
       dbms - Database management package (not yet implemented)
     images - General image processing package
   language - The command language itself
      lists - List processing package
      local - The template local package
   obsolete - Obsolete tasks
       plot - Plot package
      proto - Prototype or interim tasks
   softools - Software tools package
         tv - Image display load and control package
     system - System utilities package
  utilities - Miscellaneous utilities package
```

A package must be loaded in order to execute a task that is in it. A package is loaded by simply typing its name or enough characters to uniquely identify its name, i.e., **softools** or **so**. Note that IRAF is case sensitive! The prompt reflects the current package (the last package loaded). The last package that was loaded can be unloaded by typing **bye**. What packages are currently loaded can be checked by typing **package**. Once a package is loaded its tasks are available for execution until the package is unloaded. Loading a new package does not unload the previous package. There is essentially no overhead in having a variety of packages loaded at the same time.

```
cl> package        # check to see what packages are loaded
cl> softools       # load the SOFTOOLS package
so> package        # check to see what packages are loaded
so> bye            # unload the SOFTOOLS package
cl>                # prompt goes back to previous prompt
```

When you log into IRAF all the packages in the IRAF core system are automatically loaded for you (except SOFTOOLS), unless this has been changed by your site manager.

See Appendices A and B for a complete listing of the IRAF and NOAO packages and tasks.

3.3. General help facilities

There are several online help facilities within the IRAF system.

3.3.1. Simple tools

There are several simple help tools that the user may find useful.

```
cl> ?             # list the tasks in the current package
cl> ?dataio       # list the tasks in a specific package - the package must be loaded
cl> ??            # list all tasks currently loaded
cl> package       # list all packages currently loaded
```

3.3.2. Online manual pages

All tasks in the IRAF system have detailed online help or manual pages. All packages have online menu pages that consist of a list of the tasks in each package plus a one line description of each task. These manual pages and menus are part of an IRAF online help database that is available to the user at login time, i.e., the tasks and packages do not need to be loaded to access these online help facilities.

The online help database is dynamic and can include IRAF software packages other than the IRAF core system when these packages are installed as part of the main IRAF system. Even local software can be installed so that their help pages are part of this database.

In the next few sections on "help", the syntax used for the task executions in the examples may not be obvious to the new user. Task parameters and the syntax for command execution are covered in more detail in §3.5. and §3.6.

3.3.3. The HELP task

The HELP task will page through the online manual pages for a specific task or package menu. The user can move forward through this help file spacing by line (**CR**), half page (**d**), or full page (space bar). A **q** quits the help. A short summary help line is printed at the bottom of each page during the execution of HELP. Try, for example,

```
cl> help                         # help for the current package
cl> help help                    # help for the HELP task itself
cl> help rfits sec=examples      # help for the EXAMPLES section of the RFITS
                                   manual page
cl> help rfits opt=source        # show the source code for the task
cl> help dataio                  # help for a specific package
```

If you are uncertain about what package a task is in, use the HELP task. The first line of the help page gives the package name for a task as well as the date of the last modification of the help page. For example, the RFITS help page shows

```
RFITS (Jan90)                    dataio                    RFITS (Jan90)
```

There are other documents in IRAF that have the same format as the online manual pages (file names ending in *.hlp*) but are not part of the IRAF online help database. These files can also be accessed with the HELP task.

```
cl> cd mwcs                # change directories to the logical directory mwcs
cl> help MWCS.hlp f+       # read this file with HELP
cl> cd                     # change directories back to your IRAF home directory
```

3.3.4. The PHELP task

The PHELP task is similar to the HELP task except that it allows the user to page backwards and forwards through the online manual pages. It also has a wider range of options for paging that can be queried with **?** at any time during its execution.

```
cl> phelp phelp            # get help for PHELP task
    ?                      # print summary help line
    q                      # quit PHELP
cl> phelp combine all+     # get help on all tasks called "combine"
    N                      # go to help page for next "combine" task
    P                      # go to help page for previous "combine" task
    q                      # quite help
```

3.3.5. Printing online manual pages

Hard copies of the online manual pages or any *.hlp* file can be generated using the HELP and LPRINT tasks along with the pipe symbol (|). The output printer can be the default printer specified by the environment variable *printer* (see §3.1.2.) or it can be set explicitly.

```
cl> help help | lprint                  # send help file to default printer device
cl> help help | lprint dev=<lw>         # send help file to a specific printer
cl> help mwcs$MWCS.hlp f+ | lprint dev=<lw>
                                        # send a non-online help file to a specific
                                          printer
cl> help dataio.* | lprint              # print all online help for the DATAIO
                                          package
```

3.3.6. Finding a task with a particular function (REFERENCES)

The REFERENCES task can be used to locate a task that will provide a certain functionality. The task searches the online help database for a string supplied by the user. Only the "one-liner" information from the package menus is used in this search. Since this can be rather slow an option is available to create a quick-reference file to make future searches faster; this quick-reference file is stored in your *uparm* directory.

```
cl> references help          # list all tasks with "help" in their description
cl> refer upd+               # generate a quick-reference file
cl> refer smooth             # list all tasks with "smooth" in their
                               description using the quick-reference file
```

3.4. Doing things with directories

IRAF uses a *virtual* file system. What this means to the user is that directory and file names from within IRAF will look the same on any computer. This is particularly noticeable in VMS/IRAF - try typing **dir** and then **!dir**.†

There are many ways in IRAF to work with directories. Remember that your IRAF *home* directory is the host level directory from which you logged into IRAF. The IRAF *home* directory and the *uparm* directory are also both environment variables set to logical directories, at login time.

```
cl> show home          # show pathname for IRAF home directory
cl> show uparm         # show uparm pathname
```

3.4.1. Creating/removing directories

You can create a directory from within IRAF using the MKDIR task.‡ The directory path may be specified as either an IRAF pathname or a host level pathname. When creating subdirectories the upper level directories must already exist. Notice the use of the logical directory *home* in the second example along with the trailing *$*. The first two examples use IRAF pathnames while the last two examples use host level pathnames.

†The "!" is used as an escape to the host operating system and allows the user to execute a host level command from within IRAF.

‡Be forewarned about long pathnames in IRAF. Many IRAF tasks have an upper limit of 63 characters to file names, including the full host level pathname.

```
cl> diskspace                  # check available diskspace
cl> mkdir nite1                # create a subdirectory called nite1
cl> mkdir home$nite1           # create a subdirectory in your IRAF home directory
cl> mkdir </tmp8/irafdir>      # create a directory on a scratch disk (UNIX host)
cl> mkdir "<scr1:[irafdir]>"   # create a directory on a scratch disk (VMS host)
```

Currently there is no way to remove a directory from within IRAF; the user must do this at the host level. Again note the use of the escape character (**!**) in some of the examples.

- The following is an example of how to remove a subdirectory called *nite1*, assuming a UNIX host. Be sure to delete all IRAF images in the directory first with IMDELETE (see §3.8.5.).

```
cl> cd nite1          # go to nite1 subdirectory
cl> imdelete *.imh    # delete all images
cl> cd ..             # go up one level of directories
cl> !rm -r nite1      # delete directory nite1
```

- On a VMS host you must first remove all of the files in a directory, then change the protection on that directory, and then finally delete the directory itself.

```
cl> cd nite1                 # go to nite1 subdirectory
cl> imdelete *.imh           # delete all IRAF images
cl> delete *                 # delete all remaining files
cl> cd ..                    # go up one level of directories
cl> !set protection=(O:RWED) nite1.dir;1      # unprotect directory
cl> !delete nite1.dir;1      # remove directory
```

3.4.2. Changing directories

The user can move around the IRAF directory structure using IRAF directory names, logical directories, or host level pathnames. Some examples may demonstrate these various methods.

```
cl> path                       # type current directory
cl> cd                         # go to home directory
cl> mkdir nite1                # create subdirectory nite1
cl> cd nite1                   # change to the nite1 subdirectory
cl> cd ..                      # go up one directory level
cl> cd iraf$doc                # change to a subdirectory of the logical directory iraf
cl> back                       # go back to the previous directory

cl> set data=</tmp8/data/>     # define a logical directory (UNIX host)
cl> set data="<scr2:[data]>"   # define a logical directory (VMS host)
cl> cd data                    # change to the logical directory data
cl> cd </tmp8/data>/           # change to host directory (UNIX host)
cl> cd "<scr2:[data]>"         # change to host directory (VMS host)
```

3.5. Parameter files

IRAF tasks are the commands the user executes to perform certain operations. And the parameters for the tasks determine how those operations will be done. This section discusses the various ways to look at and to change task parameters.

The default parameter file for the RFITS task is listed below.

```
    fits_file = "mta"           FITS data source
    file_list =                 File list
    iraf_file = ""              IRAF filename
  (make_image = yes)            Create an IRAF image?
 (long_header = no)             Print FITS header cards?
(short_header = yes)            Print short header?
    (datatype = "")             IRAF data type
       (blank = 0.)             Blank value
       (scale = yes)            Scale the data?
 (oldirafname = no)             Use old IRAF name in place of iraf_file?
      (offset = 0)              Tape file offset
        (mode = "ql")
```

There are two types of parameters, required (often called query or positional) parameters and hidden parameters. In our example above the first three parameters are required parameters; these parameters must be specified each time the task is executed. The parameters in parentheses are called hidden parameters; if these parameters are not specified at execution time the current default parameter values will be used.

The user can customize the parameters for any task; these customized parameter files are then stored in the user's *uparm* directory. Whenever a task is executed IRAF first searches the user's *uparm* directory for a customized parameter file; if one does not exist the system default file is used.

A parameter file name in the *uparm* directory is a combination of the package and task name (the first two characters plus the last character of the package name followed by the first five characters plus the last character of the task name). Execute **`dir uparm`** to see what parameter files are in your *uparm* directory.

It is also possible for packages to have parameter files. In this case the values of the package parameters are used by all of the tasks in the package.

3.5.1. Listing parameters

Parameters are listed with the LPARAM task. The parent package for a task must be loaded before its parameter file can be listed. If a customized parameter file exists in the *uparm* directory then those parameter values will be listed and not the system defaults.

`cl>` **`lpar rfits`** # list the parameters for RFITS

3.5.2. Editing parameters

Parameters can be edited with the EPARAM task. EPARAM calls up an interactive screen editor controlled by the environment variable *editor* (see §3.1.3.). In its simplest mode the user simply moves the cursor to a parameter (using either the "Return" key for downward motion, the up/down arrow keys, or possibly *Ctrl-J* or *Ctrl-K*) and types in the new value for the parameter followed by a "Return". Upon exiting EPARAM with a **`:q`**, the edited parameter file will be saved in the *uparm* directory.† We say the parameters have been *learned*. If EPARAM is exited with a **`:q!`** then the parameter file is left unchanged.‡

`cl>` **`epar rfits`** # edit the parameters for RFITS

†The saved parameter file may be stored in memory for a bit before it actually gets written to the *uparm* directory, but for the most part this should be transparent to the user.

‡See IRAF Newsletter Number 7, June 1989, *Using and Customizing the EPARAM and EHISTORY Editors*, for some helpful hints.

"Control" keys can also be used to exit EPARAM. An EOF that is defined at the host level, e.g., *Ctrl-Z* or *Ctrl-D*, can be used to exit EPARAM and update the parameters, while *Ctrl-C* can be used to exit EPARAM with no updating.

Parameters can also be changed by specifying the task, the parameter, and its new value explicitly. Different syntaxes are used for the different types of parameter values, in particular, boolean values are given as "yes" or "no" and string values must by quoted. The last example shows a way to query for a particular parameter value.

```
cl> rfits.scale=no
cl> rfits.datatype="real"
cl> rfits.offset=100
cl> =rfits.datatype
```

3.5.3. Resetting task parameters

Sometimes it is useful to UNLEARN or go back to the system default parameter values for a task. It is also possible to UNLEARN all of the tasks for a particular package including the parameter file for the package itself, if one exists. Note that what UNLEARN really does is delete the appropriate parameter files from the *uparm* directory!

```
cl> unlearn rfits       # go back to system default parameters for RFITS
cl> unlearn dataio      # go back to default parameters for all tasks in the
                          DATAIO package, including any package parameters
```

3.6. Executing tasks

3.6.1. General syntax

Many task executions have been used in the previous sections as examples to the discussions at hand. It may not have always been clear to the user why a certain syntax was being used. We will try to outline some simple rules for task execution in this section.

It has probably already been noted by the reader that it is not always necessary to type the full name of a task or its parameters when executing that task. It is only necessary to type just enough characters so the task and parameters are uniquely defined.

The parameters are separated on the command line by spaces, so there can not be any spaces within the parameter specifications unless they are enclosed in double quotes, i.e., "".

Let us look at the parameters for the task IMSTATISTICS to use as a reference point for our current discussion.

```
   images =                   Images
  (fields = "image,npix,mean,stddev,min,max") Fields to be printed
   (lower = INDEF)            Lower cutoff for pixel values
   (upper = INDEF)            Upper cutoff for pixel values
(binwidth = 0.1)              Bin width of histogram in sigma
  (format = yes)              Format output and print column labels?
    (mode = "ql")
```

A simple way to execute IMSTATISTICS would be to use the "query" feature for the required parameters (in this case the single parameter at the top of the list that is not enclosed in parentheses). In this example the user simply types the task name, is "queried" for the required parameter, and answers accordingly. The current default values for the "hidden" parameters are used.

```
cl> imstat
Images: dev$pix
#   IMAGE      NPIX      MEAN    STDDEV       MIN       MAX
  dev$pix    262144     108.3     131.3       -1.    19936.
```

The user can use the "query" feature and also change "hidden" parameters on the command line at execution time. In this example the *format* parameter is switched to no - note the syntax! The "=yes" and "=no" part of boolean parameters can be specified with a "+" or a "-".

```
cl> imstat form-
Images: dev$pix
dev$pix  262144  108.3154  131.298  -1.  19936.
```

A task can also be executed putting all the parameters on the command line. Note that this syntax uses the "positional" feature of the required parameters in that these parameters must appear on the command line in the order that they appear in the parameter list. It is not necessary to specify the parameter name but only its value. Hidden parameters can also be specified on the command line but since they include the name of the parameter as well as its value they can appear in any order after the required parameters. Hidden parameters specified on the command line are **not** learned; the command line values simply override the defaults for that execution of the task.

```
cl> imstat dev$pix fields=mean,stddev
```

In this next example we extend the long command line to a second line with the **** character. The break can come at the end of a parameter string, as in the example, or after a comma in the parameter string itself (in this case no preceding space is allowed).

```
cl> imstat dev$pix fields=mean,stddev,min,max \
>>> format-
```

Of course, one can always run EPARAM on a task, modify the hidden parameters, and then execute the task while not specifying any hidden parameters on the command line.

```
cl> epar imstat
cl> imstat dev$pix
```

Another way to execute a task would be to run EPARAM on a task, modify all of the parameters including the required parameters, and then execute the task from within EPARAM with the **:go** command.

So there are many ways to execute tasks. Experience has shown that each user has his own favorite style. Many new users to IRAF find that the EPARAM technique mentioned just above is the most straightforward method for them.

3.6.2. Running tasks in the background

An IRAF task can be run in the background by ending the task execution command line with an **&**. The following IMSTATISTICS command will run as a background job. Since IMSTATISTICS prints information to the terminal during its execution, by default, the terminal output is redirected to a new file with the **>** sign (see §8.1. for more information on "redirection".

```
cl> imstat dev$pix form- > statfile &
```

If you do not specify all of the required parameters on the command line at execution time the CL will stop and wait for you to respond, as in the following sequence (the ">>" symbols are used to append to the *statfile* created earlier).

```
cl> imstat >> statfile &
[1] stopped waiting for parameter input
cl> service 1
Images: dev$pix
```

WARNING: Usually when this happens it is because the user *accidentally* omitted a required parameter so be sure you are being queried for the right parameter - remember position is important! It may be best to kill the job and start over.

The task JOBS will list the jobs that are in the background queue and give their current status. A background job can be killed with the command KILL plus the job number from the background queue.

```
cl> jobs
cl> kill 1
```

Background jobs submitted to UNIX hosts will continue to run even after the user has logged out of IRAF and off the machine. VMS users must submit jobs to a "batch" queue, as mentioned in the next paragraph, if they plan to log off the computer before the job has completed properly.

Background jobs can also be submitted to batch queues, if they are supported by the operating system. VMS is one such system. `$ IRAF` must be in the user's *login.com* file on VMS hosts in order to submit jobs to the batch queues. IRAF supports three logical batch queues - fast, batch and slow. See your local systems manager for more on your local batch queues. In the following example the task IMCOMBINE is executed with its terminal output being redirected to the file *logfile*, and the job is submitted to the VMS "batch" queue.

```
cl> imcombine <filenames> check expos+ > logfile & batch
```

3.6.3. Aborting tasks

Any interactive task can be aborted with *Ctrl-C* (the Control and the C key held down simultaneously). Although every effort is made to clean up properly after an abort sometimes things can be left in a weird state. It is generally good practice to type FLPRCACHE (fondly known as "flipper") after an abort. If problems occur after the initial FLPR, try one more FLPR. If problems still occur then it is probably necessary, or at least easiest, to log out of IRAF and then back in.

Generally speaking, each IRAF package is contained in a single executable file, not each separate IRAF task. When a task is executed its package executable is put into the user's "process cache". Each user can look at his own process cache with PRCACHE. When the user aborts a task it is possible for this executable to become "corrupted" and thus run incorrectly when another task from this same executable is executed. FLPR discards all executables from the process cache except the SYSTEM executable which is "locked" in at startup time. Fresh executables are then retrieved from the system for the next task executions.

It is particularly dangerous to abort tape tasks. Just be forewarned and do not be surprised if some rather strange error messages appear on your terminal screen. Try FLPR; if all else fails, logout and back into IRAF.

Backgrounds tasks should be aborted with the KILL command as mentioned in the previous section.

3.7. Data files

Since IRAF uses a *virtual* file system all file names in IRAF will appear the same regardless of the host computer, as mentioned in §3.4. IRAF does not support multiple versions of files like those on a VMS host (only the highest version numbers are used).

Legal file names can contain the character set [A-Za-z0-9_.].† It may be necessary to quote unusual instances of file names, such as "1245b" which will be interpreted incorrectly as an octal number unless it is quoted, or "03457" where the leading zero will be dropped unless the name is quoted.

When we talk about IRAF data files it is useful to think in terms of three different types of files: text (ASCII), simple binary, and image files. The discussion of IRAF image files is left to the next section since these are files that require special attention.

There are many tasks in the SYSTEMS package for working with text and simple binary files. When working with image files it is better to use the tasks in the IMAGES package.

The DIRECTORY task can be used to list files in the current or specified directory. Wildcards can be used to search for a particular pattern. See §8.3. for more on wildcards and file templates.

```
cl> dir                  # list current directory
cl> dir uparm            # list files in "uparm" directory
cl> dir *cl*             # list files with "cl" in the name
cl> dir login.??         # list files with "login." and any two characters at the end
cl> dir *.imh            # list all IRAF image files in this directory
```

Text files are used in many different ways in IRAF. They can be used as input to some tasks and can be produced as output from other tasks, often as database or log files. Terminal output from a task execution can be "redirected" to a text file (see §8.1. for more on redirection). Text files can be edited using the EDIT command in IRAF. Simple operations of text files (and binary files, where appropriate) are done by many tasks in the SYSTEMS package.

```
concatenate - Concatenate a list of files
       copy - Copy a file or files (use IMCOPY for imagefiles)
      count - Count the number of lines, words, characters in a text file
     delete - Delete a file or files (use IMDELETE to delete imagefiles)
       head - Print the first few lines of a text file
     lprint - Print a file on the line printer device
      match - Print all lines in a file that match a pattern
       page - Page through a file
     rename - Rename a file
       sort - Sort a text file
       tail - Print the last few lines of a file
       type - Type a text file on the standard output
```

Simple operations can be performed on lists of data by tasks in the LISTS package and the FIELDS and JOINLINES tasks in the PROTO package. Other packages like the NOAO packages have specific tasks for handling large text files that are produced during data processing as database files.

3.8. IRAF image files

IRAF supports several image formats. Only the original IRAF image format, called the OIF format, will be discussed in this section. See Appendix C for a discussion of the other image formats that are available.

The OIF format is the default image format (unless it has been changed by your site manager at installation time). Your default image format can be checked and reset, as in the

†There is a bug in VMS/IRAF that can cause problems with file names containing capital letters. VMS users should avoid file names with capital letters until this bug is fixed.

following examples, and depends on the value of the environment variable *imtype*.

```
cl> show imtype                  # check image format
cl> set imtype=imh               # set image format to OIF
```

Generally speaking the *imtype* variable only affects new images, i.e., those read in with RFITS or some other reader. The output images from tasks will usually inherit the *imtype* of the input images unless explicitly set, although this may vary depending on the image format. (Again see Appendix C for further discussion about other types of image formats.)

The OIF image format consist of two files: a header file (ending in *.imh*) and a pixel file (ending in *.pix*). On UNIX hosts there is also a ".." file associated with each *.imh* file that is used to protect the image from deletion accidentally with the DELETE task (you must execute **dir a+** to see these files).†

3.8.1. The IRAF image header

The IRAF image header file is a binary file and has the extension *.imh*. A special task IMHEADER is used to look at the header information and associated keywords. The header looks very much like a FITS header but it is actually quite different.

```
cl> imheader dev$pix               # list the short version of the image header
cl> imheader dev$pix l+ | page     # list the long version of the image header
```

The image header has a finite length determined by the environment variable *min_lenuserarea*. There is currently room for about 250 80-character records or lines (20000 characters).

```
cl> show min_lenuserarea           # show current value
cl> set min_lenuserarea = 40000    # set to larger value
```

The task HEDIT can be used to edit the header information, adding new keywords and values or modifying existing ones. The task HSELECT can be used to list selected keyword values from the image header. Try the following (the *$I* in the HSELECT command assures that the image file name is listed as well).

```
cl> imcopy dev$pix newpix                  # make copy of dev$pix
cl> hedit newpix newkey "string" add+      # add new keyword newkey
cl> hselect newpix $I,newkey yes           # list newkey value
cl> imdelete newpix                        # delete copy of image
```

3.8.2. The IRAF pixel file

The IRAF pixel file is also a binary file and has the extension *.pix*. It does not necessarily have to be stored in the same directory as the header file. When IRAF images are created the environment variable *imdir* determines where the pixel (*.pix*) files will be stored. The pathname for the *.pix* file is then written into the header (*.imh*) file so that any task operating on images will be able to find the pixel files, no matter where they are stored. The *.imh* files go into the current directory by default. In the examples below, it is important to realize that *set imdir* does not create a directory; the user will need to create any directories with the MKDIR command (except for *HDR$pixels/* - this will get created automatically by IRAF).

In the examples below the trailing slash is important!

†See IRAF Newsletter Number 6, February 1989, *UNIX/IRAF Image File Protection (the `..´) File*

```
cl> show imdir                       # show the current pixel directory
cl> dir imdir                        # list the files in this directory
cl> set imdir=HDR$                   # set the pixel directory to the current directory
cl> set imdir=HDR$pixels/            # set the pixel directory to a subdirectory of the
                                       current directory
cl> set imdir=</tmp8/irafdir/>       # set the pixel directory to a scratch area
                                       (UNIX host)
cl> set imdir="<scr2:[irafdir]>"     # set the pixel directory to a scratch area
                                       (VMS host)
```

The *HDR$* and *HDR$pixels/* logical directories are host independent directory pathnames and will appear as such in the image header (*.imh* file). The other examples are host dependent pathnames. Note the following headers produced by executing

```
cl> imhead <imagename>  l+ u-
```

- In this example *imdir* has been set to *HDR$pixels/.* The "[NO PIXEL FILE]" comment after the pixel pathname is an IRAF "feature" - if you see the *.pix* file in the *pixels* directory do not be fooled by this statement.

```
newpix[512,512][short]: m51  B  600s
    No bad pixels, no histogram, min=unknown, max=unknown
    Line storage mode, physdim [512,512], length of user area 1459 s.u.
    Created Tue 14:07:03 25-Sep-90, Last modified Tue 14:07:04 25-Sep-90
    Pixel file 'HDR$pixels/newpix.pix' [NO PIXEL FILE]
```

- In this example *imdir* has been set to a host dependent directory. If the "[NO PIXEL FILE]" statement appears with the host dependent directory name this is an indication that IRAF cannot truly find the *.pix* file. Note in this example that the node name of the host machine is included with the pixel pathname, *tucana!*<pathname>.

```
newpix[512,512][short]: m51  B  600s
    No bad pixels, no histogram, min=unknown, max=unknown
    Line storage mode, physdim [512,512], length of user area 1459 s.u.
    Created Tue 14:07:03 25-Sep-90, Last modified Tue 14:07:04 25-Sep-90
    Pixel file 'tucana!/tmp3/jbarnes/newpix.pix' [ok]
```

3.8.3. Pixel data types

IRAF supports several pixel data types, among these are: short (16-bit signed integers), long (32-bit signed integers), real (32-bit floating point), and double (64-bit floating point). Limited support exists for complex numbers. The task CHPIXTYPE can be used to change the pixel type of an image. The pixel data type can be determined with the IMHEADER task.

```
cl> imhead dev$pix
dev$pix[512,512][short]: m51  B  600s
```

Generally speaking the IRAF core tasks produce the same pixel type on output as that of the input image. The internal calculations may be done in reals or double if complicated computations are performed on the input data, otherwise the calculation type may also be done in the same data type as the input image. Some tasks have parameters that allow the user to specify the calculation and output pixel types. Let this be a caution to the user who has integer data: it may be advisable to convert integer data to reals before doing any image processing if disk space is not a concern, otherwise be alert to the possibility of truncation or wraparound (large negative numbers suddenly appearing in the data).

3.8.4. Image dimensions

IRAF supports multi-dimensional images. The size and the dimension of an image can be determined with the IMHEADER task. Using *dev$pix* as input, IMHEADER reports that this image is 2-d with 512 pixels in each dimension.

```
cl> imhead dev$pix
dev$pix[512,512][short]: m51  B  600s
```

Although a 1-d image can be represented by a 2-d image with one row/line or column, it can also be represented by a vector with just one dimension. This has been a popular format for 1-d data in the NOAO spectroscopic packages.

```
cl> imhead oned
oned[820][real]: FEIGE 98
```

The task IMSTACK can be used to stack images into a higher dimension image (inheriting the header of the first image in the list), and IMSLICE can be used to produce images of one less dimension than that of the input image.

An "image section", discussed in §8.2., can be used along with the task IMCOPY to reduce the dimensionality of an image. For example, the output of this task execution will be a 1-d vector image.

```
cl> imhead dev$pix
dev$pix[512,512][short]: m51  B  600s
cl> imcopy dev$pix[*,20] oned
cl> imhead oned
oned[512][short]: m51  B  600s
```

3.8.5. General image tools

A few general image tools will be highlighted in this section. It is important to use these tools rather than the similar tools for text and simple binary files provided in the SYSTEMS package since the image tools must deal with a more complicated file structure (two files† rather than one). Three important tasks for image handling are IMCOPY, INRENAME, and IMDELETE. IMCOPY makes a new copy of the input image. IMRENAME will rename the current image. And IMDELETE will delete images. Using the distributed image of M51, try the following,

```
cl> imcopy dev$pix newpix          # make another copy of an image
cl> imrename newpix newcop         # rename an image
cl> imdelete newcop                # delete an image
```

The IMRENAME task has another useful feature besides simply renaming the image. It also moves pixel files to the current directory specified by the environment variable *imdir*. Thus whole directories of pixel files can be moved using this task, and the image header will be updated properly to point to this new directory. (Problems can arise with disconnected pixel files if these files are moved with host level facilities. See Appendix D.) It is not necessary to actually rename or move the headers to do this.

```
cl> set imdir=</tmp3/irafdir/>     # set imdir
cl> imrename *.imh .               # move pixel files to new imdir
```

The pixel values can be listed with the task LISTPIXELS. Note the use of "image sections" in these examples. See §8.2. for more details. Again, we use M51 as our input image.

†UNIX hosts really have three files associated with each IRAF image since there is also the ".." protection file.

In the second example the output of LISTPIXELS is "piped" through to the input of the TABLE task (in the LISTS package) for reformatting before being displayed on the terminal.

```
cl> listpix dev$pix[20:25,30:32] # list the pixel values for an image section
cl> listpix dev$pix[20:25,30:32] | table     # list and reformat
```

There are other useful tools for dealing with images in the IMAGES package as well as in the PROTO package. See Appendix A for a complete listing of the tasks in these packages for the V2.10 release.

4. Reading and writing data

The IRAF utilities for reading and writing data are in the DATAIO package with some general tools available in the SYSTEMS, SOFTOOLS, and PROTO packages. Other packages outside the IRAF core system may have their own local utilities for this purpose as well, such as the MTLOCAL package in the NOAO science software, the FITSIO package in the Space Telescope Science Institute STSDAS package, or the XDATAIO package in the PROS XRAY package from the Smithsonian Astrophysical Observatory.

4.1. General tape handling facilities

IRAF supports a wide range of tape devices including 9-track tape units, QIC drives (cartridges), Exabytes, and Digital Audio Tapes (DATs). The tape devices in use at your site by IRAF are described in the *dev$tapecap* file.† Each device that is available to you from within IRAF must have a workable entry in this file. This file should be setup by your local IRAF system manager; in many cases it will only require a few modest modifications to a few existing entries in the distributed file in order to have it work with your devices. At the same time, your IRAF site manager should have modified the *dev$devices.hlp* file so that typing **devices** will show the available local tape devices.

The general tools for tape handling are best demonstrated by examples. Tape drives are accessed by the logical names defined in the *tapecap* file and not by their host names. IRAF usually knows tape devices as *mta*, *mtb*, and so on. The leading "mt" is important since it implies a tape device as the input/output device and not a file name on disk.

```
cl> devices              # list devices at your site
cl> allocate mta         # allocate tape drive mta
cl> rewind mta           # rewind tape drive mta
cl> devstatus mta        # print tape status for mta
cl> deallocate mta       # deallocate tape drive mta
```

The task DEVSTATUS will print the status of a tape drive, i.e., if it is available, if it is allocated to another user, or if allocated to you what the current tape position is. This information is obtained from a *mt*xxx*.lok* file that is created by the ALLOCATE command in the IRAF logical directory *tmp* on the parent machine of the device.‡ The *.lok* file is updated after the completion of each IRAF tape task. IRAF must know the correct position of the tape in order for subsequent tape handling facilities to work correctly. When in doubt, REWIND the device first. Be aware that a rewind at the host level is not communicated to IRAF so it is possible to have the actual tape position out of sync with the information in the *.lok* file. Again, a REWIND at the IRAF level should put things back in sync.

It is not a good idea to abort tape tasks unless they have been submitted as background jobs (see §3.6.2.). A Ctrl-C can leave things in a confused state and you may need to do a FLPR; if this fails you may need to log out and back into IRAF (see §3.6.3). It would be best

†A summary of the parameters used in the *tapecap* file can be obtained by typing **page os$zfiomt.c**.
‡This statement is true for IRAF Version 2.10.2 and later; earlier releases of IRAF Version 2.10 stored this file in the local IRAF *tmp* directory on the user's machine.

to also REWIND the tape before executing any additional tape tasks.

4.2. Setting tape densities

Several tape devices support multiple tape densities, including 9-track tapes and Exabytes. It is not necessary to specify the tape density when reading a tape with these devices with any of the IRAF tape readers. It is important, however, to be aware of how to set the tape density properly when writing data to this type of tape unit.

Writing data to a new tape at a specific density is host dependent. On most UNIX hosts this is handled from within IRAF by appending the tape density to the tape drive name at the time the write command is executed, assuming that the tape density can be set remotely and is not hardwired by a switch on the front of the drive. If no density is specified at the time of the write command the default density for that drive will be used.† It is only necessary to be concerned about this when writing to a new tape; when appending data to a tape the tape density is sensed and retained.

```
cl> allocate mta                       # allocate tape drive mta
cl> wfits mta6250 <image1> new+        # write one image to tape at 6250bpi
cl> wfits mta <image2> new-            # append second image to tape
```

On a VMS host the density for a new tape must first be set at the host level by writing a label on the tape before the tape is allocated in IRAF. The label must then be overwritten by the IRAF tape writing task since labeled tapes are not supported by IRAF. Again, this should only be done for a *new* tape; if there is already data on the tape then subsequent data will be written at that same density.

```
cl> !init/den=6250 mua1: label         # initialize new tape to 6250bpi
cl> allocate mtb                       # allocate tape drive in IRAF
cl> wfits mtb <images> new+            # write images to tape
```

4.3. FITS facilities

The FITS facilities in IRAF support the general FITS format as described in the paper, *FITS: A Flexible Image Transport System*, Wells, D. C., Greisen, E. W., and Harten, R. H., 1981, *Astronomy and Astrophysics Supplement Series* 44, 363. Blocking factors of up to ten (10) times the standard 2880 bytes and the IEEE floating point format are also supported. There is no FITS support for the group format or for FITS tables in the IRAF core system.

4.3.1. Reading FITS images

The IRAF task used to read FITS data is the task RFITS in the DATAIO package. The default parameters for this task are listed below.

†The default density is set by a parameter in the *dev$tapecap* file and is site configurable.

```
    fits_file = "mta"           FITS data source
    file_list =                 File list
    iraf_file = ""              IRAF filename
  (make_image = yes)            Create an IRAF image?
 (long_header = no)             Print FITS header cards?
(short_header = yes)            Print short header?
    (datatype = "")             IRAF data type
       (blank = 0.)             Blank value
       (scale = yes)            Scale the data?
 (oldirafname = no)             Use old IRAF name in place of iraf_file?
      (offset = 0)              Tape file offset
        (mode = "ql")
```

The following example shows a series of tasks that a user might execute to read all the data from a FITS tape onto disk (assuming there are less than 999 files on the tape!). In this example the output image names will be "n10001, n10002, n10003....". The output pixel type (integer, reals, etc.) will depend on the type of input data. Since this execution of RFITS can take awhile we have submitted the task as a background job and redirected the terminal output (short header listings) as well as any error messages to a file (see §8.1.) that can be later reviewed or printed.

```
cl> show imdir                    # check location of pixel file directory
cl> diskspace                     # check to be sure there is enough space for pictures
cl> allocate mta                  # allocate tape drive mta
da> rfits mta 1-999 n1 >& tlist &     # read in all FITS images on the tape
da> page tlist                    # page through tlist
da> lprint tlist                  # print tlist to the default printer
```

A complete listing of a FITS header of one of the FITS files on a tape can be generated as in the following example. In this example we are going to list all the header information for the 35th FITS file on the tape and "pipe" this output to the PAGE task for easy reading. Note the use of *make_image=no* or *make-* to simply list the header but not write data to disk. The full listing of the FITS header will be slightly different than that produced from IMHEADER for the same image if it had been read onto disk since some descriptive keywords defining the image format are not included explicitly in the IRAF image header.

```
cl> rfits mta 35 make- l+ | page
```

The output image type can be set explicitly when FITS files are read regardless of the input format. Many users find it advantageous to read all data into IRAF as "reals" to avoid any misunderstandings with the datatype later during processing. In the following example the output datatype is set explicitly to "reals" and only selected files are read in. Since short header information is printed to the terminal by default, this output is redirected to a file to keep things from getting messy on the terminal screen. In this example we do not redirect any error messages so any messages of this type will be printed on the terminal screen.

```
cl> rfits mta 1-10,30-56 dt data=r > dtlist &
```

The IRAF WFITS command writes the old disk image file name into the FITS header using the keyword IRAFNAME. This keyword can be read by RFITS by setting *oldirafname=yes*. In the following example we want to restore to disk only certain images from a previous session that have been saved on tape. Keep in mind that FITS files are read from tape by file position and not by image name. So it is necessary to first list the tape along with the old IRAF image name, then select the files we want to read, and then execute RFITS.

```
cl> rfits mta 1-999 make- old+ | page   # list the tape by the IRAFNAME
                                           image name
cl> rfits mta 20-25,45-47 junk old+     # read in images resetting the output
                                           image name to IRAFNAME
```

The user is not limited to reading FITS images from tape. RFITS will also read FITS images that reside on disk.† In the following example we have a set of FITS images on disk called "fits001, fits002, fits003..." that we wish to convert to IRAF images.

```
cl> rfits fits* 1 m31
```

If the FITS headers are unusually long it is possible that some of the keyword information in the FITS headers (records) will be truncated and not be read into the IRAF image header. An error will be printed: "Warning: User area too small 12 card images discarded", as an example. It will be necessary to increase the image header size by setting the environment variable *min_lenuserarea* to a larger number than the current default, and then read your images in again. The current default for this variable is 20000 (equal to about 250 records).

```
cl> set min_lenuserarea=40000
```

4.3.2. Writing FITS images

IRAF images can be written to FITS images using the task WFITS in the DATAIO package.

The user should be cautioned that although a common way to write data to a FITS tape is by use of the "scaling" feature this can lead to lost precision in the data if the scaling process is biased by odd data values. In the most severe cases all zero's can be written in the FITS file! Remember to check your tape by reading data back in as IRAF images and comparing these to the original image data on disk. If you have done some weird massaging to your data extra caution may be in order. The IEEE option is the more desirable option for writing FITS data since no scaling is done and the actual pixel values are written to the FITS file. But the user must be confident that he can then read the IEEE format with his destination FITS reader. The IEEE format can also require more tape since the choices are 32 and 64 bits. Note the IEEE format is only supported by IRAF versions 2.9 and later. The IEEE format is now the default output format for IRAF images with real or double pixel types.

The default parameters for WFITS are listed below.

```
   iraf_files =                 IRAF images
   fits_files = ""              FITS filename
      newtape =                 Blank tape?
       bscale = 1.              FITS bscale
        bzero = 0.              FITS bzero
  (make_image = yes)            Create a FITS image?
 (long_header = no)             Print FITS header cards?
(short_header = yes)            Print short header?
      (bitpix = 0)              FITS bits per pixel
(blocking_fac = 0)              FITS tape blocking factor
       (scale = yes)            Scale data?
   (autoscale = yes)            Auto_scaling?
```

Some common ways to use WFITS will be explored in the next few paragraphs.

†VMS users should be aware that RFITS requires that disk files have fixed length records.

After completing data reductions a user may have several sets of images on disk that need to be written to a new Exabyte in IEEE floating point FITS format. As an example let us call the images "bred0001, bred0002..." and "rred0001, rred0002...". We will assume these images have pixel type real. We will create a text file containing a list of these images as input to WFITS (see §8.4.). This text file could be edited if necessary to add or delete images from the list. The short header information output to the terminal by WFITS is redirected to a file for printing later as a log of the tape. Since we are using the default parameters the FITS files will be written in IEEE floating point format.

```
cl> files rred*.imh > inlist        # create a list of the input images
cl> files bbred*.imh >> inlist      # append to the same list
cl> allocate mtd                    # allocate tape drive mtd
cl> unlearn wfits                   # use default parameter values for WFITS
cl> wfits @inlist mtd new+ > tlist &   # write data to tape
```

Data can be added to a tape by setting the parameter *new=no* or *new-*. The WFITS task will start writing at the double EOF written on the tape by the last successful write, i.e., if WFITS aborts for some reason you may not have a double EOF on the tape! In this next example let us set *bitpix=32* to force autoscaling of each image and set the *blocking_factor=1*†. The terminal output from WFITS is appended to an already existing file. (See §8.3. for more information on image templates.)

```
cl> wfits xred1??.imh mta new- bitpix=32 block=1 >> tlist &
```

In the example above we have changed the default values for two special parameters, *bitpix* and *blocking_factor*. Thus WFITS issues a "warning" at the beginning of its execution to that effect. This message is only meant to notify the user that he has changed some important defaults and does not necessarily indicate a problem.

```
WARNING: Default bitpix overridden.
        Bitpix set to: 32
WARNING: FITS tape blocking factor is 1
```

WFITS allows the user to overwrite data starting at a particular file number on the tape. In our example we will start writing at file 50. This syntax is invalid for Exabytes since these devices do not allow overwriting in this manner. The IRAF test image, *dev$pix*, has a pixel type integer so the FITS file will be written as 16 bit integer data, the default for this type of data unless specified otherwise.

```
cl> wfits dev$pix mta[50]
```

The user is not limited to writing FITS images to a tape device. FITS images can also be written to disk.‡ In the following example the user writes the images *m31red* and *m31blue* to FITS files on disk using the default IEEE floating point format. The disk file names in this case will be *fits001* and *fits002*. NOTE: disk names for files cannot start with the letters "mt", otherwise IRAF will think the task is writing to a tape drive!

```
cl> wfits m31red,m31blue fits
```

4.4. Text files

In §3.7., we discussed briefly some general tools for working with text files. The DATAIO package offers tasks for reading and writing text files as well as a task that converts IRAF images to text files and a task that converts text files to IRAF images. These tasks are

†The tape blocking factor default is set in the *dev$tapecap* file for each device.

‡VMS users should note that IRAF FITS images are written to disk as fixed length 512 byte records.

straightforward to use and the user is encouraged to read the online manual pages for further information.

```
rcardimage - Convert a cardimage file into a text file
rtextimage - Convert text files to IRAF images
wcardimage - Convert text files to cardimage files
wtextimage - Convert an IRAF image to a text file
```

4.5. Non-standard data formats

Frequently users have data, not in FITS format or text format, that need to be converted into IRAF images. The task IRAFIL in the PROTO package can be used to convert some forms of binary data into IRAF images. The task handles signed or unsigned 8- or 16-bit pixels. Any header information in the input file will be lost. The data must have been previously read onto disk.

IRAFIL combined with MTEXAMINE and REBLOCK in the DATAIO package can be used to convert binary data from tape to IRAF images. MTEXAMINE can be used to list the record structure of the tape files. REBLOCK can be used to read the tape files onto disk, reblocking or byte swapping in the process, if necessary. IRAFIL can then be used to convert these binary files on disk to IRAF image files.

The user can also write his own conversion program using IMFORT (see §9.2.) or model an SPP tape reading program after one of the existing readers in IRAF.

4.6. Tar file format

The SOFTOOLS package has tasks for reading and writing "tar" file format files on disk or tape†. You must use the host level tape device name with these tasks (RTAR and WTAR) and not the IRAF tape designation such as "mta".

```
rtar - Read a TAR format archive file
wtar - Write a TAR format archive file
```

Three uses of the tar file format are noted here. Most of the IRAF external software packages are distributed as source-only tar files allowing these distributions to be machine independent, if they are read in by IRAF. Database (ASCII) files produced by IRAF can be saved in a tar file, even if the user is using a VMS/IRAF system; thus this *backup* is portable from one IRAF host to another. IRAF images can also be saved in a tar file AFTER they have first been converted to FITS file format on disk; these FITS files can then be tar'ed and moved from one machine to another (tar'ing IRAF images directly is not recommended or advised).

4.7. Miscellaneous I/O tasks

A few remaining DATAIO tasks should be mentioned.

The MTEXAMINE task can be used to examine tape and disk files. It is also possible to dump records in various formats.

The T2D task can be used to quickly copy tape files to disk files. This is particularly useful for problem tapes. It is sometimes easier to extract useful data from a disk file if the tape file has been damaged.

†This statement is true for IRAF Version 2.10.2 and later; earlier releases of IRAF Version 2.10 did not support tar "tapes" but only disk files.

The REBLOCK task serves many purposes. It can be used to byte swap data as it is read onto disk as well as reblock data as the name infers. This task can also be used for tape to tape copies, changing densities if desired.

5. Plotting data

The general IRAF core system plotting tasks are in the PLOT package, although graphics functionality is not limited to these tasks.

The STTY task must have been issued properly either at login time or later during the IRAF session so that the graphics terminal type is known correctly by IRAF (see §3.1.1.).

The current plot resides in a frame buffer. The upper limit of the size of this frame buffer is determined by the environment variable *cmbuflen.* Complex plots may appear with missing pieces; increasing the size of the frame buffer should cure this problem. The GFLUSH command may need to be executed for this new value to be used.

```
cl> show cmbuflen
cl> reset cmbuflen = 256000
cl> gflush
```

5.1. Interactive graphics cursor mode

Many tasks in IRAF produce a plot and then bring up a graphics cursor leaving the task in interactive graphics mode, allowing the user to inspect or manipulate the data in some fashion using predefined keystrokes. Other tasks simply produce a plot and then exit. In the latter case interactive graphics mode can be invoked, using the current plot in the frame buffer, with **=gcur**.

The graphics interface provides a basic set of cursor keystrokes that are accessible to all plotting tasks in IRAF. The keystrokes are either single upper-case letters or ":." (colon-dot) commands. The single lower-case letters and ":" (colon) commands are reserved for the applications tasks, and the definitions for these keystrokes can vary from task to task.

The *global* cursor keystrokes (single upper-case or ":." commands) can be listed in interactive cursor mode by typing **:.help**. The cursor keystrokes for the individual interactive tasks can be listed in interactive cursor mode by typing **?**.

The globally accessible cursor keystrokes are listed in Figure 3. The online manual pages for CURSORS contains detailed information about these cursor options and the graphics system.

```
cl> phelp cursors
```

As an exercise the user may want to execute the following and become familiar with these keystrokes and ":." commands.

```
cl> stty                    # check terminal type
cl> stty xtermjh            # set terminal type where "xtermjh" is an example
cl> contour dev$pix         # generate a plot
cl> =gcur                   # invoke interactive graphics mode
    :.help                  # generate listing of cursor options
    ...                     # experiment with global cursor keystrokes
    0                       # get full plot back if needed
    :.snap <plot device>    # send plot to specific hard copy device
    =                       # send plot to default plotting device as defined
                              by stdplot
    q                       # exit with any lower-case keystroke
```

Plots sent to a hard copy device can be buffered for efficiency thus they may not appear in hard copy form until they are explicitly forced out. This can be done in interactive graphics

Cursor Mode Keystrokes (* = not yet implemented):

A	draw and label the axes of current viewport
B	backup over last instruction in frame buffer
C	print the cursor position as it moves
D	draw a line by marking the endpoints
E	expand plot by setting window corners
F	set fast cursor (for HJKL)
H	step cursor left
J	step cursor down
K	step cursor up
L	step cursor right
M	move point under cursor to center of screen
P	zoom out (restore previous expansion)
R	redraw the screen
T	draw a text string
U	undo last frame buffer edit
V	set slow cursor (for HJKL)
W	select WCS at current position of cursor
X	zoom in, X only
Y	zoom in, Y only
Z	zoom in, both X and Y
<	set lower limit of plot to the cursor y value
>	set upper limit of plot to the cursor y value
	escape next character
:	set cursor mode options
:!	send a command to the host system
=	shorthand for :.snap (make graphics hardcopy)
0	reset and redraw
1-9	roam

Cursor Mode Commands:

:.axes[+-]	draw axes of viewport whenever screen is redrawn
:.case[+-]	enable case sensitivity for keystrokes
:.clear	clear alpha memory (e.g, this text)
:.cursor n	select cursor (0=normal,1=crosshair,2=lightpen)
:.gflush	flush plotter output
:.help	print help text for cursor mode
:.init	initialize the graphics system
:.markcur[+-]	mark cursor position after each cursor read
:.off [keys]	disable selected cursor mode keys
:.on [keys]	enable selected cursor mode keys
:.page[+-]	enable screen clear before printing help text
:.read file	fill frame buffer from a file
:.show	print cursor mode and graphics kernel status
:.snap [device]	make hardcopy of graphics display
:.txqual qual	set character generator quality (normal,l,m,h)
:.txset format	set text drawing parameters (size,up,hj,vj,etc)
:.xres=value	set X resolution (stdgraph only)
:.yres=value	set Y resolution (stdgraph only)
:.viewport x1 x2 y1 y2	set workstation viewport in world coordinates
:.write[!][+] file	save frame buffer in a spool file
:.zero	reset viewport and redraw frame

Figure 3. The *global* graphics cursor keystrokes.

mode by typing **:.gflush**. The task GFLUSH also "flushes" the frame buffer as well as initializes the graphics system (the last plot will no longer be in the frame buffer!). The CL performs a GFLUSH when you log out of IRAF properly (see §2.3.).

5.2. One-dimensional (vector) plotting tasks

There are several one-dimensional or vector plotting tasks in the PLOT package. Only the IMPLOT task is interactive. The other tasks require =GCUR to activate the interactive graphics cursor.

```
      graph - Graph one or more image sections or lists
     implot - Plot lines and columns of images using cursors
       pcol - Plot a column of an image
      pcols - Plot the average of a range of image columns
 phistogram - Plot or print the histogram of an image or list
   pradprof - Plot or list the radial profile of a stellar object
       prow - Plot a line (row) of an image
      prows - Plot the average of a range of image lines
    pvector - Plot an arbitrary vector in a 2D image
```

The IMPLOT task is the major plotting task used for inspecting two-dimensional data without using the image display (see IMEXAMINE for the image display equivalent). It has its own set of cursor keystrokes (single lower-case letters and ":" commands) that allows a user to overplot line(s) or column(s) from the same or another image plus other features that are unique to its operation. The user should familiarize himself with this task since it is an important task in any data reduction process. Note that the global set of cursor keystrokes are also available as they are in all graphics tasks.

```
cl> implot dev$pix           # plot center line of M51
    ?                        # list cursor keystrokes
    q                        # quit cursor help
    ...                      # experiment with cursor keystrokes
    q                        # quit the task
```

GRAPH is a versatile plotting task for non-interactive use. Input to GRAPH can be either ASCII files or images. The user has a great deal of control over the axis labels, thus GRAPH can produce nearly publication quality plots†.

5.3. Two-dimensional plotting tasks

There are several two-dimensional plotting tasks. These tasks require the use of =GCUR to invoke interactive graphics.

```
    contour - Make a contour plot of an image
     hafton - Generate half-tone plots of an image
    surface - Make a surface plot of an image
    velvect - Plot representation of a velocity field
```

As an aside, the CONTOUR task also has the capability of drawing into the image display window for those sites that have a compatible image display (IIS model 70/75) or display server (IMTOOL or SAOimage). More information on this capability can be found in the online manual pages for IMDKERN. This feature will be expanded to other tasks in the future. In our

†IRAF plots are not considered publication quality at this time; better fonts and axis labelling control is needed.

example the full CONTOUR task command line would not fit on one terminal line so we used the "\" to continue the task to the next line, as mentioned in §3.6.1.

```
cl> display dev$pix 1          # display M51
cl> contour dev$pix \          # overlay contour on image
>>> xres=256 yres=256 perim- fill+ label- ceil=500 dev=imdg
```

5.4. Tasks that manipulate graphics metacode files

Graphics output can be stored in files. These files are called "metacode" files. Graphics metacode files can be generated by the task itself, by the user by redirecting the graphics output to a metacode file on the command line with the **>G** syntax, or by writing the plot directly to a metacode file with the **:.write** option in interactive graphics mode.

```
cl> contour dev$pix >G meta     # redirect plot to metacode file called meta
cl> surface dev$pix             # generate a plot
cl> =gcur                       # invoke interactive cursor mode
    :.write meta                # append current plot to metacode file called meta
    :.read meta                 # display plots in metacode file called meta
    q                           # quit task
```

The following tasks are useful for looking at or plotting metacode files.

```
     gkidir - Directory listing of metacode file
 gkiextract - Extract individual frames from metacode file
  gkimosaic - Condense metacode frames to fit on one page

   stdgraph - Plot metacode on the standard graphics device
    stdplot - Plot metacode on the standard plotter device
```

A metacode file is distributed with your IRAF system and we will use it to demonstrate these tasks.

```
cl> gkidir dev$vdm.gki          # list the contents of the metacode file dev$vdm.gki
cl> gkimos dev$vdm.gki          # plot the metacode file in condensed form
                                    on your terminal
cl> gkiextract dev$vdm.gki 2 | stdgraph
                                # extract and plot the second plot in this file
cl> stdgraph dev$vdm.gki        # plot all files to your terminal
cl> stdplot dev$vdm.gki         # plot all files to your default plot device
```

6. Using the image display

IRAF supports several modes for image display. In the workstation environment, there is support under SunView using a display server called IMTOOL, which is included with the SunOS distributions, and in the X Window environment a display server called SAOimage is available (included with most distributions except SunOS). A copy of SAOimage can be obtained from the network archive in the *iraf.old* directory. IRAF also supports image display for the IIS models 70 and 75, with limited support for the Gould DeAnza IP8400/8500 (VMS only).

The TV package, a subpackage of IMAGES, contains the general image display tools. The tasks in the subpackage IIS are unique to the IIS and the Gould DeAnza, although CV and CVL only work for the IIS.

In the workstation environment the user displays an image into a frame buffer and then uses the display server, either IMTOOL or SAOimage, for panning, zooming, blinking, changing the lookup tables (greyscale and color), and so on. These functions will be done differently depending on the server you are using. There is a UNIX manual page for IMTOOL (**man imtool**), and a LaTeX version of the SAOimage manual is available on your system in *iraf$unix/x11/saoimage/doc/manual.tex* (on Sun systems the *saoimage* directory may be outside the *iraf* root directory - check with your IRAF site manager).

Before doing anything involving image display the environment variable *stdimage* must be set to the correct frame buffer size for the display servers (as described in the *dev$graphcap* file under the section "STDIMAGE devices") or to the correct image display device. The task GDEVICES is helpful for determining this information for the display servers.

```
cl> show stdimage              # show current value
cl> set stdimage=imt800        # set to a display server frame buffer
cl> set stdimage=iism70v       # set to an image display device
```

6.1. Displaying IRAF images

The DISPLAY task is the main task used for displaying images although CVL can be used with the IIS. These tasks have an upper limit of four (4) image planes or frame buffers. In the workstation environment the frame buffers are chunks of memory. IMTOOL uses all four frame buffers but SAOimage only uses one.† An image can be displayed into the first frame buffer by typing

```
cl> display dev$pix 1
```

An image is always loaded into the center of the frame buffer, by default. If the image is larger than the frame buffer only the central portion of the image will be loaded. If the image is smaller than the frame buffer then a border will appear around the image where there is no data.

Images larger than the frame buffer can be "squeezed" into this smaller frame buffer while preserving the original aspect ratio. The following commands demonstrate this capability.

```
cl> set stdimage=imt256             # set smaller frame buffer
cl> display dev$pix 1 fill+
```

Conversely, an image that is smaller than the frame buffer can be displayed so that it fills the frame buffer by block replicating the pixel values. The following commands will demonstrate this.

```
cl> set stdimage=imt1024            # set larger frame buffer
cl> display dev$pix 1 fill+ order=0
```

If you would like to set the intensity range used by the DISPLAY task rather than accept the defaults, then the following execution of the task will prompt you for the *z1* and *z2* values (the minimum and maximum intensity values to be mapped).

```
cl> set stdimage=imt512
cl> display dev$pix 1 zscale- zrange-
```

The sizes of the frame buffers can be changed in the workstation environment but are fixed for the IIS and the Gould DeAnza. For IMTOOL and SAOimage the frame buffer sizes are determined by the entries in the *dev$imtoolrc* file and the similar entries in the *dev$graphcap* file.

As an example of displaying images in the workstation environment, let us assume our images are 512 by 800. Thus we set the frame buffer size accordingly and display an image. The size of the display window (IMTOOL or SAOimage) may need to be adjusted to see the

†Since SAOimage uses only one frame buffer the frame number query parameter in the DISPLAY task can be changed to a hidden parameter: `cl> display.frame.p_mode=hidden`.

full image, or a panning option available with both display servers could be used instead to roam around the image within the smaller display window.

```
cl> gdevices | page
cl> set stdimage=imtcryo
cl> display dev$pix 1          # image not 512x800!
```

The DISPLAY task has many options and the user is encouraged to look at the parameters and the online help pages for the task for further information.

6.2. Using the interactive image cursor

Those IRAF tasks using the interactive image cursor will interact with the servers and the conventional IIS display in the same manner (although the image cursors may appear and act differently on them). These tasks will be discussed briefly below. Currently there are no *global* cursor keys for interactive image cursor mode as there are in interactive graphics cursor mode. Each interactive image mode task does have its own set of interactive image cursor keys however, much like the interactive graphics mode tasks.

The interactive image cursor is controlled by the environment variable *stdimcur*. The normal value for *stdimcur* is *stdimage*, which means the interactive image cursor is being used with the image display. However, *stdimcur* can also be set to *text* or *stdgraph*. This flexibility allows the user to use interactive image cursor tasks that were designed to work with an image display with a contour plot, as an example, by simply setting *stdimcur* to *stdgraph*.

```
cl> show stdimcur                # show current value
cl> set stdimcur=stdimage        # set to image display
cl> set stdimcur=stdgraph        # set to read the grapchics cursor
cl> set stdimcur=text            # set to text mode
```

While the user is in interactive image cursor mode the following "control" characters can be used to control the image display, if the device or server supports multiple frames or frame buffers (thus excluding SAOimage).

- *Ctrl-F*: go to the next frame
- *Ctrl-R*: go to the previous frame
- *Ctrl-B*: manually blink specified frames

6.2.1. Reading the image cursor position

The task RIMCURSOR in the LISTS package can be used to read back the current image cursor position. In its simplest mode RIMCURSOR can be used to make coordinate lists for use as input to other tasks. The output of the task can be redirected to a file. In the following examples, RIMCURSOR is executed after an image has been displayed and automatically goes into interactive image cursor mode. The user then places the cursor on a feature and presses any key to read back the cursor position. The task is exited with an EOF (a *Ctrl-Z* or a *Ctrl-D* for many users).

```
cl> display dev$pix 1            # display an image
cl> rimcursor                    # read out coordinates
cl> rimcursor > coordlist        # redirect coordinates to a file
```

The coordinates returned by RIMCURSOR, by default, are the *x* and *y* pixel coordinates plus the frame number and the key used to terminate the read. Generally only the first two numbers are useful. However, the advanced user is encouraged to read the online help pages for RIMCURSOR and CURSORS to see how to use RIMCURSOR to generate "cursor input files" for some of the interactive tasks (files that allow a task to read cursor positions and commands from a file rather than from the cursor itself).

6.2.2. Examining and editing images

Several tasks exist in the TV package for interactively examining and editing images. These tasks have proven to be extremely useful tools, and the user is encouraged to read the online help pages for details. Do not measure the importance of these tasks by the time and space allotted to them in this manual. These are powerful tasks, and the user should spend some time experimenting with them.

- IMEXAMINE allows the user to interactively examine a displayed image in a wide variety of ways.
- IMEDIT allows the user to interactively edit a displayed image using a wide variety of techniques.
- TVMARK allows the user to draw into the image display as well as to generate or append to coordinate lists.

As a simple exercise we will step through a few of the interactive cursor options in IMEXAMINE. It will be assumed that you have an image display server running, such as IMTOOL or SAOimage, and that you have the environment variable, *stty*, set correctly for vector graphics (see §3.1.1). IMEXAMINE will automatically load the specified image for you. Point the image cursor at a feature in the image before typing a cursor key. Note that IMEXAMINE allows you to use both the interactive image cursor and the interactive graphics cursor!

```
cl> imex dev$pix     # you will now be in interactive image cursor mode
    ?                # list the cursor options
    z                # print the pixel values under the cursor
    a                # print useful information about the star under the cursor
    s                # draw a surface plot of a region
    :epar            # edit the parameters for the s key
    l                # draw a line plot
    :nave 20         # average 20 lines
    g                # change to the graphics cursor
    Z                # zoom the graphics plot
    0                # return to original plot
    i                # change back to image cursor
    r                # draw a radial profile of a feature
    q                # quit
```

7. Coordinate systems within IRAF

IRAF has support for three coordinate systems. The "logical" coordinate system is defined by pixel coordinates relative to the current image or image section. The "physical" coordinate system is also in pixel coordinates but relative to the original or parent image. The "world" coordinates can be in any general world coordinate system such as right ascension and declination or wavelength.

These systems may be better understood by an example. The coordinates of the lower left pixel of the image section (see §8.2.) *dev$wpix[23:33,50:60]* are (1.0,1.0) in logical units, (23.0,50.0) in physical units, and (13:28:04.6,47:24:35.9) in world coordinate units (RA and Dec).

The image *dev$wpix*, distributed with IRAF, has a world coordinate system, mentioned in the previous paragraph, described in its header. The relevant keywords are listed below.

```
CRPIX1  =               257.75
CRPIX2  =               258.93
CRVAL1  =      201.94541667302
CRVAL2  =             47.45444
CDELT1  =        -2.1277777E-4
CDELT2  =         2.1277777E-4
CTYPE1  = 'RA---TAN'
CTYPE2  = 'DEC--TAN'
```

Most of the tasks in the IRAF core system have been modified so that they recognize and use the world coordinate system information in the header and update that information, when necessary. The tasks in the IMAGES package that update the image header, as needed, are IMCOPY, SHIFTLINES, IMSHIFT, MAGNIFY, BLKAVG, BLKREP, IMTRANSPOSE, ROTATE, IMLINTRAN, REGISTER, GEOTRAN, IMSLICE, and IMSTACK. LISTPIXELS and RIMCURSOR in the LISTS package have options for using the various coordinate information, as do IMPLOT, GRAPH, PROW(S), PCOL(S) in the PLOT package and IMEXAMINE in the TV package (these tasks read the logical coordinate system by default). The task WCSRESET (in the PROTO package) can be used to delete the physical or world coordinate system, and WCSEDIT (in the PROTO package) can be used to modify or add coordinate system information.

Further user information about the WCS can be found in *IRAF Newsletters* Number 9 (February/June 1990) and Number 12 (July 1992). Detailed technical information can be found in the file *iraf$sys/mwcs/MWCS.hlp*. Users may find the online help for WCSEDIT to be invaluable in understanding the WCS and the associated keywords from the users point of view.

Some examples of a few common uses of the WCS may be in order at this time. IMPLOT is a good place to start. This little exercise will demonstrate how to switch the axes labels to the world coordinate system and reformat these labels to a more readable form for RA and Dec.

```
cl> implot dev$wpix     # plot middle line
    :w world            # change coordinate system to world
    :f %H               # set RA label to hh:mm:ss.s
    c                   # set cursor - plot column
    :f %h               # set Dec label to dd:mm:ss
    q                   # quit
```

The task LISTPIXELS prints the pixel values for an image; the coordinates of the pixel values are also printed. In the following example we use this task to list the RA and Dec associated with an image section.

```
cl> listpix dev$wpix[10:20,5:6] wcs=world format="%H %h"
```

The task RIMCURSOR reads the image display and returns the coordinate position of the image cursor; this cursor value can be in any coordinate system. In this example the RA and Dec values are returned in hh:mm:ss.s and dd:mm:ss.

```
cl> display dev$wpix 1
cl> rimcursor wcs=world wxformat=%H wyformat=%h
    <spacebar>
    ...
    <EOF> (Ctrl-D or Ctrl-Z)
```

The task WCSLAB will overlay a labelled world coordinate grid on an image. This will work equally as well on a plot or on the image display.

```
cl> contour dev$wpix perimeter-
cl> wcslab dev$wpix dev=stdgraph append+
```

8. Additional interesting topics

A number of interesting features and capabilities of the IRAF system, which have been used in examples throughout this document but have not really been explained in any depth, will be looked at in more detail in this section. The user is also referred to the *User's Introduction to the IRAF Command Language*, by Peter Shames and Doug Tody.

8.1. Input/output redirection

Redirecting input or output from a task has already been demonstrated in many of the examples in this document.

Many tasks will print information to the terminal screen during or at the end of execution. Often you would like to save this information in a file; this is something you will certainly want to do if you are running the task as a background job. The **>** or **>>** signs can be used to redirect output, which normally goes to the terminal, to a file. Error messages can also be redirected to files by including the **&** symbol in the redirection.

```
cl> urand                   # list output to the terminal
cl> urand > flist           # redirect terminal output to new file flist
cl> urand seed=2 >> flist   # append terminal output to file flist
cl> stty <term>             # reset graphics terminal, if necessary
cl> graph flist             # use flist as input to GRAPH

cl> urand >>& flist         # include errors messages in output file flist
```

The output of a task can also be redirected to the input of another task. This is done with the *pipe* symbol, **|**.

```
cl> urand seed=5 | graph    # redirect output from URAND to input of GRAPH
```

Graphics output that normally goes to the terminal can also be redirected to a *metacode* file and plotted later with other plotting tools (see §5.). The symbols for graphics redirection are >**G** and >>**G** (for appending). Note in the following that in the last example the graphics is redirected to a special file called the *null* file, meaning the graphics is actually discarded.

```
cl> contour dev$pix >G meta       # redirect graphics output to new file meta
cl> surface dev$pix >>G meta      # append graphics output to file meta1
cl> gkidir meta                   # list contents of meta
cl> gkimos meta                   # plot meta
    q

cl> contour dev$pix >G dev$null   # discard the plot
```

Redirection can also go the other way. A task that is expecting its input from the terminal can have its input redirected from a file. A common use of this mode is in the execution of a simple script task. In this example the CL itself has its input redirected from a file.

```
cl> cl < <script.cl>
```

8.2. Image sections

Tasks that have images as input will also take "pieces" of images as input. These pieces are described by *image sections*. The syntax for an image section is

[*beginning-x:ending-x,beginning-y:ending-y*]

where *x* refers to the columns and *y* refers to the rows/lines of the image. For example, the following execution of IMCOPY will create a new image that will have 50 columns and 75 lines/rows. The image section is part of the image name and is appended to the image name with no space.

```
cl> imcopy dev$pix[201:250,301:375] newim1
cl> imhead newim1
```

An image section can also be used to describe a new image that contains every n-th pixel in that section. For example, the following execution of IMCOPY will create a new image that has 25 columns and 30 rows.

```
cl> imcopy dev$pix[201:250:2,301:360:2] newim2
cl> imhead newim2
```

Not all tasks will accept image sections as part of the image name on output. IMCOPY is one of the few tasks that allows this. This allows IMCOPY to be used for generating mosaic images. The output image must already exist and be large enough so that the output image section is defined.

```
cl> imcopy dev$pix newpix
cl> imcopy dev$pix[1:200,201:400] newpix[201:400,201:400]
cl> display newpix 1          # display, if possible
```

The *image section* can also be used to do simple coordinate transformations. Some examples using IMCOPY and IMTRANSPOSE are given below.

```
cl> imcopy dev$pix[*,-*] pix1            # flip the rows
cl> imcopy pix1[*,-*] pix1               # flip the rows in place
cl> imcopy dev$pix[-*,*] pix2            # flip the columns
cl> imcopy dev$pix[-*,-*] pix3           # rotate the image 180˚
cl> imtranspose dev$pix[*,-*] pix4       # rotate the image 90˚ CCW
cl> imtranspose dev$pix[-*,*] pix5       # rotate the image 90˚ CW
```

The image section notation can also be used to select a "band" of a three dimensional image, i.e., `<imagename>[*,*,3]`.

8.3. File and image name templates

Templates are character strings including some metacharacters. Templates can be used as input to IRAF tasks; those file names matching the template are used as input.

The following are a few examples of the more commonly used file templates.

```
cl> dir *.cl              # list all files ending in .cl
cl> dir n1*               # list all files beginning with n1
cl> dir n1*.imh           # list all OIF image files beginning with n1
cl> dir *log*             # list all files with log in the name
cl> dir reduced?.imh      # list all OIF image files that have one anything for the ?
cl> del list[1-4]         # delete list files ending in 1 through 4
cl> del [o-r]*            # delete all files that begin with o through r
```

In the last two examples above the "[]" are used to specify a range of characters. This particular syntax can be confusing if the file names being expanded are IRAF images, since this same syntax is used for *image sections*. See below, as well as §8.2. and §8.4.

Image templates can be used as input to many IRAF tasks, but a more sophisticated substitution syntax needs to be used to describe output image names. In the following example the "n1" string in the input image name is replaced with "n2red" in the output image name.

```
cl> imrename n1*.imh %n1%n2red%*.imh
```

A common error for many users is to try to execute the following. **This will not work!** Remember that templates require the file names to already exist, and of course the "n2red" files do not.

```
cl> imrename n1*.imh n2red*.imh   # NO!
```

There are two tasks that can be used to "test" templates. The FILES task can be used on files in general, including image files when there is no image section involved; the SECTIONS task can be used for image files that contain image sections. These tasks simply generate lists of existing files that match the template; no changes are made to any file names on disk.

```
cl> files *.imh                        # generate a list of all OIF image
                                          files in the directory
cl> sections *.imh[200:233,100:800]    # generate a list of all OIF image
                                          files with appended image
                                          sections
```

8.4. The @file

The *@file* (pronounced "at file") can be used for handling large lists of images for input and output. The *@file* is a text file containing a list of images. The easiest way to generate an *@file* is with the FILES or SECTIONS task. This is often the preferred input/output to tasks rather than using templates directly. Since the *@file* is a text file it can also be edited.

```
cl> files *.imh > inlist          # generate list of images
cl> edit inlist                   # edit list as needed
cl> imstat @inlist                # use list as input to task
cl> boxcar @inlist b//@inlist     # use same list as input and output to task;
                                     output names will be the same as input
                                     names with a "b" prepended
cl> copy inlist outlist           # make copy of input list
cl> edit outlist                  # edit output list to contain new names
cl> boxcar @inlist @outlist       # use different lists for input and output
```

8.5. History and command logging

The IRAF CL has a history mechanism that can save the user time by minimizing the number of keystrokes necessary to repeat a similar execution of an already executed task. Only a few basis tools will be discussed here.†

The HISTORY command will print out the last few commands executed (only the information typed on the command line will appear). The number of commands printed can be changed.

The EHISTORY command allows the user to edit and then execute a previous command. When EHISTORY or just **e** is executed the user can then work back up the history tree until the appropriate command is found. The arrow keys or the *Ctrl-[JK]* keys can be used for this purpose. Simple editing can then be done using the "left" and "right" arrow keys along with the

†See, also, the article *Using and Customizing the EPARAM and EHISTORY Editors*, by Rob Seaman, in the *IRAF Newsletter* (Number 7, June 1989).

"backspace" and "delete" keys (modify to the left of the cursor). Both HISTORY and EHISTORY have online manual pages for details.

```
cl> history                     # print out last few commands executed
cl> history 50 > histlog        # redirect the last 50 commands executed to a file
cl> e                           # call up the history editor
                                   - use "return" to execute the task after editing
cl> e rfits                     # edit last RFITS command - use "return" to execute
```

There are also some quick recall commands. Some of them are demonstrated below. These commands will be executed immediately unless the CL parameter *ehinit* includes the string *verify*. (Yes, the CL has its own parameter file; type **lpar cl**.) If *ehinit* has been modified then these commands will behave like EHISTORY.

```
cl> history         # print out last few commands executed
cl> ^35             # recall/execute command 35 from list
cl> ^rfits          # recall/execute last RFITS command
cl> ^l+^l-          # modify last command by replacing "l+" with "l-"
cl> ^rfits:p        # recall and print only the last RFITS command
```

Beginning users may want to generate a log of the commands that they type during their IRAF sessions for later review. And sometimes users will be asked by the IRAF site support to turn on logging for assistance in debugging a local problem. Logging is controlled by three parameters in the CL parameter file. See the online manual pages for LOGGING for details.

```
(keeplog = yes)       Record all interactive commands in logfile?
(logfile = "home$logfile.cl") Name of the logfile
(logmode = "commands nobackground noerrors notrace") Logging control
```

8.6. Using the CL as a calculator

The CL has a built-in calculator capability. Some variables that may be used are defined in the parameter file for the CL which includes the booleans *b1*, *b2*, and *b3*; the integer variables, *i*, *j*, and *k*; the real variables, *x*, *y*, and *z*; and the string variables, *s1*, *s2*, and *s3*. There are a variety of built-in functions that are also available including *sin*, *cos*, *abs*, *exp*, *log*, *log10*, *max*, *min*, *sqrt*, and *tan*.

For more complex examples see the document *An Introductory User's Guide to IRAF Scripts*, mentioned in §9.1.

```
cl> lpar cl
cl> i=1;j=2;x=5;=i+x**j
cl> =x
cl> =sqrt(x/10)
cl> =(sin(0.5)**2+cos(0.5)**2)
```

8.7. IRAF networking

If your system has been configured for IRAF networking then you will be able to access image displays, printers, magtape devices, files, and so on, on other machines in your local area network that also run IRAF and for which you have an account.† Check with your IRAF site manager for the status of IRAF networking at your site. Setting up IRAF networking is described in the various *site manager's guides* that each IRAF site manager should have in his possession. If you have questions or problems in this area contact the IRAF site support for

†This does not mean that all machines have to be running a complete IRAF system. A subset of the necessary files for IRAF networking, i.e., *kernel server* kits, are available for a number of hosts.

further assistance.

The task NETSTATUS can be used to see if IRAF networking has been established for a particular host.

8.7.1. Syntax

Once you have established that IRAF networking is working at your site, it is very easy to execute tasks using this feature. Let us assume that you wish to use the tape drive on another computer (running IRAF) called *orion*. This tape drive is known to IRAF as *mta*.

```
cl> devstatus mta             # check tape status of mta on local machine
cl> devstatus orion!mta       # check tape status of mta on orion
cl> allocate orion!mta        # allocate drive mta on orion
cl> rfits orion!mta ....      # read from tape drive on orion
```

Similarly, directories and files can be accessed on *orion* by including the node name in the directory pathname.

```
cl> dir orion!/scr3/richard/pixels/          # list files in remote
                                               directory
cl> imcopy orion!/home/richard/n1005 pix1    # make new copy of image
                                               n1005 in the current
                                               directory
cl> set m51 = orion!/home/richard/           # define a logical variable, m51
cl> imcopy m51$n1006 pix2                    # copy another image
```

IRAF networking can also be used to display images from another IRAF host. For example, if you are remotely logged into the machine called *orion* and are running IRAF on that host, you can display images from *orion* back onto your local workstation using IRAF networking. You must be running a display server, either SAOimage or IMTOOL, locally on your workstation. You will need to identify the name of your local workstation to the IRAF session on *orion*.

```
cl> set node = <local_machine_name>
```

8.7.2. The *.irafhosts* file

A *.irafhosts* file is created in the user's host login directory when IRAF networking is first initiated. The use of this file is for the most part transparent to the user (although this was not true with earlier releases of IRAF). The reader is referred to *IRAF Version 2.10 Revisions Summary*, §2.4.1.2., for more detailed information about the use of the *.irafhosts* file with IRAF networking.

9. Writing your own software

IRAF currently provides three levels in which the user can implement his own software.

- IRAF scripts, much like UNIX shell scripts or VMS command procedures
- FORTRAN or C programs, written at the host level that are interfaced to IRAF with the IMFORT library routines
- programs in IRAF's own SPP/VOS programming environment

Each of these levels will be mentioned only briefly referring the reader to other documents for details. All of the documents referenced below can be found in the IRAF network archive, unless specified otherwise.

9.1. Writing IRAF scripts

A sequence of repetitive IRAF operations can often be combined into an IRAF script, making the execution of these tasks simpler for the user. If scripts become too complicated, the user is encouraged to make the jump into SPP programming.

Some examples of tasks in IRAF that are actually scripts are MKSCRIPT, ROTATE, REGISTER, IMALIGN and IMLINTRAN. The source code for MKSCRIPT can be found in the *system* directory, the source code for IMALIGN can be found in the *proto* directory, and the source for the others can be found in the *images* directory. By convention a script file ends in *.cl*. Try executing the following for examples of two script tasks.

```
cl> page images$rotate.cl
cl> page proto$imalign.cl
```

Users that are interested in writing scripts should see the document *An Introductory User's Guide to IRAF Scripts*, revised by Rob Seaman, September 1989. The potential script writer should also familiarize himself with the tasks in the LANGUAGE package.

9.2. The IMFORT interface

Often a user would like to use his own programs written in either FORTRAN or C as well as IRAF to reduce and analyze some data. The IMFORT interface provides the capability to read and write the IRAF image format from within FORTRAN or C programs by incorporating subroutine calls from the IRAF IMFORT library into these programs. These host level programs can be executed outside of IRAF or can be redefined to be "foreign" tasks in IRAF and executed in the IRAF environment. For C programs the user will need to acquire the necessary C bindings from the network archive in *iraf.old/cbind.c*.

This programming environment is well documented in *The User's Guide to Fortran Programming in IRAF, The IMFORT Interface*, by Doug Tody, September 1986. An additional memo, *Specifying Pixel Directories with IMFORT*, by Doug Tody, June 1989, covers an important upgrade to the interface.

9.3. Programming in IRAF with SPP

The SPP/VOS (Subset Pre-Processor/Virtual Operating System) programming environment is the native programming environment of IRAF. All of the application software in IRAF is written in this language. The user who decides to step up into SPP programming will have all the tools at hand that all of the IRAF programmers themselves do. Novice SPP programmers are referred to Rob Seaman's manual *An Introductory User's Guide to IRAF SPP Programming*, October 1992. Another useful document is the *SPP Programmer's Reference*, edited by Zolt Levay at Space Telescope Science Institute (available via anonymous ftp to *stsci.edu* in the *software/stsdas/v1.2/doc/programmer/spp* directory).

One of the best ways to learn SPP programming, besides referencing the two manuals above, is by example. And of course the distributed IRAF system is abundant with examples since all source code is included as part of all IRAF distributions!

Appendix A

The IRAF Packages
IRAF Version 2.10
(This list does not reflect any tasks added by the various patches to V2.10.)

- **clpackage.clpackage:**

dataio	-	Data format conversion package (RFITS, etc.)
dbms	-	Database management package (not yet implemented)
images	-	General image processing package
language	-	The command language itself
lists	-	List processing package
local	-	The template local package
obsolete	-	Obsolete tasks
noao	-	The NOAO optical astronomy packages
plot	-	Plot package
proto	-	Prototype or interim tasks
softools	-	Software tools package
system	-	System utilties package
utilities	-	Miscellaneous utilities package

- **clpackage.dataio:**

bintxt	-	Convert a binary file to an IRAF text file
mtexamine	-	Examine the structure of a magnetic tape
rcardimage	-	Convert a cardimage file into a text file
reblock	-	Copy a binary file, optionally reblocking
rfits	-	Convert a FITS image into an IRAF image
rtextimage	-	Convert text files to IRAF images
t2d	-	Fast tape to disk copy
txtbin	-	Convert an IRAF text file to a binary file
wcardimage	-	Convert text files to cardimage files
wfits	-	Convert an IRAF image into a FITS image
wtextimage	-	Convert an IRAF image to a text file

- **clpackage.images:**

blkavg	-	Block average or sum a list of N-D images
blkrep	-	Block replicate a list of images
boxcar	-	Boxcar smooth a list of 1 or 2-D images
chpixtype	-	Change the pixel type of a list of images
convolve	-	Convolve a list of 1 or 2-D images with a rectangular filter
fit1d	-	Fit a function to image lines or columns
fmedian	-	Quantize and median filter a 2-D image or a list of images
fmode	-	Quantize and modal filter a 2-D image or list of images
gauss	-	Convolve a list of 1 or 2-D images with an elliptical Gaussian
geomap	-	Calculate a coordinate transformation
geotran	-	Geometrically transform a set of 2-D images
gradient	-	Convolve a list of 1 or 2-D images with a gradient operator
hedit	-	Header editor
hselect	-	Select a subset of images satisfying a boolean expression
imarith	-	Simple image arithmetic
imcombine	-	Combine images pixel-by-pixel using various algorithms

imcopy	-	Copy an image
imdebug	-	Image debugging package (currently undocumented)
imdelete	-	Delete a list of images
imdivide	-	Image division with zero checking and rescaling
imgets	-	Return the value of an image parameter as a string
imheader	-	Print an image header
imhistogram	-	Compute image histogram
imlintran	-	Linearly transform a list of 2-D images
imrename	-	Rename one or more images
imshift	-	Shift a list of 2-D images
imslice	-	Slice images into images of lower dimension
imstack	-	Stack images into a single image of higher dimension
imstatistics	-	Compute and print statistics for a list of images
imsum	-	Compute the sum, average, or median of a set of images
imsurfit	-	Fit a surface to a 2-D image
imtranspose	-	Transpose a 2-D image
laplace	-	Laplacian filter a list of 1 or 2-D images
lineclean	-	Replace deviant pixels in image lines
listpixels	-	Convert an image section into a list of pixels
magnify	-	Magnify a list of 1-D or 2-D images
median	-	Median filter a 2-D image or list of images
minmax	-	Compute the minimum and maximum pixel values in an image
mode	-	Modal filter a 2-D image or list of images
register	-	Register a set of images
rotate	-	Rotate and shift a list of 2-D images
sections	-	Expand an image template on the standard output
shiftlines	-	Shift image lines
tv	-	Image display load and control package

- **clpackage.images.tv:**

display	-	Load an image or image section into the display
iis	-	IIS image display control package
imedit	-	Examine and edit pixels in images
imexamine	-	Examine images using image display, graphics, and text
tvmark	-	Mark objects on the image display
wcslab	-	Overlay a displayed image with a world coordinate grid

- **clpackage.images.tv.iis:**

blink	-	Blink two frames
cv	-	Control image device, display "snapshot"
cvl	-	Load image display (newer version of ’display’)
erase	-	Erase an image frame
frame	-	Select the frame to be displayed
lumatch	-	Match the lookup tables of two frames
monochrome	-	Select monochrome enhancement
pseudocolor	-	Select pseudocolor enhancement
rgb	-	Select true color mode (red, green, and blue frames)
window	-	Adjust the contrast and dc offset of the current frame
zoom	-	Zoom in on the image (change magnification)

- **clpackage.language:**

intro	-	A brief introduction to IRAF

Language components:

break	*	Break out of a loop
case	*	One setting of a switch
commands	-	A discussion of the syntax of IRAF commands
cursors	-	Graphics and image display cursors
declarations	-	Parameter/variable declarations
default	*	The default clause of a switch
else	*	Else clause of IF statement
for	*	C-style for loop construct
if	*	If statement
goto	*	Goto statement
logging	-	Discussion of CL logging
next	*	Start next iteration of a loop
parameters	-	Discussion of parameter attributes
procedure	*	Start a procedure script
return	*	Return from script with an optional value
switch	*	Multiway branch construct
while	*	While loop

Builtin Commands and Functions:

access	-	Test if a file exists
back	-	Return to the previous directory (after a chdir)
beep	-	Send a beep to the terminal
bye	-	Exit a task or package
cache	-	Cache parameter files, or print the current cache list
cd	-	Change directory
chdir	-	Change directory
cl	-	Execute commands from the standard input
clbye	-	A cl followed by a bye (used to save file descriptors)
clear	-	Clear the terminal screen
defpac	-	Test if a package is defined
defpar	-	Test if a parameter is defined
deftask	-	Test if a task is defined
dparam	-	Dump a pset as a series of task.param=value assignments
edit	-	Edit a text file
ehistory	-	Edit history file to re-execute commands
envget	-	Get the string value of an environment variable
eparam	-	Edit parameters of a task
error	-	Print error code and message and abort
flprcache	-	Flush the process cache
fprint	*	Print a line into a parameter
fscan	*	Scan a list
gflush	-	Flush any buffered graphics output
hidetask	-	Make a task invisible to the user
history	-	Display commands previously executed
jobs	-	Display status of background jobs
keep	-	Make recent set, task, etc. declarations permanent
kill	-	Kill a background job
logout	-	Log out of the CL
lparam	-	List the parameters of a task

mathfcns	-	Mathematical routines
mktemp	-	Make a temporary (unique) file name
osfn	-	Return the host system equivalent of an IRAF filename
package	-	Define a new package, or print the current package names
prcache	-	Show process cache, or lock a process into the cache
print	-	Format and print a line on the standard output
putlog	-	Put a message to the logfile
radix	-	Encode a number in the specified radix
redefine	-	Redefine a task
reset	-	Reset the value of an environment variable
scan	*	Scan the standard input
service	-	Service a query from a background job
set	-	Set an environment variable
show	-	Show an environment variable
sleep	-	Hibernate for a specified time
strings	-	String manipulation routines
stty	-	Set/show terminal characteristics
task	-	Define a new task
time	-	Print the current time
unlearn	-	Restore the default parameters for a task or package
update	-	Update a task's parameters (flush to disk)
wait	-	Wait for all background jobs to complete

- **clpackage.lists:**

average	-	Compute the mean and standard deviation of a list
columns	-	Convert multicolumn file to separate files
lintran	-	Perform linear transformation of a list
rgcursor	-	Read the graphics cursor (makes a list)
rimcursor	-	Read the image display cursor (makes a list)
table	-	Format a list of words into a table
tokens	-	Break a file up into a stream of tokens
unique	-	Delete redundant elements from a list
words	-	Break a file up into a stream of words

- **clpackage.obsolete:**

imtitle	-	Change the title of an image (noao.proto V2.9)
mkhistogram	-	List or plot the histogram of a data stream (noao.proto V2.9)
oimcombine	-	Combine images (images.imcombine V2.9)
radplot	-	PLot the radial profile of an object (noao.proto V2.9)

The previous package and release is listed in parenthesis

- **clpackage.plot:**

calcomp	-	Plot metacode on a Calcomp pen plotter
contour	-	Make a contour plot of an image
crtpict	-	Generate greyscale plots of IRAF images
gdevices	-	List available imaging or other graphics devices
gkidecode	-	Decode metacode on the standard output
gkidir	-	Directory listing of metacode file
gkiextract	-	Extract individual frames from metacode file
gkimosaic	-	Condense metacode frames to fit on one page

graph	-	Graph one or more image sections or lists
hafton	-	Generate half-tone plots of an image
imdkern	-	Image display device (IMD) graphics kernel
implot	-	Plot lines and columns of images using cursors
nsppkern	-	Plot metacode on a NSPP (NCAR) plotter device
pcol	-	Plot a column of an image
pcols	-	Plot the average of a range of image columns
phistogram	-	Plot or print the histogram of an image or list
pradprof	-	Plot or list the radial profile of a stellar object
prow	-	Plot a line (row) of an image
prows	-	Plot the average of a range of image lines
pvector	-	Plot an arbitrary vector in a 2D image
sgidecode	-	Decode an SGI format metacode file
sgikern	-	Simple graphics interface (SGI) graphics kernel
showcap	-	Show and decode graphcap entries
stdgraph	-	Plot metacode on the standard graphics device
stdplot	-	Plot metacode on the standard plotter device
surface	-	Make a surface plot of an image
velvect	-	Plot representation of a velocity field

- **clpackage.proto:**

binfil	-	Create a binary file from an IRAF image
bscale	-	Linearly transform the intensities of a list of images
epix	-	Edit pixels in an image
fields	-	Extract specified fields from a list
fixpix	-	Fix bad pixels by linear interpolation from nearby pixels
hfix	-	Fix image headers with a user specified command
imalign	-	Register and shift a list of images
imcentroid	-	Compute relative shifts for a list of images
imcntr	-	Locate the center of a stellar image
imfunction	-	Apply a function to the image pixel values
imreplace	-	Replace pixels in a range by a constant
imscale	-	Scale an image to a specified (windowed) mean
interp	-	Interpolate for a value in a table of X,Y pairs
irafil	-	Create an IRAF image from a binary data file
joinlines	-	Join text files line by line
suntoiraf	-	Convert Sun rasters into IRAF images
wcsedit	-	Edit the image coordinate system
wcsreset	-	Reset the image coordinate system

- **clpackage.softools:**

generic	-	Preprocess a generic source file
hdbexamine	-	Examine a help database
lroff	-	Lroff (line-roff) text formatter
mkhelpdb	-	Make (compile) a help database
mkmanpage	-	Make a manual page
mkpkg	-	Make or update an object library or package
mktags	-	Tag all procedure declarations in a set of files
mkttydata	-	Build cache for termcap/graphcap device entries
rmbin	-	Find/delete binary files in subdirectories
rmfiles	-	Find/delete files in subdirectories
rtar	-	Read a TAR format archive file

wtar	-	Write a TAR format archive file
xc	-	Compile and/or link a program
xyacc	-	Build an SPP language parser

- **clpackage.system:**

allocate	-	Allocate a device, i.e., magtape drive mta, mtb, ...
concatenate	-	Concatenate a list of files
copy	-	Copy a file or files (use IMCOPY for imagefiles)
count	-	Count the number of lines, words, characters in a text file
deallocate	-	Deallocate a previously allocated device
delete	-	Delete a file or files (use IMDELETE to delete imagefiles)
devices	-	Print information on the locally available devices
devstatus	-	Print the status of a device (mta, mtb, ...)
directory	-	List the files in a directory
diskspace	-	Show how much diskspace is available
files	-	Expand a file template into a list of files
gripes	-	Send suggestions, complaints, etc. to the system
head	-	Print the first few lines of a text file
help	-	Print online documentation
lprint	-	Print a file on the line printer device
match	-	Print all lines in a file that match a pattern
mkdir	-	Create a new directory
mkscript	-	Make a command script
movefiles	-	Move files to a directory
netstatus	-	Print the status of the local network
news	-	Page through the system news file
page	-	Page through a file
pathnames	-	Expand a file template into a list of OS pathnames
phelp	-	Paged HELP: collects and pages the output of HELP
protect	-	Protect a file from deletion
references	-	Find all help database references for a given topic
rename	-	Rename a file
rewind	-	Rewind a device (magtape)
sort	-	Sort a text file
spy	-	Show processor status
tail	-	Print the last few lines of a file
tee	-	Tee the standard output into a file
type	-	Type a text file on the standard output
unprotect	-	Remove delete protection from a file

- **clpackage.utilities:**

curfit	-	Fit data with Chebyshev, Legendre or spline curve
detab	-	Replace tabs with tabs and blanks
entab	-	Replace blanks with tabs and blanks
lcase	-	Convert a file to lower case
polyfit	-	Fit polynomial to list of X,Y data
split	-	Split a large file into smaller segments
translit	-	Replace or delete specified characters in a file
ucase	-	Convert a file to upper case
urand	-	Uniform random number generator

Note: Language package keywords are starred. To get help on a keyword enclose it in quotes.

Appendix B

The NOAO Packages
IRAF Version 2.10
(This list does not reflect any tasks added by the various patches to V2.10.)

- **noao:**

artdata	-	Artificial data generation package	[up]
astrometry	-	Astrometry package	
astutil	-	Astronomical utilities package	[up]
digiphot	-	Digital stellar photometry package	[up]
focas	-	Faint object classification and analysis package	
imred	-	Image reductions package	[up]
mtlocal	-	Magtape i/o for special NOAO format tapes	[up]
nobsolete	-	Obsolete tasks to be phased out in a future release	[up]
nproto	-	Prototype (temporary, contributed) tasks	[up]
observatory	-	Examine and define observatory parameters	[up]
onedspec	-	One dimensional spectral red & analysis package	[up]
rv	-	Radial velocity analysis package	[up]
surfphot	-	Galaxy isophotal analysis package	
twodspec	-	Two dimensional spectral red & analysis package	[up]

- **noao.artdata:**

gallist	-	Make an artificial galaxies list
mk1dspec	-	Make/add artificial 1D spectra
mk2dspec	-	Make/add artificial 2D spectra using 1D spectra templates
mkechelle	-	Make artificial 1D and 2D echelle spectra
mkexamples	-	Make artificial data examples
mkheader	-	Append/replace header parameters
mknoise	-	Make/add noise and cosmic rays to 1D/2D images
mkobjects	-	Make/add artificial stars and galaxies to 2D images
mkpattern	-	Make/add patterns to images
starlist	-	Make an artificial star list

- **noao.astutil:**

airmass	-	Compute the airmass at a given elevation above the horizon
asttimes	-	Compute UT, Julian day, epoch, and siderial time
ccdtime	-	Compute time required to observe star of given magnitude
galactic	-	Convert ra, dec to galactic coordinates
gratings	-	Compute and print grating parameters
pdm	-	Find periods in light curves by Phase Dispersion Minimization
precess	-	Precess a list of astronomical coordinates
rvcorrect	-	Compute radial velocity corrections
setairmass	-	Compute effective airmass and middle UT for an exposure
setjd	-	Compute and set Julian dates in images

- **noao.digiphot:**

apphot	-	Aperture Photometry Package
daophot	-	Dao Crowded-Field Photometry Package
photcal	-	Photometric Calibration Package

ptools	-	Photometry Tools Package

- **noao.digiphot.apphot:**

aptest	-	Run basic tests on the apphot package tasks
center	-	Compute accurate centers for a list of objects
centerpars	-	Edit the centering parameters
daofind	-	Find stars in an image using the DAO algorithm
datapars	-	Edit the data dependent parameters
fitpsf	-	Model the stellar psf with an analytic function
fitsky	-	Compute sky values in a list of annular or circular regions
fitskypars	-	Edit the sky fitting parameters
phot	-	Measure magnitudes for a list of stars
photpars	-	Edit the photometry parameters
polymark	-	Create polygon lists for polyphot
polyphot	-	Measure magnitudes inside a list of polygonal regions
polypars	-	Edit the polyphot parameters
qphot	-	Measure quick magnitudes for a list of stars
radprof	-	Compute the stellar radial profile of a list of stars
wphot	-	Measure magnitudes for a list of stars with weighting
lintran	-	Linearly transform a coordinate list
pexamine	-	Interactively examine or edit an apphot output file
txdump	-	Dump select fields from an apphot output file

- **noao.digiphot.daophot:**

addstar	-	Add artificial stars to an image using the computed psf
allstar	-	Group and fit psf to multiple stars simultaneously
centerpars	-	Edit the centering algorithm parameters
daofind	-	Find stars in an image using the DAO algorithm
daopars	-	Edit the daophot algorithms parameter set
daotest	-	Run basic tests on the daophot package tasks
datapars	-	Edit the data dependent parameters
fitskypars	-	Edit the sky fitting algorithm parameters
group	-	Group stars based on positional overlap and signal/noise
nstar	-	Fit the psf to groups of stars simultaneously
peak	-	Fit the psf to single stars
phot	-	Compute sky values and initial magnitudes for a list of stars
photpars	-	Edit the photometry parameters
psf	-	Fit the point spread function
seepsf	-	Compute an image of the point spread function
substar	-	Subtract the fitted stars from the original image
pappend	-	Concatenate a list of daophot databases
pconvert	-	Convert a text database to a tables database
pdump	-	Print selected fields from a list of daophot databases
grpselect	-	Select groups of a specified size from a daophot database
pexamine	-	Interactively examine and edit a daophot database
prenumber	-	Renumber stars in a daophot database
pselect	-	Select records from a daophot database
psort	-	Sort a daophot database

- **noao.digiphot.ptools:**

istable	-	Is a file a table or text database file ?
pappend	-	Concatenate a list of apphot/daophot databases
pconvert	-	Convert from an apphot/daophot text to tables database
pdump	-	Print selected columns of a list of daophot/apphot databases
prenumber	-	Renumber a list of apphot/daophot databases
pexamine	-	Interactively examine and edit an apphot/daophot database
pselect	-	Select records from a list of apphot/daophot databases
psort	-	Sort a list of apphot/daophot databases
pttest	-	Run basic tests on the ptools package tasks
tbappend	-	Concatenate a list of apphot/daophot tables databases
tbdump	-	Print selected columns of a list of tables databases
tbrenumber	-	Renumber a list of apphot/daophot tables databases
tbselect	-	Select records from a list of apphot/daophot tables databases
tbsort	-	Sort a list of apphot/daophot tables databases
txappend	-	Concatenate a list of apphot/daophot text databases
txdump	-	Print selected columns of a list of apphot/daophot text databases
txrenumber	-	Renumber a list of apphot/daophot text databases
txselect	-	Select records from a list of apphot/daophot text databases
txsort	-	Sort a list of apphot/daophot text databases

- **noao.imred:**

argus	-	CTIO ARGUS reduction package
bias	-	General bias subtraction tools
ccdred	-	Generic CCD reductions
ctioslit	-	CTIO spectrophotometric reduction package
dtoi	-	Density to Intensity reductions for photographic plates
echelle	-	Echelle spectra reductions (slit and FOE)
generic	-	Generic image reductions tools
hydra	-	KPNO HYDRA (and NESSIE) reduction package
iids	-	KPNO IIDS spectral reductions
irred	-	KPNO IR camera reductions
irs	-	KPNO IRS spectral reductions
kpnocoude	-	KPNO coude reduction package (slit and 3 fiber)
kpnoslit	-	KPNO low/moderate dispersion slits (Goldcam, RCspec, Whitecam)
specred	-	Generic slit and fiber spectral reduction package
vtel	-	Solar vacuum telescope image reductions

- **noao.imred.argus:**

apall	-	Extract 1D spectra (all parameters in one task)
apdefault	-	Set the default aperture parameters
apedit	-	Edit apertures interactively
apfind	-	Automatically find spectra and define apertures
aprecenter	-	Recenter apertures
apresize	-	Resize apertures
apsum	-	Extract 1D spectra
aptrace	-	Trace positions of spectra
bplot	-	Batch plots of spectra
continuum	-	Fit the continuum in spectra
dispcor	-	Dispersion correct spectra

dopcor	-	Doppler correct spectra
identify	-	Identify features in spectrum for dispersion solution
msresp1d	-	Create 1D response spectra from flat field and sky spectra
refspectra	-	Assign wavelength reference spectra to other spectra
reidentify	-	Automatically identify features in spectra
sapertures	-	Set or change aperture header information
sarith	-	Spectrum arithmetic
scombine	-	Combine spectra having different wavelength ranges
scopy	-	Select and copy apertures in different spectral formats
setairmass	-	Compute effective airmass and middle UT for an exposure
setjd	-	Compute and set Julian dates in images
slist	-	List spectrum header parameters
specplot	-	Stack and plot multiple spectra
splot	-	Preliminary spectral plot/analysis
doargus	-	Process ARGUS spectra
demos	-	Demonstrations and tests

- **noao.imred.bias:**

colbias	-	Fit and subtract an average column bias
linebias	-	Fit and subtract an average line bias

- **noao.imred.ccdred:**

badpiximage	-	Create a bad pixel mask image from a bad pixel file
ccdgroups	-	Group CCD images into image lists
ccdhedit	-	CCD image header editor
ccdinstrument	-	Review and edit instrument translation files
ccdlist	-	List CCD processing information
ccdproc	-	Process CCD images
ccdtest	-	CCD test and demonstration package
combine	-	Combine CCD images
cosmicrays	-	Detect and replace cosmic rays
darkcombine	-	Combine and process dark count images
flatcombine	-	Combine and process flat field images
mkfringecor	-	Make fringe correction images from sky images
mkillumcor	-	Make flat field illumination correction images
mkillumflat	-	Make illumination corrected flat fields
mkskycor	-	Make sky illumination correction images
mkskyflat	-	Make sky corrected flat field images
setinstrument	-	Set instrument parameters
zerocombine	-	Combine and process zero level images

ADDITIONAL HELP TOPICS

ccdgeometry	-	Discussion of CCD coordinate/geometry keywords
ccdtypes	-	Description of the CCD image types
flatfields	-	Discussion of CCD flat field calibrations
guide	-	Introductory guide to using the CCDRED package
instruments	-	Instrument specific data files
package	-	CCD image reduction package
subsets	-	Description of CCD subsets

- **noao.imred.ccdred.ccdtest:**

artobs	-	Create an artificial CCD observation
demo	-	Run a demonstration of the CCD reduction package
mkimage	-	Make or modify an image with simple values
subsection	-	Create an artificial subsection CCD observation

- **noao.imred.ctioslit:**

apall	-	Extract 1D spectra (all parameters in one task)
apdefault	-	Set the default aperture parameters
apedit	-	Edit apertures interactively
apfind	-	Automatically find spectra and define apertures
aprecenter	-	Recenter apertures
apresize	-	Resize apertures
apsum	-	Extract 1D spectra
aptrace	-	Trace positions of spectra
bplot	-	Batch plot of spectra with SPLOT
calibrate	-	Apply extinction and flux calibrations to spectra
continuum	-	Fit and normalize the continuum of multispec spectra
deredden	-	Apply interstellar extinction corrections
dispcor	-	Dispersion correct spectra
dopcor	-	Doppler correct spectra
identify	-	Identify arc lines and determine a dispersion function
refspectra	-	Assign reference spectra to object spectra
reidentify	-	Reidentify arc lines and determine new dispersion functions
sarith	-	Spectrum arithmetic
scombine	-	Combine spectra
scopy	-	Copy spectra including aperture selection and format changes
sensfunc	-	Create sensitivity function
setairmass	-	Compute effective airmass and middle UT for an exposure
setjd	-	Compute and set Julian dates in images
slist	-	List spectral header elements
specplot	-	Stack and plot multiple spectra
splot	-	Plot and analysis spectra
standard	-	Identify standard stars to be used in sensitivity calc
doslit	-	Process CTIO slit spectra
demos	-	Demonstrations and tests

- **noao.imred.dtoi:**

dematch	-	Match a list of density values to exposure values
hdfit	-	Fit a curve to density, log exposure values
hdshift	-	Align related HD curves
hdtoi	-	Apply DTOI transformation to density image
selftest	-	Self test program to check DTOI transformation
spotlist	-	Generate a list of calibration spot values

- **noao.imred.echelle:**

apall	-	Extract 1D spectra (all parameters in one task)
apdefault	-	Set the default aperture parameters and apidtable
apedit	-	Edit apertures interactively

apfind	-	Automatically find spectra and define apertures
apfit	-	Fit 2D spectra and output the fit, difference, or ratio
apflatten	-	Remove overall spectral and profile shapes from flat fields
apmask	-	Create an IRAF pixel list mask of the apertures
apnormalize	-	Normalize 2D apertures by 1D functions
aprecenter	-	Recenter apertures
apresize	-	Resize apertures
apscatter	-	Fit and subtract scattered light
apsum	-	Extract 1D spectra
aptrace	-	Trace positions of spectra
bplot	-	Batch plots of spectra
calibrate	-	Apply extinction and flux calibrations to spectra
continuum	-	Fit the continuum in spectra
deredden	-	Apply interstellar extinction corrections
dispcor	-	Dispersion correct spectra
dopcor	-	Doppler correct spectra
ecidentify	-	Identify features in spectrum for dispersion solution
ecreidentify	-	Automatically reidentify features in spectra
refspectra	-	Assign wavelength reference spectra to other spectra
sarith	-	Spectrum arithmetic
scombine	-	Combine spectra
scopy	-	Select and copy apertures in different spectral formats
sensfunc	-	Compute sensitivity function
setairmass	-	Compute effective airmass and middle UT for an exposure
setjd	-	Compute and set Julian dates in images
slist	-	List spectrum header parameters
specplot	-	Stack and plot multiple spectra
splot	-	Preliminary spectral plot/analysis
standard	-	Identify standard stars to be used in sensitivity calc
doecslit	-	Process Echelle slit spectra
dofoe	-	Process Fiber Optic Echelle (FOE) spectra
demos	-	Demonstrations and tests

- **noao.imred.generic:**

background	-	Fit and subtract a line or column background
cosmicrays	-	Detect and replace cosmic rays
darksub	-	Scale and subtract a dark count image
flat1d	-	Make flat field by fitting a 1D func. to the lines or columns
flatten	-	Flatten images using a flat field
normalize	-	Normalize images
normflat	-	Create a flat field by normalizing and replacing low values

- **noao.imred.hydra:**

apall	-	Extract 1D spectra (all parameters in one task)
apdefault	-	Set the default aperture parameters
apedit	-	Edit apertures interactively
apfind	-	Automatically find spectra and define apertures
aprecenter	-	Recenter apertures
apresize	-	Resize apertures
apscatter	-	Fit and remove scattered light

apsum	-	Extract 1D spectra
aptrace	-	Trace positions of spectra
bplot	-	Batch plots of spectra
continuum	-	Fit the continuum in spectra
dispcor	-	Dispersion correct spectra
dopcor	-	Doppler correct spectra
identify	-	Identify features in spectrum for dispersion solution
msresp1d	-	Create 1D response spectra from flat field and sky spectra
refspectra	-	Assign wavelength reference spectra to other spectra
reidentify	-	Automatically identify features in spectra
sapertures	-	Set or change aperture header information
sarith	-	Spectrum arithmetic
scombine	-	Combine spectra having different wavelength ranges
scopy	-	Select and copy apertures in different spectral formats
setairmass	-	Compute effective airmass and middle UT for an exposure
setjd	-	Compute and set Julian dates in images
slist	-	List spectrum header parameters
specplot	-	Stack and plot multiple spectra
splot	-	Preliminary spectral plot/analysis
dohydra	-	Process HYDRA spectra
demos	-	Demonstrations and tests

- **noao.imred.iids:**

addsets	-	Add subsets of strings of spectra
batchred	-	Batch processing of IIDS/IRS spectra
bplot	-	Batch plots of spectra
bswitch	-	Beam-switch strings of spectra to make obj-sky pairs
calibrate	-	Apply sensitivity correction to spectra
coefs	-	Extract mtn reduced ceofficients from henear scans
coincor	-	Correct spectra for detector count rates
continuum	-	Fit the continuum in spectra
deredden	-	Apply interstellar extinction corrections
dispcor	-	Dispersion correct spectra
dopcor	-	Doppler correct spectra
extinct	-	Use BSWITCH for extinction correction
flatdiv	-	Divide spectra by flat field
flatfit	-	Sum and normalize flat field spectra
identify	-	Identify features in spectrum for dispersion solution
lcalib	-	List calibration file data
mkspec	-	Generate an artificial spectrum
names	-	Generate a list of image names from a string
powercor	-	Apply power law correction to mountain reduced spectra
process	-	A task generated by BATCHRED
refspectra	-	Assign reference spectra to object spectra
reidentify	-	Automatically identify features in spectra
scombine	-	Combine spectra having different wavelength ranges
sensfunc	-	Create sensitivity function
setairmass	-	Compute effective airmass and middle UT for an exposure
setjd	-	Compute and set Julian dates in images
sinterp	-	Interpolate a table of x,y pairs to create a spectrum
slist1d	-	List spectral header elements

specplot	-	Stack and plot multiple spectra
splot	-	Preliminary spectral plot/analysis
standard	-	Identify standard stars to be used in sensitivity calc
subsets	-	Substract pairs in strings of spectra
sums	-	Generate sums of object and sky spectra by aperture

- **noao.imred.irred:**

center	-	Compute accurate centers for a list of objects
centerpars	-	Edit the centering parameters
datapars	-	Edit the data dependent parameters
flatten	-	Flatten images using a flat field
iralign	-	Align the image produced by irmosaic
irmatch1d	-	Align and intensity match the image produced by irmosaic (1D)
irmatch2d	-	Align and intensity match the image produced by irmosaic (2D)
irmosaic	-	Mosaic an ordered list of images onto a grid
mosproc	-	Prepare images for quick look mosaicing
txdump	-	Select fields from the center task output text file

- **noao.imred.irs:**

addsets	-	Add subsets of strings of spectra
batchred	-	Batch processing of IIDS/IRS spectra
bplot	-	Batch plots of spectra
bswitch	-	Beam-switch strings of spectra to make obj-sky pairs
calibrate	-	Apply sensitivity correction to spectra
coefs	-	Extract mtn reduced ceofficients from henear scans
continuum	-	Fit the continuum in spectra
deredden	-	Apply interstellar extinction corrections
dispcor	-	Dispersion correct spectra
dopcor	-	Doppler correct spectra
extinct	-	Use BSWITCH for extinction correction
flatdiv	-	Divide spectra by flat field
flatfit	-	Sum and normalize flat field spectra
identify	-	Identify features in spectrum for dispersion solution
lcalib	-	List calibration file data
mkspec	-	Generate an artificial spectrum
names	-	Generate a list of image names from a string
process	-	A task generated by BATCHRED
refspectra	-	Assign reference spectra to object spectra
reidentify	-	Automatically identify features in spectra
scombine	-	Combine spectra having different wavelength ranges
sensfunc	-	Create sensitivity function
setairmass	-	Compute effective airmass and middle UT for an exposure
setjd	-	Compute and set Julian dates in images
sinterp	-	Interpolate a table of x,y pairs to create a spectrum
slist1d	-	List spectral header elements
specplot	-	Stack and plot multiple spectra
splot	-	Preliminary spectral plot/analysis
standard	-	Identify standard stars to be used in sensitivity calc
subsets	-	Substract pairs in strings of spectra
sums	-	Generate sums of object and sky spectra by aperture

- **noao.imred.kpnocoude:**

apall	-	Extract 1D spectra (all parameters in one task)
apdefault	-	Set the default aperture parameters
apedit	-	Edit apertures interactively
apfind	-	Automatically find spectra and define apertures
aprecenter	-	Recenter apertures
apresize	-	Resize apertures
apsum	-	Extract 1D spectra
aptrace	-	Trace positions of spectra
bplot	-	Batch plot of spectra with SPLOT
calibrate	-	Apply extinction and flux calibrations to spectra
continuum	-	Fit and normalize the continuum of multispec spectra
deredden	-	Apply interstellar extinction corrections
dispcor	-	Dispersion correct spectra
dopcor	-	Doppler correct spectra
identify	-	Identify arc lines and determine a dispersion function
msresp1d	-	Create fiber response spectra from flat field and sky spectra
refspectra	-	Assign reference spectra to observations
reidentify	-	Reidentify arc lines and determine new dispersion functions
sapertures	-	Set or change aperture header information
sarith	-	Spectrum arithmetic
scombine	-	Combine spectra
scopy	-	Copy spectra including aperture selection and format changes
sensfunc	-	Create sensitivity function
setairmass	-	Compute effective airmass and middle UT for an exposure
setjd	-	Compute and set Julian dates in images
slist	-	List spectrum headers
specplot	-	Stack and plot multiple spectra
splot	-	Plot and analyze spectra
standard	-	Identify standard stars to be used in sensitivity calc
do3fiber	-	Process KPNO coude three fiber spectra
doslit	-	Process KPNO coude slit spectra
demos	-	Demonstrations and tests

- **noao.imred.kpnoslit:**

apall	-	Extract 1D spectra (all parameters in one task)
apdefault	-	Set the default aperture parameters
apedit	-	Edit apertures interactively
apfind	-	Automatically find spectra and define apertures
aprecenter	-	Recenter apertures
apresize	-	Resize apertures
apsum	-	Extract 1D spectra
aptrace	-	Trace positions of spectra
bplot	-	Batch plot of spectra with SPLOT
calibrate	-	Apply extinction and flux calibrations to spectra
continuum	-	Fit and normalize the continuum of multispec spectra
deredden	-	Apply interstellar extinction corrections
dispcor	-	Dispersion correct spectra
dopcor	-	Doppler correct spectra
identify	-	Identify arc lines and determine a dispersion function

refspectra	-	Assign reference spectra to observations
reidentify	-	Reidentify arc lines and determine new dispersion functions
sarith	-	Spectrum arithmetic
scombine	-	Combine spectra
scopy	-	Copy spectra including aperture selection and format changes
sensfunc	-	Create sensitivity function
setairmass	-	Compute effective airmass and middle UT for an exposure
setjd	-	Compute and set Julian dates in images
slist	-	List spectrum headers
specplot	-	Stack and plot multiple spectra
splot	-	Plot and analyze spectra
standard	-	Identify standard stars to be used in sensitivity calc
doslit	-	Process slit spectra
demos	-	Demonstrations and tests

- **noao.imred.specred:**

apall	-	Extract 1D spectra (all parameters in one task)
apdefault	-	Set the default aperture parameters and apidtable
apedit	-	Edit apertures interactively
apfind	-	Automatically find spectra and define apertures
apfit	-	Fit 2D spectra and output the fit, difference, or ratio
apflatten	-	Remove overall spectral and profile shapes from flat fields
apmask	-	Create and IRAF pixel list mask of the apertures
apnormalize	-	Normalize 2D apertures by 1D functions
aprecenter	-	Recenter apertures
apresize	-	Resize apertures
apscatter	-	Fit and subtract scattered light
apsum	-	Extract 1D spectra
aptrace	-	Trace positions of spectra
bplot	-	Batch plot of spectra with SPLOT
calibrate	-	Extinction and flux calibrate spectra
continuum	-	Fit the continuum in spectra
deredden	-	Apply interstellar extinction correction
dispcor	-	Dispersion correct spectra
dopcor	-	Doppler correct spectra
fitprofs	-	Fit gaussian profiles
identify	-	Identify features in spectrum for dispersion solution
msresp1d	-	Create 1D response spectra from flat field and sky spectra
refspectra	-	Assign wavelength reference spectra to other spectra
reidentify	-	Automatically reidentify features in spectra
sapertures	-	Set or change aperture header information
sarith	-	Spectrum arithmetic
scombine	-	Combine spectra
scopy	-	Select and copy apertures in different spectral formats
sensfunc	-	Compute instrumental sensitivity from standard stars
setairmass	-	Compute effective airmass and middle UT for an exposure
setjd	-	Compute and set Julian dates in images
sfit	-	Fit spectra and output fit, ratio, or difference
skysub	-	Sky subtract extracted multispec spectra
slist	-	List spectrum header parameters
specplot	-	Scale, stack, and plot multiple spectra

splot	-	Preliminary spectral plot/analysis
standard	-	Tabuluate standard star counts and fluxes
dofibers	-	Process fiber spectra
doslit	-	Process slit spectra

- **noao.imred.vtel:**

destreak	-	Destreak He 10830 grams.
destreak5	-	First pass processing CL script for 10830 grams.
dicoplot	-	Make dicomed plots of carrington maps.
fitslogr	-	Make a log of certain header parameters from a FITS tape.
getsqib	-	Extract the squibby brightness image from a full disk scan.
makehelium	-	Cl script for processing destreaked 10830 grams(second pass).
makeimages	-	Cl script for processing magnetograms into projected maps
merge	-	Merge daily grams into a Carrington map.
mrotlogr	-	Log some header parameters from a FITS rotation map tape.
mscan	-	Read all sector scans on a tape and put them into images.
pimtext	-	Put text directly into images using a pixel font.
putsqib	-	Merge a squibby brightness image into a full disk image.
quickfit	-	Fit an ellipse to the solar limb.
readvt	-	Read a full disk tape and produce an IRAF image.
rmap	-	Map a full disk image into a 180 by 180 flat image.
syndico	-	Make dicomed print of daily grams 18 cm across.
tcopy	-	Tape to tape copy routine.
trim	-	Set all pixels outside the limb to 0.0 (use sqib for limb).
unwrap	-	Remove effects of data wraparound on continuum scans.
vtblink	-	Blink daily grams on the IIS to check for registration.
vtexamine	-	Examine a vacuum telescope tape, print headers and profile.
writetape	-	Cl script to write 5 full disk grams to tape.
writevt	-	Write an IRAF image to tape in vacuum telescope format.

- **noao.mtlocal:**

ldumpf	-	List the permanent files on a Cyber DUMPF tape
r2df	-	Convert a CTIO 2-d frutti image into an IRAF image
rcamera	-	Convert a CAMERA image into an IRAF image
rdumpf	-	Convert IPPS rasters from a DUMPF tape to IRAF images
ridsfile	-	Convert IDSFILES from a DUMPF tape to IRAF images
ridsmtn	-	Convert mountain format IDS/IRS data to IRAF images
ridsout	-	Convert a text file in IDSOUT format to IRAF images
rpds	-	Convert a PDS image into an IRAF image
rrcopy	-	Convert IPPS rasters from an RCOPY tape to IRAF images
widstape	-	Convert ONEDSPEC spectra to IDSOUT text format

- **noao.nproto:**

binpairs	-	Bin pairs of (x,y) points in log separation
findgain	-	Estimate the gain and readnoise of a CCD
findthresh	-	Estimate a CCD's sky noise from the gain and readnoise
iralign	-	Align the mosaiced image produced by irmosaic
irmatch1d	-	Align and intensity match image produced by irmosaic (1D)
irmatch2d	-	Align and intensity match image produced by irmosaic (2D)
irmosaic	-	Mosaic an ordered list of images onto a grid

linpol	-	Calculate polarization frames and Stoke’s parameters
slitpic	-	Generate IRAF image of aperture slit mask

- **noao.onedspec:**

bplot	-	Batch plots of spectra
calibrate	-	Apply extinction and flux calibrations to spectra
continuum	-	Fit the continuum in spectra
deredden	-	Apply interstellar extinction correction
dispaxis	-	Dispersion axis parameters for 2D images
dispcor	-	Dispersion correct spectra
dopcor	-	Apply doppler corrections
fitprofs	-	Fit gaussian profiles
identify	-	Identify features in spectrum for dispersion solution
lcalib	-	List calibration file data
mkspec	-	Generate an artificial spectrum
names	-	Generate a list of image names from a string
ndprep	-	Make neutral density filter calibration image
refspectra	-	Assign wavelength reference spectra to other spectra
reidentify	-	Automatically identify features in spectra
sapertures	-	Set or change aperture header information
sarith	-	Spectrum arithmetic
scombine	-	Combine spectra having different wavelength ranges
scopy	-	Select and copy apertures in different spectral formats
sensfunc	-	Create sensitivity function
setairmass	-	Compute effective airmass and middle UT for an exposure
setjd	-	Compute and set Julian dates in images
sfit	-	Fit spectra and output fit, ratio, or difference
sinterp	-	Interpolate a table of x,y pairs to create a spectrum
slist	-	List spectrum header parameters
specplot	-	Stack and plot multiple spectra
splot	-	Preliminary spectral plot/analysis
standard	-	Identify standard stars to be used in sensitivity calc

ADDITIONAL HELP TOPICS

package	-	Discussion and overview of package including sections on: spectral formats, dispersion coordinates, and units

- **noao.rv:**

continpars	-	Edit continuum subtraction parameters
filtpars	-	Edit the filter function parameters
fxcor	-	Radial velocities via Fourier cross correlation
keywpars	-	Translate the image header keywords used in RV package
rvcorrect	-	Compute radial velocity corrections

- **noao.twodspec:**

apextract	-	Aperture Extraction Package
longslit	-	Longslit Package

- **noao.twodspec.apextract:**

apall	-	Extract 1D spectra (all parameters in one task)
apdefault	-	Set the default aperture parameters and apidtable
apdemos	-	Various tutorial demonstrations
apedit	-	Edit apertures interactively
apfind	-	Automatically find spectra and define apertures
apfit	-	Fit 2D spectra and output the fit, difference, or ratio
apflatten	-	Remove overall spectral and profile shapes from flat fields
apmask	-	Create and IRAF pixel list mask of the apertures
apnormalize	-	Normalize 2D apertures by 1D functions
aprecenter	-	Recenter apertures
apresize	-	Resize apertures
apscatter	-	Fit and subtract scattered light
apsum	-	Extract 1D spectra
aptrace	-	Trace positions of spectra

ADDITIONAL HELP TOPICS

apbackground	-	Background subtraction algorithms
approfiles	-	Profile determination algorithms
apvariance	-	Extractions, variance weighting, cleaning, and noise model
package	-	Package parameters and general description of package

- **noao.twodspec.longslit:**

background	-	Fit and subtract a line or column background
extinction	-	Apply atmospheric extinction corrections to images
fitcoords	-	Fit user coordinates to image coordinates
fluxcalib	-	Apply flux calibration to images
identify	-	Identify features
illumination	-	Determine illumination calibration
reidentify	-	Reidentify features
response	-	Determine response calibration
setairmass	-	Compute effective airmass and middle UT for an exposure
setjd	-	Compute and set Julian dates in images
transform	-	Transform longslit images to user coordinates

Many of the tasks in the NOAO packages are visible in several different packages. Although these are the same tasks and provide the same functionality, the default parameters are often set differently to account for different instruments or uses.

Appendix C

IRAF Image Formats

IRAF supports several different image formats. The original IRAF format (OIF) has been discussed throughout this document. A brief overview of the other available formats will be give in this Appendix.

The STF (Space Telescope Format) is a fully supported IRAF image format. The STF image has an image header file with a *.??h* extension and a pixel file with a *.??d* extension. Both the header file and the pixel file are stored in the same directory; the *imdir* environment variable plays no role. The header file is a text file but can still be listed with the IMHEADER task. The STF image is not a protected image and can be deleted with the DELETE task as well as with IMDELETE.

The environment variable *imtype* may be set to the STF format so that all new images generated by a reading task will be in this format. But as noted in §3.8. the output image format for tasks will be the same as the input image format unless explicitly set in the output image name, regardless of the value of *imtype*. For example,

```
cl> set imtype = hhh                 # set image type to STF
cl> imcopy dev$pix newpixa           # newpixa is OIF
cl> imcopy dev$pix newpixb.hhh       # newpixb is STF
```

The QPOE (Quick Position Ordered Event) image format supports event (photon) lists data. It is a read only format within IRAF. This image format can be generated by special tasks in the XRAY package distributed by the Smithsonian Astrophysical Observatory. The QPOE image is a single binary file with the *.qp* extension. It is not a protected image file so it can be deleted with DELETE or IMDELETE. Since the QPOE format is read only, the output image format is determined by the value of the environment variable *imtype*, or the output image extension. More information about this format is available in the technical paper, *QuickPOE (Position Ordered Event File) Interface Design*, by Doug Tody, July, 1988. This paper can be found in the *sys$qpoe/QPOE.hlp* file in the IRAF system or in the network archive.

Another format, not used extensively yet in the IRAF system, is the pixel list format. This format has the *.pl* extension. It is a single binary file. This format will be particularly useful in the future to create bad pixel mask files. The user can generate a *.pl* file by including the extension as part of the output image name. More information about this format can be found in the technical paper *iraf$sys/plio/PLIO.help*, by Doug Tody, revised June 1988.

Appendix D

Fixing Pixel File Pathnames

If the IRAF image (OIF) pixel files have been moved to another directory other than the one described in the IRAF image header, the image header keyword "pixfile" will need to be modified to account for the new pixel file pathname (which includes the node or machine name as well). These disconnected pixel files can come about in a number of ways; the UNIX *mv* command may have been used to move the pixel files to another disk, or the disk partition name may have changed for some reason, as examples.

If only a few images need to be modified the task HEDIT can be used directly on each image. In this example the node name was not entered as part of the new pathname thus the host machine's name was automatically inserted when the header was updated, as was intended.

```
cl> hedit
images to be edited: newpix
fields to be edited: pixfile
value expression: /tmp3/irafdir/newpix.pix
newpix,i_pixfile (tucana!/tmp3/jbarnes/newpix.pix -> /tmp3/irafdir/
newpix.pix):
newpix,i_pixfile: tucana!/tmp3/jbarnes/newpix.pix -> /tmp3/irafdir/
newpix.pix
update newpix ? (yes):
newpix updated
```

For large data sets a simple IRAF script can be used.

```
cl> hselect *.imh $I,pixfile yes > modlist
```

The task HSELECT will dump the file names along with the keyword values for *pixfile* to the file "modlist". Now using your favorite editor modify the pixel file pathnames in the file "modlist" so they point to the correct pixel file directory on the proper node. Then execute the following.

```
cl> list="modlist"
cl> {
>>> while (fscan(list,s1,s2) != EOF)
>>> hedit (s1,"pixfile",s2,verify-)
}
```

The task IMHEADER will verify that the headers have been updated correctly and that the pixel file pathnames have been modified, i.e., **imhead *.imh l+ u- | page**.

Appendix E

The IRAF Network Archive

All IRAF distributions and associated documentation are available over the Internet free of charge to all interested parties. The IRAF software has been copyrighted but users are free to share their distributions with others, if this is easier than obtaining individual copies. The IRAF network archive can be accessed in the following manner.

```
     % ftp iraf.noao.edu   (or 140.252.1.1)
  ftp> log in as anonymous
  ftp> use your email address as the password
  ftp> get README
  ftp> quit
```

Most subdirectories in the network archive have associated README files to help you with your selections and transfers. It is a good idea to look at these file first before you attempt to transfer any additional IRAF materials.

An IRAF order form / price list is available in *iraf/v210/ORDERFORM* if you would prefer to order a distribution or documentation by mail. Or we can send you an order form by electronic mail (send your request to *iraf@noao.edu*).

The file *iraf/REGISTER* should be filled out and returned to us (at *iraf-requests*) once you have retrieved an IRAF distribution. We will then add your name to the IRAF mailing list for the *IRAF Newsletter*. The *REGISTER* file can be used by anyone requesting the *Newsletter*.

The *iraf/docs* directory contains all of the IRAF documentation that is currently available, including old copies of the *IRAF Newsletter*. Most of the files in this directory are in compressed PostScript. See Appendix G for a selected list of documents considered relevant to IRAF Version 2.10.

Questions or concerns may be addressed to *iraf@noao.edu*.

Appendix F

IRAF Layered Packages

Additional IRAF software is available to the IRAF user as layered or addon packages. These addon packages are generally distributed as source only systems and require an installed and running IRAF core system before the addons themselves can be installed and executed.

Two major software packages are available as addon packages to the IRAF core system from third party developers.

- STSDAS - developed by the SDAS group at the Space Telescope Science Institute in Baltimore for the analysis of Hubble Space Telescope data (email: *sdas@stsci.edu*).
- XRAY - developed by the PROS group at the Smithsonian Astrophysical Observatory in Cambridge for the analysis of X-ray data from the ROSAT mission (email: *harris@cfa.harvard.edu*)

Other contributed software from outside of NOAO can be found in the *iraf/contrib* directory in the IRAF network archive (see Appendix E). See the *README* file in this directory.

The directory *iraf.old* in the IRAF network archive contains a variety of addon packages from NOAO, including a local package from CTIO. Some of these packages are beta releases of IRAF software that will be included, after user testing, in a subsequent IRAF release. There is no general README file in this directory but there are "readme" files for each of the addon packages themselves. The latest issue of the *IRAF Newsletter* is the best source for a list of the currently available addon packages, or you can send mail to *iraf@noao.edu*.

Appendix G

A Selected List of IRAF Documentation

IRAF Version 2.10

All IRAF documentation is available in the IRAF network archive in the *iraf/docs* directory. A subset of these documents is listed below along with the associated file names in the archive; this list was generated according to the document's relevance to the latest release of IRAF, Version 2.10. Since new documents can be added to the archive at any time please check there for an up-to-date list (see the README file).

Introductory Materials

- *A Beginner's Guide to Using IRAF (IRAF Version 2.10)*, Jeannette Barnes, August 1993, *beguide.ps.Z*
- *A User's Introduction to the IRAF Command Language Version 2.3*, Peter MB Shames and Doug Tody, August 1986, *cluser.ps.Z*
- *Preliminary Test Procedure for IRAF, IRAF Version 2.10*, Jeannette Barnes, revised May 1992, *testproc2.ps.Z*

Cookbooks and Guides

- *A User's Guide to CCD Reductions with IRAF*, Philip Massey, June 1992, *ccduser2.ps.Z*
- *A User's Guide to Reducing Slit Spectra with IRAF*, Phil Massey, Frank Valdes, Jeannette Barnes, April 1992, *spect.ps.Z*
- *Guide to the Coude Three Fiber Reduction Task DO3FIBER*, Francisco Valdes, April 1992, *do3fiber.ps.Z*
- *Guide to the ARGUS Reduction Task DOARGUS*, Francisco Valdes, April 1992, *doargus2.ps.Z*
- *Guide to the Multifiber Reduction Task DOFIBERS*, Francisco Valdes, April 1992, *dofibers2.ps.Z*
- *Guide to the HYDRA Reduction Task DOHYDRA*, Francisco Valdes, April 1992, *dohydra.ps.Z*
- *Guide to the Slit Spectra Reduction Task DOSLIT*, Francisco Valdes, April 1992, *doslit2.ps.Z*
- *Guide to the Slit Spectra Reduction Task DOECSLIT*, Francisco Valdes, April 1992, *doecslit.ps.Z*
- *Guide to the Fiber Optic Echelle Reduction Task DOFOE*, Francisco Valdes, April 1992, *dofoe.ps.Z*
- *A User's Guide to Stellar CCD Photometry with IRAF*, Philip Massey and Lindsey Davis, April 1992, *daophot2.ps.Z*

Programming Guides

- *A User's Guide to Fortran Programming in IRAF The IMFORT Interface*, Doug Tody, September 1986, *imfort.ps.Z*

- *Specifying Pixel Directories with IMFORT*, Doug Tody, June 1989, *imfortmem.ps.Z*
- *An Introductory User's Guide to IRAF Scripts*, revised by Rob Seaman, September 1989, *script.ps.Z*
- *An Introductory User's Guide to IRAF SPP Programming*, Rob Seaman, October 1992, *sppguide.ps.Z*

Other Useful Documents

- *Table of Contents for IRAF Newsletters, TOC_news.txt* (ASCII)
- *Table of Contents for IRAF User Handbook Volume 1A, TOC210_vol1a.txt* (ASCII)
- *Table of Contents for IRAF User Handbook Volume 2B, TOC210_vol2b.txt* (ASCII)
- *The IRAF Packages, IRAF Version 2.10, glos210a.ps.Z*
- *The NOAO Packages, IRAF Version 2.10, glos210b.ps.Z*
- *IRAF Version 2.10 Revisions Summary*, July 1992, *v210revs.ps.Z*