Version 1.3
April 1994

STSDAS User's Guide

SPACE
TELESCOPE
SCIENCE
INSTITUTE

Science Software Branch
Science Computing and Research Support Division
3700 San Martin Drive
Baltimore, Maryland 21218

Operated by the Association of Universities for Research in Astronomy, Inc., for the National Aeronautics and Space Administration

The STSDAS Group

Bob Hanisch	Chief, Science Software Branch
Betty Stobie	STSDAS Programming Supervisor
Dick Shaw	STSDAS Project Scientist
Ray Williamson	System Administration and Distribution
Howard Bushouse	FOS Calibration and Analysis, Synthetic Photometry
Jonathan Eisenhamer	Graphics, Image Display, WCS, GHRS Calibration and Analysis
Phil Hodge	Table System, Fourier Analysis, FOC Calibration and Analysis
J.C. Hsu	HSP, FGS, and WF/PC Calibration and Analysis
Zolt Levay	Graphics, Image Display, Filmwriter, Documentation
Jane Morrison	GASP
Bernie Simon	Calibration Data Base, Table Editor, IRAF System Support
Nelson Zarate	FITS, IRAF System Support, Ports, Benchmarking
Cheng-Yue Zhang	WF/PC II Calibration and Analysis
Mark Stevens	Technical Writing
Fred Romelfanger	X-Windows-Based User Interface
Jinger Mo	Software Testing, Image Restoration

Send comments or corrections to:
STSDAS Group
Space Telescope Science Institute
3700 San Martin Drive
Baltimore, Maryland 21218
E-mail: hotseat@stsci.edu

Table of Contents

List of Figures

List of Tables

CHAPTER 1:

Introduction to IRAF, TABLES, and STSDAS

In This Chapter...

The Space Telescope Science Data Analysis System (STSDAS) is a powerful set of tools and support software used to calibrate and analyze data from the Hubble Space Telescope (HST). A companion package, TABLES, is a set of tools for creating and manipulating tabular data, reading and writing FITS images and tables, and creating customized graphics. STSDAS and TABLES are layered on the Image Reduction and Analysis Facility (IRAF) software from the National Optical Astronomy Observatories (NOAO); you must have IRAF to run STSDAS and TABLES.

STSDAS and TABLES are portable and, because they are layered on IRAF, should run on any system for which an IRAF port exists. STScI, in conjunction with NOAO, actively supports STSDAS and TABLES on VAX systems running VMS (including VAXStations), DECStations running Ultrix, Sun workstations and file servers running Sun/OS 4.1.3, and HP and SGI workstations. Ports to Sun systems running Solaris and to DEC Alpha systems running OSF-1 and Open VMS are under development. HST observers can use both STSDAS and IRAF applications for their data reduction and analysis.

Together, STSDAS and TABLES provide a wide range of data analysis tools, including the following capabilities:

- Calibration of HST science instrument data
- Synthetic photometry and HST throughput modelling
- Interactive curve fitting and surface photometry
- Image restoration
- Fourier analysis
- Non-parametric statistical analysis
- Table creation and manipulation
- FITS image and table I/O
- Graphics tasks tailored to HST data

The STSDAS calibration software is the same software used in the Institute's calibration pipeline. HST observers can, therefore, recalibrate their data, examine intermediate calibration steps, and re-run the pipeline using different calibration switch settings and reference data. STSDAS includes the software needed to generate new versions of calibration reference data and calibration parameters. STSDAS also provides tools for on-site users to access the Calibration Data Base and the Data Management Facility (the on-line archive of HST observations)[1].

In addition to IRAF's support for SPP (Subset Preprocessor) programmers and CL (IRAF command language) script writers, STSDAS supports user-developed applications through a Fortran programming interface, called *f77vos*, within the IRAF environment. This interface provides access to data in both image and table formats, and provides a full set of graphics I/O functions. The TABLES package also contains a full SPP programming interface for table creation and manipulation.

About STSDAS and TABLES...

With the release of Version 1.3, STSDAS has undergone a major restructuring (see the diagram on p. 25). The TABLES package, which used to be a subset of the utilities in STSDAS, is now completely separate. That is, the STSDAS facilities for FITS I/O, table manipulation, and certain graphics tasks are actually provided through the TABLES package.

1. Off-site users can retrieve calibration reference files from the HST Archive using StarView, as explained in the *HST Data Handbook*.

In this sense, STSDAS is layered both upon IRAF and TABLES, and both must be installed before STSDAS is installed. Throughout the rest of this guide we will describe features of STSDAS which apply equally well to TABLES, though the latter may not be named explicitly. But for those sites that do not install STSDAS, descriptions of utilities found only in TABLES are concentrated at the beginning of Chapter 4 on page 59.

Users who are familiar with IRAF will find that TABLES and STSDAS tasks look similar. Standard IRAF command language functions (i.e., the parameter editor, input/output redirection, file name templates, graphics cursor, etc.) work the same with STSDAS tasks as they do with IRAF tasks. In this sense, STSDAS is simply a project-specific extension of the IRAF system.

IRAF and STSDAS work together as complementary packages with minimal overlap. As a general rule, STSDAS does not duplicate software written by NOAO because the NOAO software can be used within STSDAS for analyzing HST data. Similarly, NOAO observers can use STSDAS software that is not available in the standard IRAF distribution, such as the TABLES utilities, or Fourier analysis. The STSDAS software development group at STScI works in cooperation with the IRAF software development group at NOAO to coordinate software development projects.

STSDAS and IRAF have different native image formats, but because all applications programs access image data through an IRAF system interface, data in either format can be both read and written by applications from either NOAO or STScI. There are some important differences, however, and STSDAS users should see the chapter on data formats beginning on page 37 for more details. As a rule, if you are working primarily with STSDAS applications, keep your data in the STSDAS format. Similarly, if you have Kitt Peak (or Cerro Tololo) data and are using primarily NOAO applications, you should keep your data in the IRAF format. The task **imcopy** in the IRAF **images** package can be used to copy data from one format to the other.

IRAF supports a third data format—called Quick Position Oriented Event, or QPOE—for X-ray data, but STSDAS users are not likely to need this format. If you are interested in QPOE format, read the user documentation for the Post-Reduction Off-line Software (PROS) package from the Smithsonian Astrophysical Observatory.

How to Get STSDAS and TABLES...

The STSDAS and TABLES software is provided free of charge to the astronomical community. The software is distributed from STScI as source only (but pre-compiled binaries for several platforms are available from NOAO) on magnetic tape in several formats, or can be copied via anonymous ftp from node `stsci.edu` (`130.167.1.2`). If you are using the ftp server, move to the directory `/software/stsdas/v1.3` and look for the `README` file; this file will explain which files you should retrieve for your system configuration. Users who wish to install STSDAS or TABLES at their home institutions should submit an STSDAS Registration Form to obtain a tape copy of the software and to be added to our mailing list so that you will be informed of new releases and system enhancements. Because STSDAS and TABLES are built on IRAF, sites *must* install IRAF first. If your home institution does not already have IRAF installed, you may simply request STSDAS or TABLES and we will forward a request for IRAF to NOAO on your behalf.

New releases of STSDAS, TABLES, and IRAF are often coordinated, and an update of IRAF may be required with a new release of STSDAS. Your site must be running IRAF V2.10.2 or later to run STSDAS.

You *must* retrieve and install the TABLES package before installing STSDAS.

If you have problems retrieving or installing STSDAS or TABLES, contact the STSDAS group via e-mail at: `hotseat@stsci.edu`

Documentation

The most extensive information about the STSDAS package and tasks can be found in the online help files. These are continuously updated and will always be more current than printed documentation—your first line of inquiry should always be to consult the online help.

The STSDAS group provides various user manuals and other documents to help explain the software and its use. This documentation is provided in printed form to users who request the software on tape from the

Institute. Users who retrieve the software via anonymous ftp can also retrieve a complete set of documentation in the directory `documents/stsdas-docs`. These files are in PostScript form; if you have trouble using PostScript files, please request a printed documentation set from the HotSeat staff or the User Support Branch at STScI.

A complete user documentation set consists of:

- *STSDAS User's Guide* (revised April 1994)
- *IGI User's Guide* (revised October 1992)
- *SPP Reference Manual* (revised October 1992)
- *Synphot User's Guide* (revised September 1993)
- *STSDAS Site Manager's Installation Guide and Reference* (revised September 1993)
- *Site Guide for IRAF and STSDAS* (revised October 1993)
- Quick Reference Cards for IRAF and STSDAS, IGI, Site Managers, and CL programmers
- Documentation on IRAF (see "IRAF Documentation" on page 207)

The *STSDAS Calibration Guide* has been replaced by the *HST Data Handbook*

User Support

HotSeat

The STSDAS staff provides user support Monday through Friday from 9 a.m. to 5 p.m., eastern time. Users experiencing any problems with STSDAS should contact the HotSeat staff. Comments about the software, its use, problems, or about the documentation are encouraged. Although the STSDAS HotSeat staff will answer questions about IRAF, users should refer specific IRAF questions to the IRAF HotSeat at NOAO by sending e-mail to: iraf@noao.edu.

Contacting the STSDAS HotSeat

E-Mail `hotseat@stsci.edu` **Phone** **(410) 516-5100**

Installation Help

If you have any problems retrieving or installing TABLES or STSDAS:

- Read the installation instructions in the *STSDAS Site Manager's Installation Guide and Reference*.
- If you still have problems, contact the STSDAS HotSeat.

First Time Setup for Users

Before running IRAF for the *first time*, you must:

1 • Define your root IRAF directory

2 • Define system logicals and symbols

3 • Run **mkiraf**

First, you must decide which directory you will use as your *root* IRAF directory (also referred to as your *home* directory). Users generally name their IRAF home directory `iraf` and set it up in their account's root directory (i.e., the default directory that you are in when you log in to the system). The IRAF home directory doesn't need to be in your account's root directory, nor does it need to be called `iraf`; it can be anywhere and be named anything, but you should *not* put it on a scratch disk that is periodically erased.

After you create your home directory, switch to that directory and run **mkiraf**, which will create your `login.cl` file, your `uparm` directory, and will define system logicals and symbols. This is done as follows:

VMS:

```
$ CREATE/DIR [.IRAF]
$ SET DEFAULT [.IRAF]
$ IRAF
$ MKIRAF
```

Unix:

```
% mkdir iraf
% cd iraf
% setenv iraf /usr/stsci/iraf/
% source $iraf/unix/hlib/irafuser.csh
% mkiraf
```

The directory name is site-dependent—check with your system staff

The **mkiraf** command initializes IRAF. The command works the same under either VMS or Unix. The **mkiraf** command does two things for you:

- Initializes your `login.cl` file
- Creates your `uparm` directory

After typing the **mkiraf** command, you will see the following:

```
% mkiraf
-- creating a new uparm directory
Terminal types: gterm=ttysw+graphics,vt640...
Enter terminal type:
```

Enter the type of terminal or workstation you will most often use with IRAF. Terminal types and workstation emulators will vary from site to site and you should look at the `dev$termcap` file for a complete list of terminal types.[2] There are literally dozens of terminal types to choose from, and a good match between your hardware and the `termcap` entry will allow IRAF to take full advantage of your terminal's graphics and text capabilities. However, the match need not be perfect for IRAF to function reasonably well (and you can always change it later). Three generic terminal types will work for most users:

- `vt100` works for most terminals (and workstations too, after a fashion)
- `xtermjhs` is preferred for most workstation window managers
- `gterm` works for Sun workstations running under SunView

2. Users at STScI should consult the *STScI Site Guide for IRAF and STSDAS*.

After you enter your terminal type, you will see the following output before getting your regular prompt:

```
A new LOGIN.CL file has been created in the current ...
You may wish to review and edit this file to change ...
```

The `login.cl` file is the *startup file* used by the CL. It is similar to the `LOGIN.COM` file used by VMS or the `.login` file used by Unix. Whenever IRAF starts, it looks at the `login.cl` file. You can edit your copy of the file to customize your IRAF environment. In fact, you *should* look at it to make sure that everything in it is correct. In particular, there is a line starting with `set home =` that tells IRAF where to find your IRAF home directory. You should verify that this statement does, in fact, point to your IRAF directory. If you will be using IRAF format images, you should also check the setting of `imdir`, which is the directory on disk where IRAF will store your pixel files (more information on IRAF format images is provided in "IRAF Images" on page 38).

If you will be using IRAF networking to display images across the network, you should also check the setting of the `node` variable, which you would set to the name of your workstation, for example:

```
set node     = "spudley.stsci.edu"
```

The `uparm` directory will contain your own copies of IRAF task parameters. This allows you to further customize your IRAF environment by setting certain parameter values as defaults. Once you set up IRAF, you should rarely need to do it again (you might need to if a major revision is installed on your system).

Now you are ready to use IRAF. To start IRAF's command language, type `cl`. IRAF starts by displaying several lines of introductory text and then puts a prompt at the bottom of the screen. Figure 1.1 is a sample IRAF startup screen.

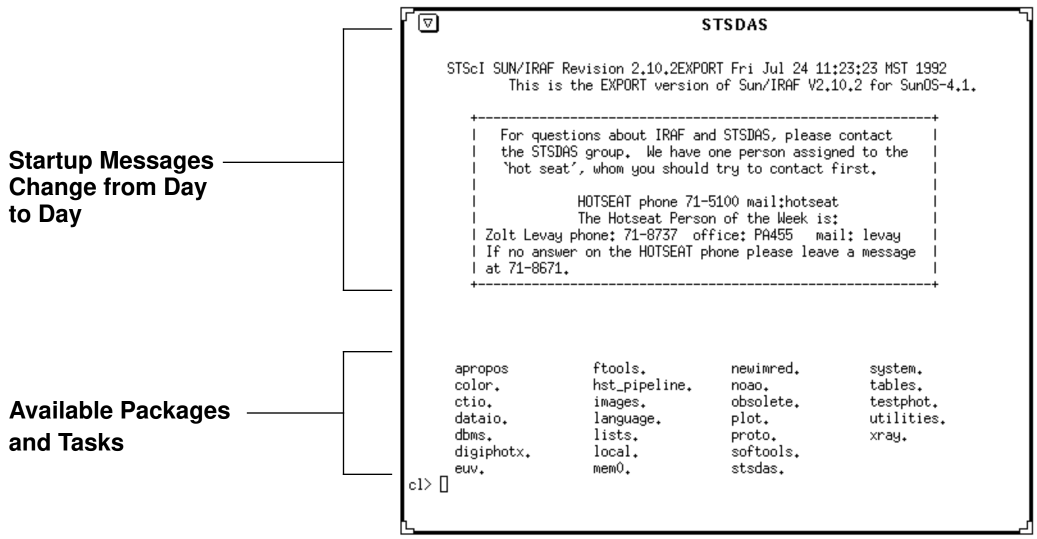

Figure 1.1: IRAF Startup Screen

CHAPTER 2:

IRAF and the CL

In This Chapter...

STSDAS is the software system for calibrating and analyzing data from the Hubble Space Telescope. The package is large and powerful, containing programs—called *tasks*—to perform a wide range of functions supporting the entire data analysis process, from reading your tapes, through recalibration and analysis, to producing your final plots and images.

STSDAS is built on top of the IRAF system developed at NOAO. This means that any task in IRAF can be used in STSDAS and that the software will be portable across a number of platforms and operating systems. To effectively exploit the power of STSDAS, you will need to learn the basics of IRAF.

This chapter will include information about how to use IRAF for the first time, as well as background information about IRAF's underlying concepts. Basic information about reading FITS tapes and displaying images and spectra is also provided. For more information about basic IRAF concepts and usage, see *A Beginner's Guide to Using IRAF*, available from the IRAF group at NOAO.

Concepts

This section is for new users. It describes basic techniques such as:

- Loading packages
- Running tasks and commands (page 13)
- Viewing and setting parameters (page 15)
- Setting and using environment variables (page 18)

The typical way of doing things in IRAF is:

1 • Define any environment variables that you may want to use during your session, such as the location of reference files and tables.

2 • Load the packages you want to use.

3 • Edit parameter sets for the tasks you want to use.

4 • Run the tasks.

Each of these steps are described in the following sections, although not in this order because the most fundamental and commonly-used concepts are described first. For example, not every user will need to redefine environment variables, especially during a first session.

Loading Packages

In IRAF terminology, a command or program is called a *task* and logically-related tasks are grouped together in a *package*. Before you can use a task, its package must be loaded.

- Load a package by typing its name.
- When you load a package, the prompt changes to the first two letters of the newly-loaded package.
- As the package loads, a list of newly-available tasks and subpackages is displayed.

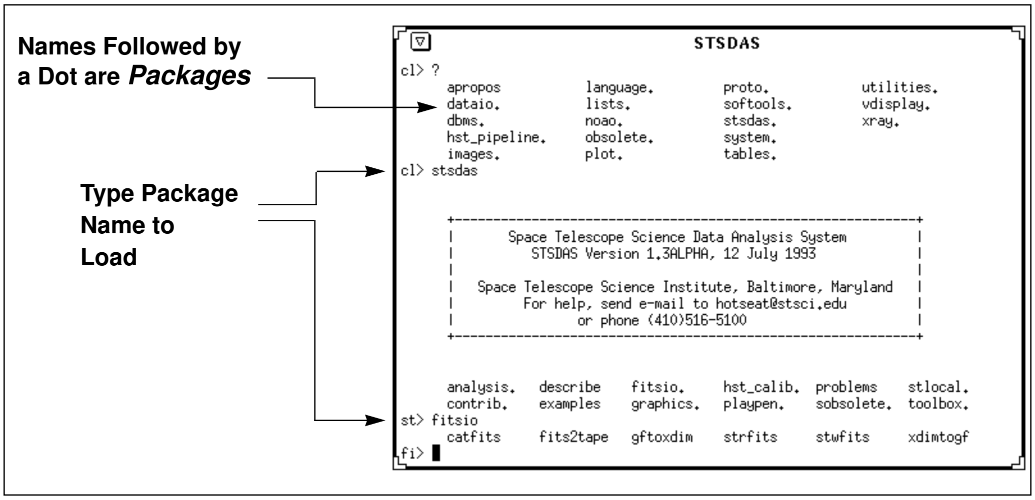

Figure 2.1: Loading Packages

Some commands that will help you manage packages are:

- `?` - List tasks in the current package.
- `??` - List all tasks loaded, regardless of package.
- `package` - List names of all loaded packages.
- `bye` - Exit the current package.

Note that many packages, including TABLES and its packages, are implicitly loaded when the STSDAS package is loaded. See the diagram on page 25.

Tasks and Commands

This section explains how to run tasks, run background tasks, run system-level commands, and use piping and redirection. Tasks in the IRAF environment usually operate on static disk files.

Running a Task

Run a task by typing its name.

Normally, you would also specify values for any required *parameters* on the command line when you run the task, however, you don't need to. If you start a task by simply typing its name, the task will prompt you for any required information, such as the names of input files, starting values, etc. Note that minimum matching is used for task and parameter names; this means that you need only type enough characters to uniquely identify the task or parameter name, for example, instead of typing the full task name `eparam`, you need only type `epar`.

The use of parameters is an important concept and will be discussed in the next section.

Background Tasks

To run a task as a background job, freeing your workstation window for other work, add an ampersand (&) to the end of the command line. For example:

```
cl> taskname &
```

When you use background jobs, you would typically redirect your output and any possible error messages to a file (as explained below) to prevent the background task output from interfering with your interactive session. For example:

```
cl> taskname >& task.out &
```

Escaping System Level Commands

To run an operating system-level command (i.e., Unix or VMS commands) from within the IRAF CL, precede the command with an exclamation point (!). This is called *escaping* the command. For example:

```
cl> !system_command
```

Piping and Redirection

You can run tasks in sequence, with the output of one task being used as the input for another. This is called piping and is done by separating commands with a vertical bar (|). For example:

```
cl> command filename | task2
```

You can also save output from any task or command to a file by using the greater-than symbol (>), for example:

```
cl> command > outputfile
```

You can also direct *both* the standard task output (called STDOUT) and any error messages (called STDERR) to a file by appending an ampersand to the greater-than symbol, for example:

```
cl> command >& outputfile
```

Batch Jobs

You can submit tasks to run as batch jobs (usually tasks that would take a long time to run and that don't require user interaction) using the `&batch` syntax, for example:

```
cl> task >& outfile &batch
```

Setting Parameters

Information is provided to tasks through *parameters*. Parameters are used to specify names of input or output files, starting or ending values, option settings, and many other types of information that controls the behavior of the task.

The two most useful commands for handling parameters are:

- **lpar** to display the current parameter settings.
- **epar** to edit parameters (page 16).

Parameters can be set in several ways:

- From the command line; for example, you could type:

  ```
  cl> task.parameter="value"
  ```

- When running the task; for example:

  ```
  cl> task parameter_value hidden_par=value
  ```

  Note that in the above example `parameter_value` represents a positional parameter while `hidden_par` represents a hidden parameter (these are described below).

- Using **epar**.

Parameters on the Command Line: Positional and Hidden

There are two types of parameters that you would set from the command line when running tasks:

- *Positional parameters*: These are required parameters that must be specified in a certain order. For example, input and output files are often specified by name on the command line, and must appear in the correct order. For example, the **imcopy** task takes an input image name and an output image name:

  ```
  cl> imcopy oldimage.hhh copy.hhh
  ```

- *Hidden parameters*: These show up in an **lpar** listing in parentheses. If you specify them on the command line, you must set them by specifying the name of the parameter followed by an equal sign (=) and the value that you want the parameter to take. For example, to use **strfits** to read a tape and optionally convert blanks to -1, you would type:

```
fi> strfits mta 1-3 images blank=-1
```

Viewing Parameters with lpar

The **lpar** command lists the current parameter settings for a given task (Figure 2.2).

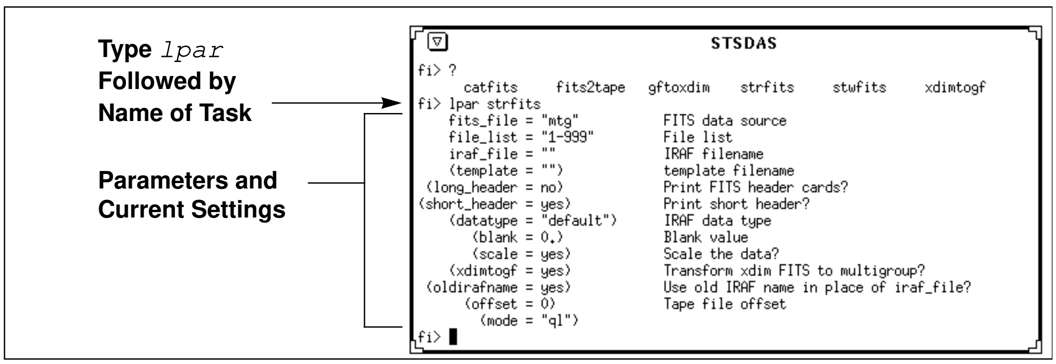

Figure 2.2: Displaying Parameter Settings with lpar

Setting Parameters with epar

The **epar** command is an interactive parameter set editor. All of the parameters and their current settings are displayed on the screen and you can move around the screen using the arrow keys (also called *cursor* keys) and can type new settings for any parameters you wish to change. Figure 2.3 shows a sample of the **epar** editor at work.

```
fi> epar strfits
```

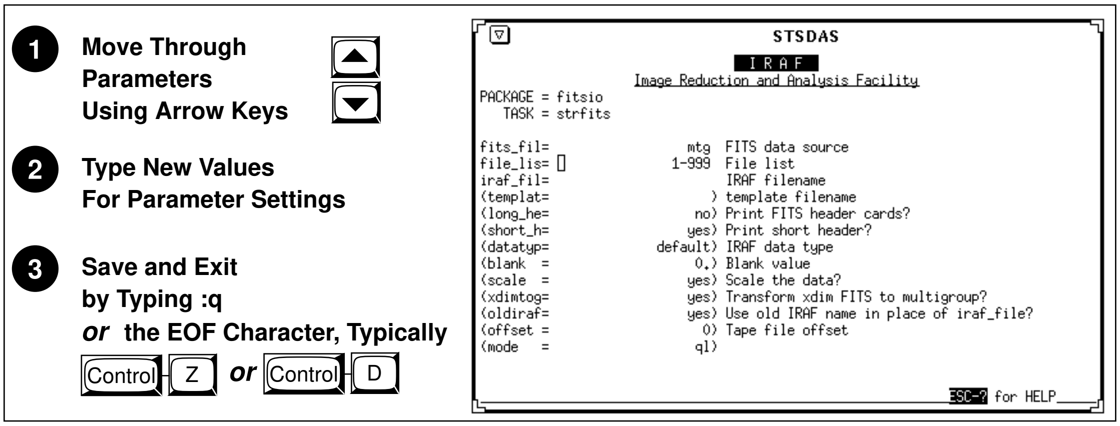

Figure 2.3: Editing Parameters with epar

Parameter Data Types—What to Specify

Parameters are either *required* or *hidden*, and each parameter expects information of a certain *type*. Usually, the first parameter is required, and very often it expects a file name. Parameters are described in the help file for each task. When a parameter is shown in parentheses (either in the help file or in the **lpar** listing), the parameter is hidden.

If you type an invalid value for a parameter, **epar** has some error checking capability and will display an error message. For example, the message might say "Parameter Value is Out of Range." The message is displayed when you move to another parameter or if you press Return. Parameter types are listed in Table 2.1.

Type	*Description*
`File Name`	Full name of the file. Wild card characters are often allowed. Some tasks will allow you to use special features when specifying file names, including “@” lists, IRAF networking syntax, and image section or group syntax. (See “Working with Files”below).
`Integer`	Whole number. Often the task will enforce minimum or maximum values (see the help pages).
`Real`	Floating point numbers, can be expressed in exponential notation. Often the task will enforce minimum and maximum values.
`Boolean`	Logical “yes” or “no” values.
`String`	Any characters. Sometimes file names are specified as string.
`Pset`	Parameter set.

Table 2.1: Parameter Data Types

Restoring Parameter Default Values

Occasionally, IRAF will get confused by your parameter values. If this happens, you can restore the default parameters with the **unlearn** command, which restores the default parameter values. You can use **unlearn** on either a task or on an entire package.

The **unlearn** command can be used if you start having problems with IRAF or STSDAS. Parameter conflicts are the most common source of trouble for the new user.

Setting Environment Variables

IRAF uses *environment variables* to define which devices are used for certain operations. For example, your terminal type, default printer, and disk and directory used for storing images are all defined through environment variables. Environment variables are set using the **set** command. Table 2.2 lists some of the environment variables that you might want to change along with examples of how they are used.

To display the current setting for an environment variable, use the **show** command. Typing `show` by itself will display *all* environment variables, typing the name of a variable on the command line will show only the setting for that variable. For example, if you wanted to see which printer your `printer` environment variable was set to use, you would type the following command:

```
st> show printer
```

Variable	*Description*	*Example of Changing*
`printer`	Default printer for text	`set printer = lp2`
`stdplot`	Default printer for all graphics output	`set stdplot = ps2`
`stdimage`	Default terminal display setting for image output (most users will want this set to either imt512 or imt800)	`set stdimage = imt800`
`clobber`	Allows or prevents overwriting of files	`set clobber = yes`
`imtype`	Default image type for output images. "imh" is old IRAF format, "hhh" is STSDAS GEIS format	`set imtype = "hhh"`

Table 2.2: Environment Variables

The show command is especially useful for setting variables to long directory path names.

Working with Files

This section describes:

- File structures commonly used in STSDAS and IRAF.
- How to use directory specifications.
- How to specify file names.

File Structures

Images used by IRAF and STSDAS consist of two files. The two files are *always* used together as a pair and will always have a file name that differs *only* in the last character of the extension:

- A *header file*, which consists of descriptive information; these are ASCII text files for STSDAS images and a combination of ASCII text and binary for IRAF images. IRAF header files are identified by an extension of `.imh`. STSDAS header files are identified by an extension of `.hhh` or another extension ending in "h", such as `.c0h` or `.q1h`.
- A *binary data file*,[1] consisting of pixel information. IRAF data file names end with a `.pix` extension. STSDAS data files end with an extension of `.hhd` or another extension that ends with "d", such as `.c0d` or `.q0d`.

STSDAS always expects both component files of a Generic Edited Information Set (GEIS) image to be kept together in the same directory. Both components *must* have the same root name.

When working with IRAF or STSDAS images, you need only specify the header file name—the tasks will automatically use the binary data file when necessary.

IRAF and STSDAS image formats are not similar in any way, other than that they each have two component files. The IRAF header is binary and has no fixed format, whereas the STSDAS format specifies 80 byte records consisting of ASCII text. The IRAF binary file consists of a grid of binary data preceded by a 512-byte binary description. The STSDAS binary file consists of a series of groups of binary data, each representing an entire image, and blocks of group description information called the *Group Parameter Block* (GPB).

See Chapter 3 on page 37 for more information about data formats.

1. The binary data file format is host-dependent and may need to be converted before it can be moved to a computer using a different architecture. The tasks that do this are **sun2vax** and **vax2sun**.

Directory Specification

As we mentioned above, the two components of an STSDAS image are expected to be in the same directory, and tasks will look for these files in whatever directory you are currently located.[2]

To navigate through directories, you should use the following commands:

- **path** - Lists the current working directory.
- **cd *directory*** - Move to the named directory.
- **show imdir** - Lists the directory where IRAF pixel files will be stored.

Specifying Files

Most tasks in IRAF and STSDAS operate on files and expect you to specify a file name for one or more parameters. There are several types of special syntax that can be used with certain tasks when specifying file names. These syntax features include:

- **Wild card characters**, often called *templates*: These are used to specify multiple files using pattern matching techniques. The wild cards are:
  - `*` Matches any number of characters, e.g., `z*.c0h`
  - `?` Matches any single character, e.g., `z01x23x.c?h`

When using wildcards with image-processing tasks, be sure to exclude the binary pixel files by ending your file name specification with an "h", for example: `y*.??h`

2. This is different than the way in which IRAF images are often handled, with separate directories for the header and pixel files.

- **List files**, often called *@-files*: A list file is an ASCII file that contains a list of file names, one per line. If the task supports the list file feature, you would type the name of your list file, preceded by the "@" character. Figure 2.4 shows an example of how @-files are built and used.

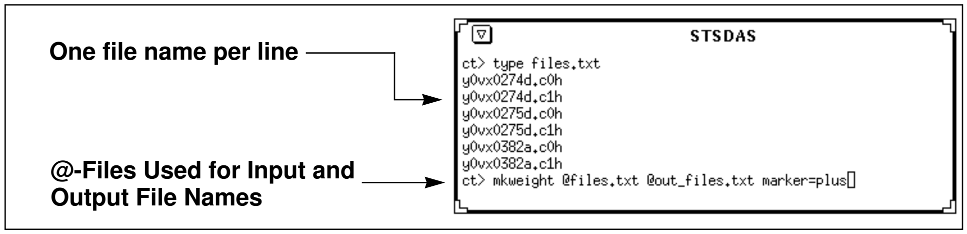

Figure 2.4: Using @-Files

- **Image section** specification: Tasks that work with image data will often let you specify that you want to work on only a small area of the image rather than the entire image. (See "Displaying Images" on page 26.) To specify an image section, specify the X axis range and the Y axis range in square brackets, for example: `image.hhh[10:200,20:200]`
- **Group** specification: HST images are stored in a format known as *GEIS multigroup format*. This lets one file contain one or more sets of related data, such as images taken from different detectors or time-resolved spectra. Many STSDAS tasks will let you choose which of the groups in a multigroup image will be used. To specify a group, enclose its number in brackets, for example: `imagefile.hhh[7]`. A description of GEIS format is available on-line within STSDAS by typing `help geis`.
- **IRAF networking** specification: IRAF is capable of reading and writing files to and from remote systems on a network. This is often used with tasks in the **fitsio** and **toolbox.convfile** packages. You must do some work to enable this feature, however, and this is described in an online help file that can be read by typing `help networking`. To specify that you want to use the IRAF networking feature, you will specify the remote host name followed by an exclamation point (!), followed by the file name. For example: `ra!mta`.

On-Line Help

This section describes:

- How to use IRAF's on-line help facility.
- How to find a task that does what you want (page 24).
- How to get user support if you have trouble (page 25).

Getting Help

Online help is displayed using the help command,[3] which takes the name about which you want help. For example:

```
fi> help mkmulti
```

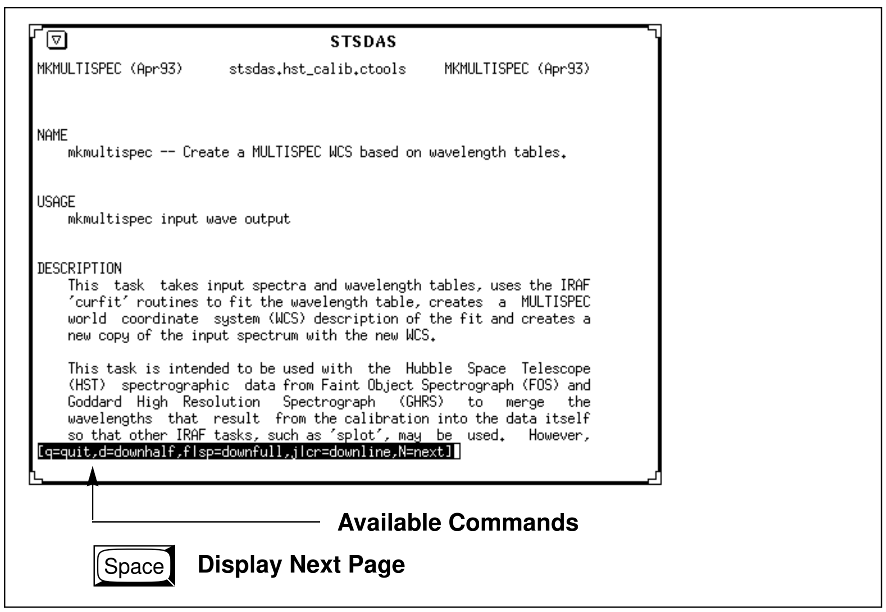

Figure 2.5: Displaying On-line Help

3. There is an optional *paging* front-end for help called **phelp**. For more information, type `help phelp` from within IRAF.

Two STSDAS tasks are available that will display only certain sections of the help file.

- **examples** - Displays only the examples for a task.
- **describe** - Displays only the description.

Finding Tasks

There are three ways to find a task that does what you need:

- Use the **apropos** task to search the apropos database.
- Look at the package structure to find the right package.
- Type `help` *`package`* to list one-line descriptions of each task in the package.

Using apropos

The **apropos** task looks through a list of IRAF and STSDAS package menus to find tasks that match a specified keyword. Figure 2.6 shows how to use **apropos**. Note that the name of the package in which the task is found is shown in parentheses. (A similar task is **references**.)

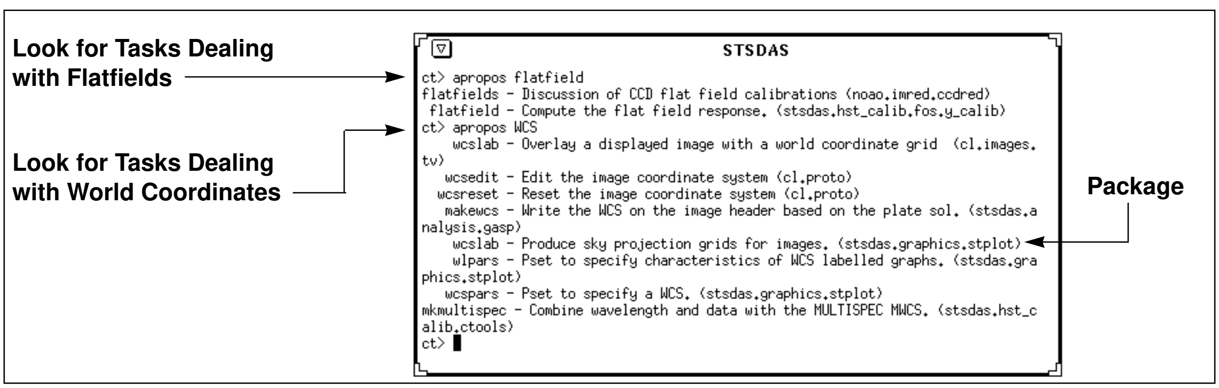

Figure 2.6: Using apropos

STSDAS Package Structure

STSDAS version 1.3 is structured so that tasks are logically grouped. In many cases, you can easily find a task that does whatever function you need simply by looking in the appropriate package. For example, all of the tasks that are used in the calibration process can be found in the **hst_calib** package and tasks used for general image display and plotting can be found in the **graphics** package. Figure 2.7 shows the STSDAS package structure.

Sample data sets are also provided with STSDAS so that you can experiment with tasks for which you might not otherwise have suitable data. The sample data sets are found in the directory `scidata$`.

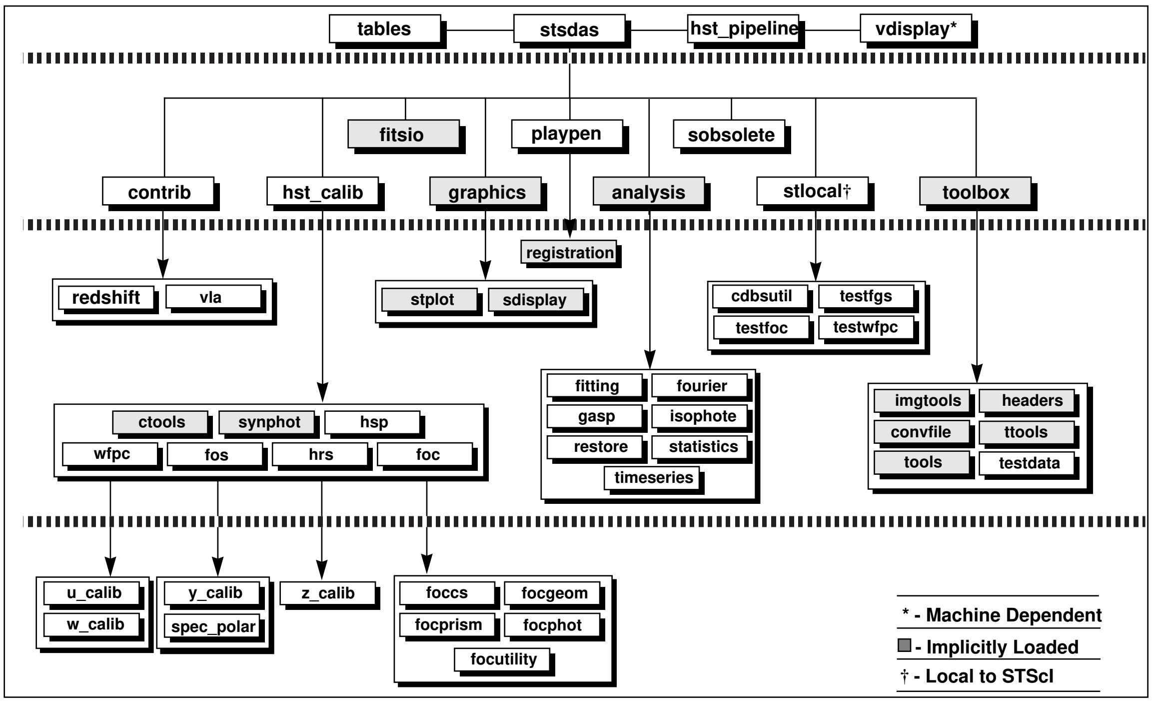

Figure 2.7: STSDAS Version 1.3 Package Structure

High-level descriptions of nearly every STSDAS package are available within the IRAF online help facility using the `opt=sys` qualifier. For example, to see an overview of the packages and tasks in the **hst_calib** package you could type: `help hst_calib opt=sys`

User Support—HotSeat

If you have problems using STSDAS or IRAF and you can't find the solution in the manuals or the on-line help, you can contact the STSDAS user support staff by sending e-mail to: `hotseat@stsci.edu`

Before calling the HotSeat, there are a couple of easy things you can do to make sure that you don't have a simple memory or parameter conflict:

- Look at the parameter settings and make sure that you have specified reasonable values for every parameter.
- Use the **flprcache** command twice to clear the process cache. To do this type: `flpr`
- Occasionally, you may need to logout of the CL, restart IRAF, and try your command again.
- Read the troubleshooting information provided in Chapter 8 (page 173).

Using Images

This section will primarily be of interest to WF/PC and FOC observers, it explains:

- How to display images in IRAF.
- How to print a displayed image.
- How to work with HST multigroup format images (page 28).
- Basic tools for working with images (page 29).

Displaying Images

To display an image, you need to:

1 • Start SAOimage in a separate window from your IRAF session (i.e., either from a different xterm window, or as a background job before starting IRAF).

2 • Display the image using the IRAF **display** task.

Figure 2.8 shows how to display the standard IRAF `dev$pix` image.

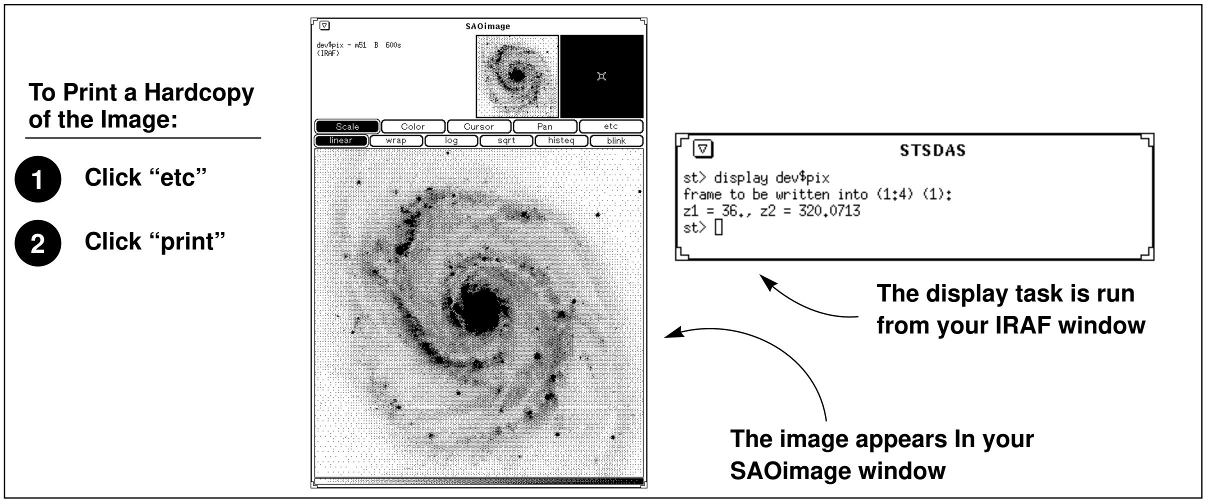

Figure 2.8: Displaying an Image

Once the image appears in your SAOimage screen, you can use the SAOimage commands to manipulate or print your image.

Printing Images

The “print” button on the “etc” menu in SAOimage will print the visible display to a PostScript printer. You may optionally save the captured screen to a PostScript file by reconfiguring the SAOimage environment before starting it. (See the *SAOimage User’s Guide* for instructions.)

Any displayed window may be saved as a raster image using one of several host (i.e., non-IRAF) tools. These tools include:

- The Unix **xwd** command saves any visible window as an X-Window dump (xwd) format image.
- The X-Windows tool **xv** can capture any visible window, manipulate it, and save it in one of several popular image formats.
- The OpenWindows **snapshot** tool captures screens as raster images.

Perhaps the most universally useful format for printing or importing images into documents is PostScript. The **xv** program provides a very nice way to save images in PostScript. In addition, the PBM+ suite of programs include the ability to write PostScript, among other formats.

Non-PostScript printers require various image formats to print continuous-tone images. Usually, there is a specific tool that will format images appropriately for a given printer. For example, the EasyCopy/X (**ecx**) program reformats images from Sun rasterfile format to the format used by Mitsubishi Shinko color printers. (See "Shinko Thermal Color Printer" on page 26 of the *STScI Site Guide for IRAF and STSDAS*).

Image Sections and Groups

HST images are stored in a format called *GEIS format*, which actually consists of one or more related images, each in a separate *group*. You can work with the groups individually by simply enclosing the group number in square brackets. For example, in Figure 2.9, we show only group 2 of a WF/PC image.

If you do *not* specify a group, most tasks will operate on *only* the first group in the image.

You can also work with only part of the total image by specifying a pixel range, also enclosed in square brackets. The range is the starting point and ending point, with a colon separating the two. The X (horizontal) axis is specified first, then the Y (vertical) axis, separated by a comma. For example, specify group 3 of an image as: `image.hhh[3]`. You could specify a pixel range from 101 to 200 in the X direction and all pixels in the Y direction as: `image.hhh[101:200,*]`

If you use both group and image section syntax together, the group number must come first.

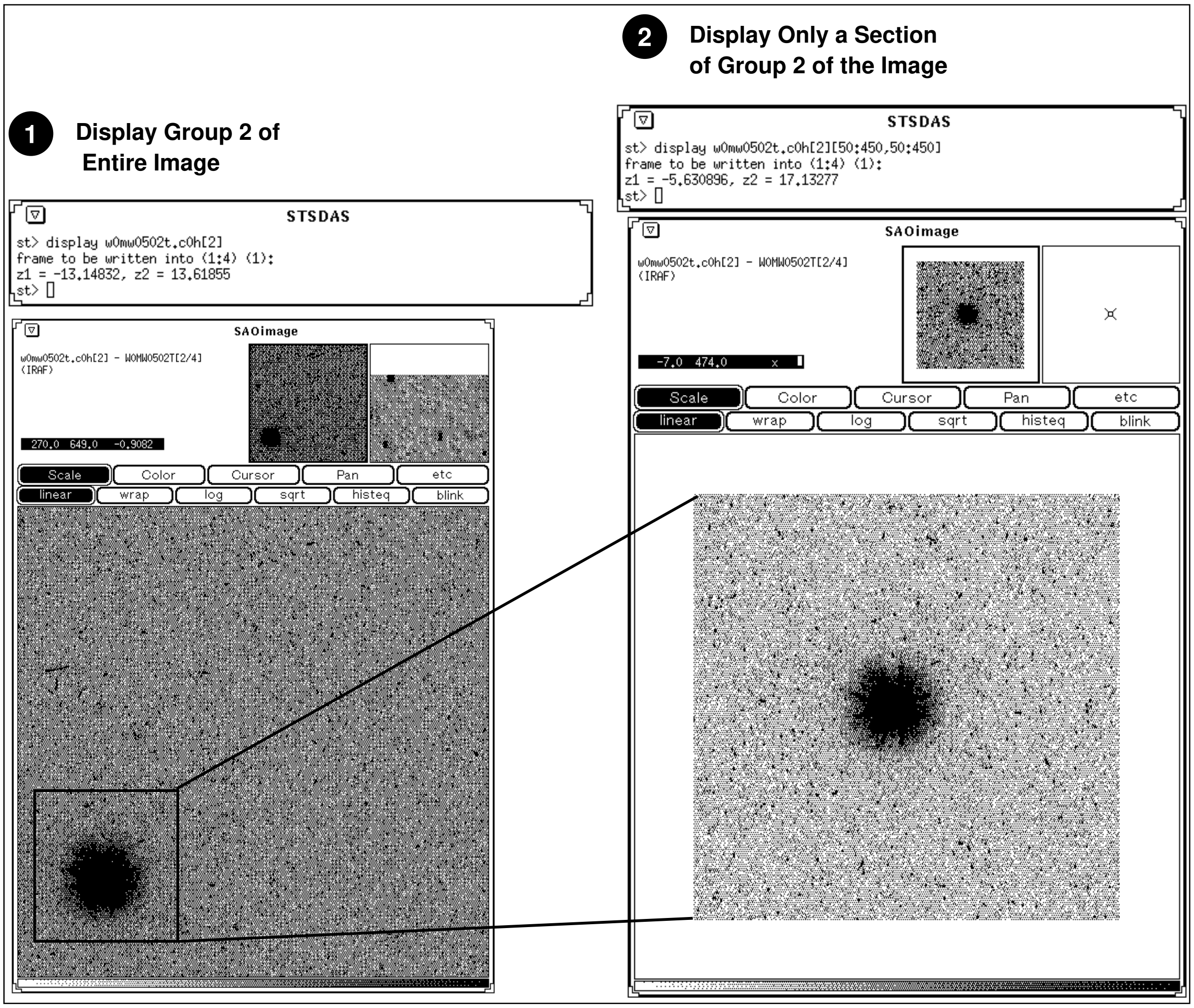

Figure 2.9: Displaying Sections and Groups of an Image

Tools for Working with Images

Both IRAF and STSDAS contain a large number of tasks that work with images. Some of the packages that you should investigate are:

- **images**: This package includes general tasks for copying (**imcopy**), moving (**imrename**), and deleting (**imdelete**) image files. These tasks operate on both the header and data portions of the image. The package also contains a number of general purpose tasks for operations such as rotating and combining images and doing image arithmetic.
- **stsdas.toolbox.imgtools**: This package includes general tools for working with multigroup GEIS images. This package includes tasks for working with pixel masks, and general purpose tasks for interactive pixel editing (**pixedit**) and arithmetic with multiple input images (**imcalc**).
- **stsdas.toolbox.headers**: This package includes general tasks for working with the header portion of an image. Tasks in this package can be used to edit header keywords (**eheader**, **chkeywrd**), display instrument-oriented information about the image (**iminfo**), and otherwise work with the image headers.

STSDAS Tables

Several of the analysis packages in STSDAS require an input file or create output files in STSDAS *table format*. This is a data format, distinct from IRAF or STSDAS image formats, designed for tabular (row and column oriented) data. Different columns may have different data types, but all the values in one column must have the same data type. A table may also contain header parameters identified by keywords. Tables may be binary or text (ASCII). While the binary format is more efficient, text tables may be used with host-level tools such as text editors. You may find STSDAS tables on your HST data tape since many datasets are shipped with the STSDAS-format tables used in the calibration process.

Tables can be viewed, created, printed, and manipulated using the tasks in the STSDAS **ttools** package (or in the external TABLES package). Tasks in **ttools** are listed below, based on their function.

- Create a table: **tcreate**, **tedit**
- Display contents: **tprint**, **tread**, **tedit**, **tcheck**, **tupar**, **gtedit**
- Modify contents: **tedit**, **tupar**, **gtedit**
- Information about a table: **tinfo**, **tlcol**

- Database-like utilities: **tquery**, **tsort**, **tproject**, **tselect**, **tdiffer**, **tjoin**, **tproduct**, **texpand**
- Statistics, etc: **thistogram**, **tlinear**, **trebin**, **tstat**
- Arithmetic: **tcalc**, **tintegrate**
- Convert between table or image header parameter, CL parameter, table datum: **keypar**, **keytab**, **parkey**, **partab**, **tabkey**, **tabpar**, **keyselect**
- Change column definition or table size: **tchcol**, **tchsize**
- Miscellaneous: **tcopy**, **trename**, **tdelete**

To simply read the contents of a table, you can use **tread** to display the file and move through it using the arrow keys. Use **tprint** to display the file in a formatted manner.

For more details, see “STSDAS Tables: the ttools Package” on page 68.

Networking

IRAF has the ability to transfer files, display images, and access devices such as tape drives, on one computer host from another in a local area network. IRAF networking can be used with almost any IRAF or STSDAS task because the networking capability is part of the Virtual Operating System (VOS), upon which all IRAF tasks are layered. *However*, file access between hosts with incompatible architectures (like between a Unix machine and VAXes running VMS) will *not* generally be possible unless the files are stored in a machine-independent format. (The exceptions are the tasks in the **stsdas.toolbox.convfile** package, which are specifically designed for transferring images and binary tables across machine architectures.) Note that OIF (`.imh`) and GEIS (`.hhh`) images and binary tables are stored in the native machine format, so accessing such images or tables *will not* work *unless* the local and remote hosts are of the same architecture, i.e., either both Unix or both VMS. FITS (`.fit`), pixel list (`.pl`), and ASCII files *are* machine independent, so numerical compatibility is not an issue in these cases.

To use IRAF networking, you must have a login account on each machine to which you intend to connect, and IRAF V2.10.2 or later (or at least the IRAF Kernel Server) must be installed on these machines. Your IRAF site manager must also create a set of files in the "dev$" directory which contain, among other things, the names of all the nodes on the network; these files must reside on all the servers in the network. To see if your machine is in this list, run the **netstatus** task. If your host is not listed, it must be added before you can use IRAF networking.

The Irafhosts File

To use the networking facility, you must have a file named ".irafhosts" in your login directory on *each* system or cluster (group of workstations sharing a common file system) to which you will connect. This file tells the IRAF networking system how to connect to the IRAF kernel server on a remote node. This file can be copied from the IRAF system directory (dev$). For example:

```
cl> copy dev$irafhosts ./.irafhosts
```

The contents of this file are machine dependent. For Suns and other Unix machines, it is:

```
port = default
auth = default
hiport = default
timeout = default
* <user> <user>           # host list
```

For VAXes, only the host list is used. For Unix systems, the networking parameters are set automatically by the networking system and will rarely need to be modified by the user. PORT may be set to "none" to prevent the use of networking daemons, or to an explicit TCP port number if the default value generated by the networking code cannot be used for some reason. AUTH is an authorization code used to prevent unauthorized access to your networking servers. The value is "default" initially but is filled in by the networking system when networking is first used. For maximum efficiency, AUTH should be the same on all hosts (after using networking for the first time copy the .irafhosts file to your other local hosts). Each user should have a unique AUTH value (the value provided by the system ensures this). HIPORT is used by the networking code to generate PORT. TIMEOUT is the time in seconds after which an idle networking daemon will shut down.

The syntax of the host list is:

```
hostname : loginname password
```

There should be one host per line, with the last hostname set to "`*`" (i.e, match any host). The loginname should be "`<user>`" (use your local login name), an explicit login name, or "`none`" to disable networking for that host. The password can be "`<user>`" (use your local password), and "`?`" to be queried for the password at run time. If an explicit password is given the REXEC protocol is used, otherwise RSH is used. Usually, you will get a prompt only once for each IRAF session per remote machine per task.

Do not put your password in the `.irafhosts` file; this is a glaring security risk.

VAX-Specific Details

Most users have a `LOGIN.COM` file in their root directories so that various system logicals, aliases, etc., are defined during the login procedure. However, you should realize that this command file will be executed for non-interactive sessions as well, including IRAF networking. Therefore, it is important to bypass those portions that would conflict with a network process. Users should place the following lines in their `LOGIN.COM` file:

```
! < first line of LOGIN.COM: >
$ IF F$MODE() .EQS. "OTHER" THEN GOTO OTHER
!: ! < contents of LOGIN.COM file > !:
$ OTHER:
! < set logicals for IRAF: >
$ @DISK$STSDAS:[IRAF.VMX.HLIB]IRAFUSER.COM
$ EXIT
```

System Dependent

Note the next to the last line above. This disk and pathname may be different for your site. Consult your site manager if you don't know the location of the `IRAFUSER.COM` file.

The VAX machine must be running MULTINET if you intend to access it from Unix machines. Network access via DECNET—i.e., to another VAX on the same cluster—is also possible, in which case you can substitute "`none`" for "`<user>`" in your host list to avoid being prompted for your password. That is, access by user "kilroy" to the node "AIRY" from "AVION" (if both are VAX machines) would be handled in your AVION host list as:

```
airy: kilroy none
```

Networking Syntax

When referring to files on remote machines, use the syntax *node*!*pathname*. The pathname for the remote host and directory can be referred to with IRAF environment variables. This is usually good practice (it helps to avoid typos), but it is often necessary for IRAF networking because wildcards in file names are *not* expanded if the path begins with a node name. Simply append the "$" character to the environment variable and use it in place of the "node!pathname" syntax.

The pathname is machine-specific, and would look something like the following to refer to a Unix machine (note the trailing "/"):

```
cl> set sd = sol!/sol/data1/kilroy/galaxies/
cl> dir sd$w*.??h
```

For a VAX machine, it might look like:

```
cl> set vd = avion!disk\$data1:\[kilroy.galaxies]
cl> dir vd$w*.??h
```

Note that special CL characters, such as "$" or "[", must be escaped with a backslash (\).

Examples

As mentioned, most IRAF and STSDAS tasks can be used in conjunction with IRAF networking. What follows are a few of the most common tasks that could make use of networking.

- Log in to a VAX host through a window on the local Sun "sol", display an image on the Sun machine that is stored (on the remote VAX host) in a GEIS-format file called `M51.HHH`. This assumes that the image display server (such as SAOimage) is running in a window of the Sun host. Note that the byte-order of the native host file is immaterial in this case, since the display buffer is of type "byte".

```
cl> reset node = sol.stsci.edu
cl> display m51.hhh 1
```

- Determine the status of the tape drive "mta" on the remote Sun machine "ra" from a local VAX machine.

```
cl> devstatus ra!mta
device ra!mta is not currently allocated
tape position for ra!mta is undefined
```

- Transfer all images beginning with the letter "v" from the remote Sun machine "sol" to the local directory on another Sun. The pixel files for the OIF-format (`.imh`) images will be created in the directory specified by `imdir`. Notice that the local directory is specified with a "." (i.e., a period).

```
cl> set sd = sol!/sol/data1/kilroy/galaxies/
cl> gcopy sd$v*.??h .
```

This will *not* work between a Sun and a VAX unless the files are in machine-independent format! (Use **sun2vax** and **vax2sun**).

- Transfer all images beginning with "`v`" from the local Sun host to the remote VAX host SCIVAX. Both GEIS (`.hhh`) and OIF images will be transferred, and pixel files for the `.imh` files will be created in the same directory as the header file (`HDR$`).

```
st> set dvax = scivax!disk\$data1:\[kilroy.galaxies]
st> sun2vax v*.??h dvax$
```

Troubleshooting Networking

Very often, when IRAF is unable to establish a connection to a remote machine, the message is obscure or incorrect. For example, if the `.irafhosts` file is missing or is incorrectly set up, IRAF networking may generate the message "`no files found`". It is generally a good idea to first use the **dir** command to be sure that IRAF can see the files in question, before trying to access them over the network.

CHAPTER 3: Data Formats

In This Chapter...

To properly analyze data from the Hubble Space Telescope it is best to understand the basic data formats that are used.[1] Both IRAF and STSDAS have headers containing descriptive information, although the implementations vary. Both also have separate files for headers and data. In this chapter, we will describe:

- IRAF data format (OIF)
- STSDAS data format (GEIS)
- How HST data sets are named and used for each instrument
- How to work with image sections and groups
- How world coordinates are handled in GEIS files
- STSDAS table format

1. This chapter is quite detailed and novice users may wish to skim this material, returning to it later after gaining some experience with STSDAS.

IRAF Images

The IRAF system's native format is known simply as the *IRAF* format, or *OIF* (Old IRAF Format). The IRAF format is conceptually quite simple, an image consists of two files:

- A *header file* containing various sorts of descriptive information. The header file always has an extension of `.imh`. The header is a combination of ASCII text and binary information.
- A separate *pixel file* containing pixel values. Thus, one "image" actually consists of two files. The pixel file always ends in an extension of `.pix`. The pixel file is completely binary.

The header (`.imh`) and pixel (`.pix`) files can either be in the same directory or in different directories. The header file contains a pointer to the name of the corresponding pixel file. The IRAF environment variable `imdir` is used to control the location of the pixel files for any newly-created images. On some systems `imdir` is set to point to a subdirectory of your IRAF home directory, or to a large scratch area elsewhere on the system.

Advantages and Problems with IRAF Images

The advantage of storing pixel files separate from header files is that the pixel files can be large, and don't clutter your main directory with large binary data files. Also, when you look at the contents of your IRAF directory, you see only the names of the header files, making the listing easier to read.

There are also some disadvantages to storing the pixel files separately:

- If you put the pixel files on a scratch area, the pixel data may get deleted while your header files remain.
- You can easily lose track of how much disk space is being used for pixel files because you do not see them as part of your directory listings—this can quickly cause disk space problems.

Where Pixel Files Get Stored

You can control the location of your IRAF pixel files by reassigning the value of `imdir` in your `login.cl` file (or at the CL prompt). For example, the following command would direct your pixel files to the subdirectory `pixfiles` under your IRAF home directory.

```
cl> set imdir = home$pixfiles/
```

Another option is to keep IRAF pixel files in the same directory as the header files. This is how STSDAS format data files are stored (as described below), and will help manage your data consistently. To do this, set `imdir` to the value `HDR$` (which means "same directory as header file"), as follows:

```
cl> set imdir = HDR$
```

Examining the Header

Note that neither an IRAF header nor pixel file can be printed using simple commands like **type** or **page**. To view an IRAF header file, you must use a task such as **imheader** in the IRAF **images** package.

The IRAF pixel file is not structured as a simple stream of data values. Depending on the length of the data axes, the file may have embedded null values that maximize I/O efficiency, even at the expense of having a larger file on disk. For example, look at the following long header listing for the standard IRAF test image `dev$pix`.

```
STSDAS
cl> imhead dev$pix long+
dev$pix[512,512][short]: m51  B  600s
    No bad pixels, no histogram, min=-1., max=19936.
    Line storage mode, physdim [512,512], length of user area 1540 s.u.
    Created Wed 08:51:57 08-Apr-92, Last modified Wed 08:51:57 08-Apr-92
    Pixel file 'HDR$pix.pix' [NO PIXEL FILE]
    'KPNO-IRAF'          /
    '08-04-92'           /
    IRAF-MAX=         1.993600E4  /  DATA MAX
    IRAF-MIN=        -1.000000E0  /  DATA MIN
    IRAF-BPX=                 16  /  DATA BITS/PIXEL
```

Figure 3.1: Partial Header Listing for IRAF Test Image

This listing indicates that this is a 512 x 512 image with short-integer pixel values. The third line tells you that the data is in line storage mode—each row, or line, of the image is stored sequentially in the binary pixel file. The pixel file is shown as having `physdim[512,512]`—that is, the data will map byte-for-byte to the binary file. (The warning `NO`

PIXEL FILE on the last line is irrelevant; this appears when you have imdir = HDR$.)

In contrast, look at the header for an image called temp which is extracted as a subset from dev$pix (Figure 3.2).

```
STSDAS
cl> imcopy dev$pix[1:490,1:50] temp
dev$pix[1:490,1:50] -> temp
cl> imhead temp long+
temp[490,50][short]: m51  B  600s
    No bad pixels, no histogram, min=unknown, max=unknown
    Line storage mode, physdim [512,50], length of user area 1783 s.u.
    Created Mon 12:01:24 27-Jul-92, Last modified Mon 12:01:25 27-Jul-92
    Pixel file 'lager.sts!/usr/tmp/stevens/temp.pix' [ok]
    'KPNO-IRAF'           /
    '08-04-92'            /
    New copy of dev$pix[1:490,1:50]
    IRAF-MAX=          1.993600E4  /  DATA MAX
    IRAF-MIN=         -1.000000E0  /  DATA MIN
    IRAF-BPX=                  16  /  DATA BITS/PIXEL
    IRAFTYPE= 'SHORT   '           /  PIXEL TYPE
```

Figure 3.2: Partial Listing of Partial Image Section

The listing indicates that the image is now of dimension 490 x 50, but on the third line it indicates that the physical storage space is 512 x 50. This is done to make I/O to an arbitrary row occur on a block boundary, which is more efficient. IRAF data files will often, as a result, have some wasted space in them. If you try to compute the size of a pixel file by just taking the product of the total number of pixels times the bytes per pixel, you will find that the actual size of the data file can be larger by 10 to 20 percent.

STSDAS Images

All HST data sets are stored in Generic Edited Information Set (*GEIS)* format. Like IRAF format, images are stored in two files:

- An ASCII header file with default file extensions of .hhh (although other extensions are used).
- A binary data file with default extension .hhd.

The two biggest differences between OIF and GEIS files, from the user's point of view, are:

- GEIS format can accommodate multiple images in a single file (each of these images is called a *group*)

- Some image descriptors are found in the binary data file, in an area called the *Group Parameter Block*, rather than in the header file.

Data Structure Overview

GEIS data files have a structure that is based on, but significantly different from, the FITS group format.[2] The GEIS group format was designed to accommodate data such as time-resolved spectroscopy, where many small data arrays share common header information, but also have a certain amount of array-specific information. For example, the FOS and HRS instruments aboard HST have observing modes for time-resolved spectroscopy which can produce hundreds of spectra, all of the same size (in pixels), and all for the same observation set. If these were each stored as separate files, the overhead in file size and management would be horrendous. These data are, therefore, combined into single observation sets in GEIS group format. In this case, the descriptors that are common for all the spectra (such as the size of each spectrum, the instrument configuration, etc.) are stored in the header file, but the group-specific descriptors (e.g., the offsets for the GIMP correction) would be stored with the binary data.

Parameters that are specific to a subset of the data are called *group parameters*, and they are stored in a reserved area of the binary data file called the *group parameter block*. Each group[3] of binary data has a reserved area for the group parameter block, allowing each group to have different values for group-dependent parameters. A representation of the structure is shown in Figure 3.3.

Note that in GEIS format the group parameter block *follows* each group of binary data, and that the group parameters can be any supported data type. But as with FITS groups, each image must be of the same data type.

2. See “An Extension of FITS for Groups of Small Arrays of Data,” by E.W. Greisen and R.H. Harten, 1981, *Astron. Astrophys. Suppl.*, 44, 371.

3. A single image of a group-format GEIS file is often referred to as a *group*. The potentially confusing terminology retains some historical baggage.

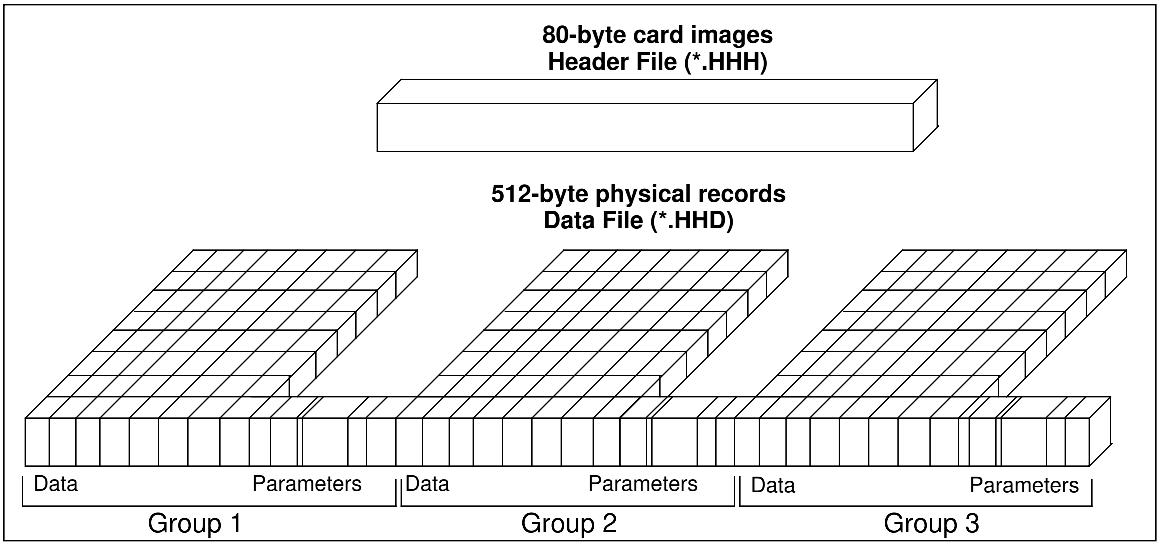

Figure 3.3: Structure of a STSDAS Group Format File

Image Descriptors—Header Keywords

Header files are composed solely of ASCII text, and can be printed using standard text-handling facilities (**type**, **page**, etc.). However, the lines in GEIS headers are always 80-character, fixed-length ASCII records. Any change you make in the text must not change the length of a line from 80 characters. The **images.hedit** task will preserve the 80-character line width. There are also tasks such as **chcalpar** (in the **ctools** package) which will change the calibration switches if you wish to recalibrate a dataset.

Always edit headers using tasks like **hedit**. Editing headers with a text editor may corrupt the files by creating incorrect line lengths.

Group Descriptors—Group Parameter Block

Because the actual values of group parameters are stored in the group parameter block with the binary data, you must use the tasks **images.hedit** or **toolbox.headers.groupmod**. Similarly, you cannot view the values of group parameters simply by looking at the ASCII text header. Rather, you must use a header listing tool such as **images.imheader**, which will list the full set of header parameters, including both the main header and the group parameter block. For example, the first several lines of the file `hrs.hhh` might appear as shown in Figure 3.4.

```
STSDAS
st> imheader hrs.hhh long+
hrs.hhh[500][real]: HRS[1/1]
    No bad pixels, no histogram, min=4.153793E-4, max=64826.
    Line storage mode, physdim [500], length of user area 5549 s.u.
    Created Mon 20:47:15 13-Apr-92, Last modified Wed 11:30:42 11-Dec-91
    Pixel file 'hrs.hhd' [ok]
    GROUPS  =                    T
    GCOUNT  =                    1
    PCOUNT  =                   16
    PSIZE   =                  864
    CRVAL1  =             2392098.
    CRVAL2  =                  -1.
    CRVAL3  =                  -1.
    CRPIX1  =                   1.
    CD1_1   =                   1.
    DATAMIN =          4.153793E-4
    DATAMAX =               64826.
    FILLCNT =                    0
    ERRCNT  =                    0
    PKTTIME = ' 2-MAY-1988 11:03:32.25 '
    CTYPE1  = 'CHANNEL '
    CTYPE2  = 'RA___TAN'
    CTYPE3  = 'DEC__TAN'
    OBSRPT  =                    0
```

Figure 3.4: Header Listing from imheader Task

Note that the six group parameters are listed as part of the normal header, by name (`DATAMIN`, `DATAMAX`, `CRPIX1`, `CRVAL1`, `CTYPE1`, `CD1_1`), and their values have been extracted from the binary data for the listing. It is not immediately obvious which header parameters are from the main ASCII header, and which are from the group parameter block, but from the user's (and programmer's) point of view, it does not matter. If you need to know the origin of a parameter, just search the file header for the string "PTYPE", and you will find the names of all group parameters as shown in Figure 3.5. For each group parameter there is a corresponding keyword `PDTYPE`*n* (where *n* is the parameter number concatenated to `PTYPE`) which gives the data type of the parameter and `PSIZE`*n* which gives its size (in bits). `PCOUNT` gives the total number of group parameters. `GCOUNT` gives the total number of groups in the binary file.

```
STSDAS
st> match PTYPE hrs.hhh
PTYPE1  = 'CRVAL1  '          /
PTYPE2  = 'CRVAL2  '          /
PTYPE3  = 'CRVAL3  '          /
PTYPE4  = 'CRPIX1  '          /
PTYPE5  = 'CD1_1   '          /
PTYPE6  = 'DATAMIN '          /
PTYPE7  = 'DATAMAX '          /
PTYPE8  = 'FILLCNT '          /
PTYPE9  = 'ERRCNT  '          /
PTYPE10 = 'PKTTIME '          /
PTYPE11 = 'CTYPE1  '          /
PTYPE12 = 'CTYPE2  '          /
PTYPE13 = 'CTYPE3  '          /
PTYPE14 = 'OBSRPT  '          /
PTYPE15 = 'OBSINT  '          /
PTYPE16 = 'YDEF    '          /
st>
```

Figure 3.5: Finding Group Parameters

Disk Binary Formats—Physical Implementation

The binary data is stored in the host computer's natural format for all supported data types. There is no conversion to IEEE floating point format on systems such as VMS where the natural floating point format is not IEEE. The physical structure of GEIS files is machine dependent. On VAX/VMS systems, a GEIS file header is a sequential file consisting of fixed-length 80-byte records. The binary data are written as VAX reals, VAX integers (i.e., byte-swapped, or little-Endian format), or whatever the default machine format is for the given data type.

The Unix file system is much simpler than the VMS file system, and all files are treated as byte streams. The GEIS header file is still composed of 80-character card images, with each card image terminated by a "\n" (newline). The binary data is a byte stream, just as on VMS, but with the data written in the natural binary format of the host system.

On most Unix systems the floating point format is the IEEE floating point standard and data are not byte-swapped, as they are on the VAX. The byte order is big-Endian. On the DECStation family (based on the MIPS RISC chip) the floating point format is IEEE, but all data are byte-swapped (little Endian).

STSDAS data files may contain data in any standard Fortran data type, as well as several non-standard types. The allowed data types are shown in Table 3.1.

Data Type	*Meaning*
`INTEGER*2`	Short integer
`UNSIGNED*2`	
`INTEGER*4`	Long integer
`REAL*4`	Single-precision floating point
`REAL*8`	Double-precision floating point
`COMPLEX*8`	Complex

Table 3.1: Data Types Allowed in STSDAS Files

In practice, HST observers are likely to only encounter short integer and single-precision real data. IRAF and STSDAS programs dynamically determine the data type of an input file, and will convert data into whatever internal representation is needed.

Working With Multigroup Images

There will likely be times when you want to work on only a part of an entire image. In this section we will describe and present examples showing how to work with:

- Group syntax to specify a particular group
- Image section syntax to specify an area of an image

Omitting File Extensions

Applications tasks in can generally figure out what file you want to process without your typing the file name extension, at least if you are using STSDAS files with the standard default extensions of `.hhh` and `.hhd`. In such cases, you can drop the explicit reference to `.hhh` or `.imh` when specifying the name of the input image. If the input STSDAS file has an extension other than `.hhh` and `.hhd`, it must be specified. See the rules for STF file name extensions on page 48.

Specifying Groups

Because GEIS files can contain multiple images (each stored in a *group*), you will often need to specify a particular group upon which an application program should operate. In fact, if you do *not* specify a group, most IRAF and STSDAS tasks will operate *only* on the first group.

To specify a group, enclose the group number in square brackets. For example, if the file `mydata.hhh` were a 10-group file, the command:

```
cl> imarith mydata.hhh[2] + 2. output.hhh
```

would add 2 to the values in group 2 of the file `mydata.hhh`, writing the results to a new file called `output.hhh`, which would be a single-group file.

The output from any operation on a single group of a multigroup GEIS file is a single group output file. You can, however, create multigroup output files the *first time* the new image is written, for example, look at the first line of the following commands:

```
cl> imcopy mydata.hhh[2] newimage.hhh[3/4]
cl> imcopy mydata.hhh[3] newimage.hhh[4]
```

The first statement copies group 2 of the input image to group 3 of the output image, and creates the output image with enough storage space to accommodate 4 groups. The other 3 groups will be empty, but you could copy other data into them as long as the image dimensions match. Note that the second time the output file is used, the new total group count (4) is not repeated. If you were to repeat it, IRAF would think you wanted to create an entirely new file, rather than updating a group in an existing file, and would warn that you were trying to overwrite an existing file. Similarly, other standard tasks can be used to create multi-group output files, for example:

```
cl> imarith mydata.hhh[10] * 2. newimage.hhh[2/5]
```

There is currently no automatic way to have an IRAF or STSDAS task process all groups in an input image and write the results to corresponding groups in an output image. However, there are several STSDAS tasks, particularly in the **toolbox.imgtools**, **fitsio**, and **hst_calib.ctools** packages, that simplify using group format data. Also, the pipeline calibration tasks explicitly process all groups in a GEIS file. For example, the **gcopy** task in the **imgtools** package will copy all of the groups to another GEIS image:

```
cl> gcopy mydata.hhh newimage.hhh
```

Specifying Image Sections

There are also likely to be times when you do not want to work with an entire image, perhaps because you want to limit a task to a particular area of interest or because you want to exclude suspect regions from affecting your analysis. You can limit the area affected by a task using *image section* syntax. For example, the following command will add 2.0 to a 100 × 100 one-group pixel subimage of the second group of `mydata.hhh` and store the result in a new 100 × 100 image called `output.hhh`.

```
cl> imarith mydata.hhh[2][101:200,101:200] + 2. output.hhh
```

If you specify both a group number and an image section, the group number must *precede* the image section. (See “Image Sections and Groups” on page 28).

Image Types

File formats can be controlled by the environment variable `imtype`. When a new data file is created (for example, by a FITS reading task), `imtype` can be used to define the format of the newly-created file without giving an explicit file name extension. To create STSDAS format files by default, use the command:

```
cl> set imtype = hhh
```

To create IRAF format files by default, use:

```
cl> set imtype = imh
```

The value of `imtype` has no effect when an image is created as a copy of an existing image.

Any application in IRAF or STSDAS can read a file in one format and write an output file in the other format. By default, the output file is assumed to use the same format as the input file (IRAF format for IRAF input files, STSDAS format for STSDAS input files). This can be overridden by giving the output file name a different extension, for example:

```
cl> imcopy data.hhh output.imh # copy file to IRAF format
cl> imcopy output.imh newdata.hhh # copy to STSDAS format
```

Remember that commands that do not specify a group (like those above) usually work *only* on the first group in the image.

HST Datasets

All HST datasets use multigroup GEIS format but the file names differ by instrument (they will not be .hhh and .hhd when you get a dataset). There are several absolute rules about STSDAS file names:

- Extensions must be three characters long.
- The third character of the header file extension must be “h”.
- The third character of the data file extension must be “d”.
- The root part of the file name must be identical for both the header and data parts of the image.
- The extensions must be identical except for the “h” and “d” characters in the third position.

The following are valid STSDAS data file names:

```
image.hhh   image.hhd
data.abh    data.abd
fosimg.d0h  fosimg.d0d
```

The following, however, are *not* valid STSDAS image names:

```
image.hhh   image.ddd   # extensions are different
data.abh    data.dat    # extensions totally different
fosimg.d0h  FOSimg.d0d  # root names different
```

Don’t forget that with IRAF’s virtual file name conventions, file names are case sensitive; this is what invalidates the last example.

Actual data from the RSDP pipeline follows a more restricted set of rules:

- File root names are constructed from five character fields and have a total of 9 characters. In generic terms a file is known as `IPPPSSOOT`. This generic file name can be decoded by substituting the characters in the file name with the values in Table 3.2.

Character	*Meaning*	*Replacement Values*
`I`	Instrument Code	F - FGS U - WFPC-2 V - HSP W - WFPC X - FOC Y - FOS Z - HRS
`PPP`	Program ID	Any combination of letters and numbers
`SS`	Observation Set ID	Any combination of letters and numbers
`OO`	Observation ID	Any combination of letters and numbers
`T`	Origin of data	T - Dumped to onboard tape recorder[a] R - Real time[a] M - Merged real time and tape recorded N - Retransmitted merged O - Retransmitted real time P - Retransmitted tape recorded

a. “T” and “R” are most common.

Table 3.2: Interpretation of STSDAS Root File Names

- File extension names are constructed of two fields and are of the form `XXT`. This generic file name can be decoded by substituting the characters in the file name with the values in Table 3.3.

Character	*Meaning*	*Replacement Values*
`XX`	Type of Data	TR - Trailer File SH - Standard Header Packet UL - Unique Data Log D*n* - Raw Data (n is a number from 0-9) C*n* - Calibrated Data Q*n* - Quality File A*n* - Astrometry Data
`T`	File Type	H - ASCII Header D - Binary Data

Table 3.3: Interpretation of STSDAS File Name Extensions

Other conventions also apply, but these cover most standard science data. (See *Interface Control Document 19: PODPS to STSDAS*, Section 6, for details.)

A typical output file from the calibration pipeline might, for example, be called `w09h10a0t.d0h` and `w09h10a0t.d0d`. This file can be identified as a WF/PC data file, with program ID `09H`, observation set ID `10`, observation ID `A0`, originating from the on-board tape recorder. It is uncalibrated science data.

Obviously, HST data files names are going to be awkward to type and will take some practice to identify. The task **iminfo** in the STSDAS **tools** package will decode a file name and print out the most relevant information from the file header (tailored for each instrument).

Instrument-Specific Data Structures

The data from each instrument are stored in a number of separate files—there are files for raw data, calibrated data, data quality, standard header packets, unique data logs, and trailer files.[4] These files are named according to the conventions described on page 48; the raw and calibrated data file structures are described on page 40. The structure of data quality

4. See *Interface Control Document 19: PODPS to STSDAS* for details.

files is similar to the science data, except that the data values have a rather restricted range: zeroes and ones for some instruments, or specific flag values indicating the types of pathologies that might affect a given pixel in the associated data files.

The standard header packets (SHP) and unique data logs (UDL) contain information related to telemetry, instrument command, spacecraft orbit and pointing, and other instrument monitoring measurements. Most of the useful information in these files appears in the file header, in FITS style `KEYWORD = VALUE` notation. Binary data can only be interpreted as bit-streams; one needs to know the start and stop bits for a particular piece of information. Users ordinarily do not need to extract information from binary data in the SHP or UDL files because a more readable copy of the data exists in the header. Trailer files are a log of processing steps and any anomalies noted during routine calibration. These files are in plain text format.

The Wide Field/Planetary Camera has eight CCD detectors—four of which are used in PC mode, four in WFC mode—that create four images. These four images are stored as four groups in an STSDAS data file. A particular observation may not use all four CCDs, so fewer groups may appear in a WF/PC data file. The group parameter block for a WF/PC science data file has 37 group parameters and includes information about the image (minimum and maximum values, coordinate system), the data quality (numbers of defective pixels), and photometry.

Like WF/PC, the WF/PC-2 has four CCD detectors that create four images, each of which are stored as a group in an STSDAS image. A given observation may not use all four detectors so there may be fewer than four groups in a WF/PC image. The group parameter block has 49 parameters that are used for information about the image (minimum and maximum values, coordinate system, etc.), the data quality (numbers of defective pixels), image statistics, and photometry.

Faint-Object Camera datasets have only one group. The group parameter block has 20 parameters that are used for image-related information such as minimum and maximum values and the coordinate system.

High Speed Photometer files also have only one group. However, HSP data sets can be very large (such as when an observation is taken using the highest possible time resolution over half an orbit) and may be broken up into more manageable parts. The group parameter block has 14 parameters, including the usual image descriptors.

Faint Object Spectrograph and High Resolution Spectrograph data files have many groups—thousands in their time-resolved modes. The HRS group parameter block has 27 parameters, and the FOS has 19 group parameters.

Note that groups are used differently for each instrument. WF/PC uses a group for each detector, FOC and HSP use only a single group for an observation, and FOS and GHRS use dozens (or hundreds) of groups corresponding to each instance of data being read out.

Data from the Fine Guidance Sensors does not get handled in the same way as data from the other science instruments. Because the FGS is considered part of the engineering system on HST, the data from FGS comes down as part of the engineering data stream, not part of the science data stream. The Astrometry and Engineering Data Processing System (AEDP) reformats the FGS data for scientific analysis.

Astrometry Data

Astrometry data from the Fine Guidance Sensors is also available in STSDAS GEIS format, but the structure of this data is more complicated than that of the science instruments. Also, astrometry data are not formatted or processed in the routine data processing pipeline, but rather are extracted from the telescope engineering data stream by a system called AEDP (Astrometry and Engineering Data Processing).

Each astrometry science data set consists of three sets of files (header and binary data)—one for each of the three Fine Guidance Sensors. These files are named much like other science data, with a root file name that follows the same guidelines described in the previous section (and defined in Table 3.2). The file name extensions are of the form `.AnH` and `.AnD`, where `n` is the digit 1, 2, or 3 and corresponds to the FGS number. Thus, a typical astrometry data set might consist of files like the following:

```
f055030em.a1h  f055030em.a1d
f055030em.a2h  f055030em.a2d
f055030em.a3h  f055030em.a3d
```

Each file contains a total of 17 groups, or subsets, these groups are described in Table 3.4.

Group Number	*Contents*
1-4	Photomultiplier counts (X-A, X-B, Y-A, Y-B)
5-6	Star-sensor encoder readings (A,B)
7-8	Fine-error signal averages (X,Y)
9-10	Star-sensor averages (A,B)
11-12	Absolute magnitudes (X,Y)
13	The age-word (used to determine timing)
14	Empty: not currently used
15-16	Star position coordinates
17	Data status flags

Table 3.4: Groups in Astrometry Files

It should be apparent that the groups in an FGS data file do not represent the same type of data, and there will not necessarily be the same number of valid data points in each group (by definition each group has the same number of values, but each group can be filled out to the total group length with fill values). The `AGEWORD` group is used to properly assign the correct time of observation to each data value (the STSDAS task **puttime**, in the **stlocal.testfgs** package, computes observation times); this is necessary because various quantities in the FGS data file are sampled at different rates.

The headers of the FGS data files contain lots of other information about the observation, and users should consult *Interface Control Document 46—AEDP to PODPS* for more detailed information.

World Coordinates

World Coordinate Systems (WCS) are any coordinate systems that describe the physical coordinates associated with a data array, such as sky coordinates (RA and Dec, galactic latitude and longitude, etc.) for an astronomical image, wavelength scale for a spectrum, or time scale for a time series. The IRAF system is able to manage world coordinate systems for both IRAF OIF and STSDAS STF format data files. This means that operations such as image sections take proper actions to update the world coordinate system information in the output image header—and do so for both OIF and STF data. However, the applications software in IRAF does not deal uniformly with the WCS information, and users are likely to discover that the coordinates written by an STSDAS application are not properly interpreted by an IRAF application, and vice versa. We hope to correct this in future versions of IRAF and STSDAS.

STSDAS data files contain information in their headers (usually in the group parameter block) which describes a mapping between pixel coordinates and world coordinates. For example, a spectral dispersion relation can be given, or the mapping between (X,Y) and (α, δ) can be specified.

The STSDAS conventions for specifying the world coordinate system are similar to the FITS conventions, but differ in several significant ways. Actually, papers describing the FITS format do not fully define how a WCS is to be specified, but some conventions have come into practical use.

The mapping between pixel coordinates and world coordinates is defined by the keywords described in Table 3.5.

Keyword	*Contents*
`CTYPEi`	Type of Coordinate on Axis *i*
`CRPIXi`	Reference Pixel on Axis *i*
`CRVALi`	Value of the World Coordinate at Axis i
`CDi_j`	A Matrix of Partial Derivatives of the World Coordinates with Respect to the Pixel Coordinates

Table 3.5: Keywords Defining Mapping Between Pixel and World Coordinates

The keywords `CDi_j` (CD stands for Coordinate Description) are used instead of the FITS keywords `CDELTi` and `CROTAi` because of the ambiguity in the FITS keyword: the sense of the rotation has not been clearly or uniformly defined. CDELT*i* and CROTA*i* cannot unambiguously define a non-orthogonal coordinate system.

CTYPE*i* is used to define the type of coordinate. In STSDAS data, the values assigned to CTYPE*i* are typically RA---TAN, DEC--TAN (equatorial coordinates with a tangent plane projection), TIME, CHANNEL, LAMBDA (a wavelength dispersion relation), and PIXEL. The value of CTYPE*i* defines both a label for the axes and, for sky coordinates, a projection geometry to use in converting (X,Y) to (α,δ). The other values of CTYPE*i* denote linear transformations.

CRPIX*i* and CRVAL*i* are interrelated. CRVAL*i* is the value of the world coordinate at pixel CRPIX*i*. For linear world coordinate systems, CRVAL*i* and CRPIX*i* can both be modified, as long as the changes are self-consistent. In sky projection geometries, however, CRVAL*i* denotes the point of tangency and cannot be changed without resampling the image. When an image section is extracted from an image, the values of CRPIX*i* are updated to reflect the change in the pixel numbering. CRPIX*i* may have any value—the reference pixel can lie outside the image (CRPIX*i* < 0 or CRPIX*i* > NAXISi) and can have a fractional value.

The CD matrix is defined by the partial derivatives of the world coordinates with respect to the pixel coordinates as evaluated at the reference pixel. For linear coordinate systems this distinction is not significant, but for non-linear systems such as a tangent plane projection the distinction is critical. For example, for a linear wavelength dispersion relation we could have something like the following (if NAXIS1 were, say, 1024, these keywords would describe a spectrum with wavelengths running from 4000.0 to 5258.29 corresponding to pixels 1 through 1024):

```
CTYPE1 = 'LAMBDA' ! wavelengths
CRPIX1 = 1.0      ! reference pixel
CRVAL1 = 4000.0   ! wavelength is 4000.0 at pixel 1.0
CD1_1  = 1.23     ! wavelength increment is 1.23 per pixel
```

For an image of the sky, the coordinate system representation might be something like Figure 3.6.

```
CTYPE1 = 'RA---TAN'    ! r.a. in tangent plane projection
CTYPE2 = 'DEC--TAN'    ! dec. in tangent plane projection
CRPIX1 = 400.0         ! reference pixel on first axis
CRPIX2 = 400.0         ! reference pixel on second axis
CRVAL1 = 180.01234     ! r.a. at pixel (400,400) in degrees
CRVAL2 = 30.98765      ! dec. at pixel (400,400) in degrees
CD1_1  = -2.777778E-4 ! change in RA per pixel along first
                       ! axis evaluated at reference pixel
CD1_2 = 0.0            ! change in RA per pixel along second
                       ! axis evaluated at reference pixel
CD2_1 = 0.0            ! change in dec per pixel along first
                       !axis evaluated at reference pixel
CD2_2 = 2.777778E-4    ! change in dec per pixel along second
                       ! axis evaluated at reference pixel
```

Figure 3.6: Coordinate System Representation

The values of the CD matrix are also given in floating point degrees (the example shown indicates a pixel size of 1 arcsecond square). `CD1_1` is negative because right ascension decreases with increasing pixel number in a normally oriented astronomical image.

A common definition of the CD matrix is as follows:

$$\mathrm{CD1\_1} = \frac{\partial l}{\partial x} = -\Delta_1 \cos\theta$$

$$\mathrm{CD1\_2} = \frac{\partial l}{\partial y} = -\Delta_2 \sin\theta$$

$$\mathrm{CD2\_1} = \frac{\partial m}{\partial x} = -\Delta_1 \sin\theta$$

$$\mathrm{CD2\_2} = \frac{\partial m}{\partial y} = \Delta_2 \cos\theta$$

where Δ_1 and Δ_2 are the pixel sizes along the X and Y axes, respectively. From the more general form, it is easy to see that a rotation of the coordinate system is shown in the appearance of non-zero cross-terms in the CD matrix.

Please note that the first pixel in an image is numbered (1,1), and whole pixel numbers correspond to the center of the pixel.

Table Format

In addition to the image data formats described above, STSDAS supports a table format. The table format is especially useful for storing the output of certain data processing tasks (like a list of star positions and magnitudes from **daophot**) and for exchanging data between different applications packages. Tasks that create, edit, and perform other functions with tables are found in the STSDAS **ttools** package (see "STSDAS Tables: the ttools Package" on page 68).

STSDAS tables are stored in a binary format, and table data is a mix of character string, integer, short integer, boolean, and single and double-precision floating point. Each column must be of the same predefined format, and each row must contain the same number of columns. Figure 3.7 shows how a STSDAS table is structured.

Each table column has four attributes:

- ***Name:*** String of up to 19 characters, case insensitive. Similar names differing only in the use of upper or lower case are indistinguishable.
- ***Data type:*** Real, double, integer, boolean, or text.
- ***Display format:*** The specification for printing the contents of the column, used only for printed listings. Note that this is related to but distinct from the internal data type. That is, a double-precision column may be printed with limited precision although any computations using that column would be carried out in full precision.
- ***Units:*** A string of up to 19 characters. May be used by some tasks, for example, to distinguish between right ascension in hours or degrees. The default is null.

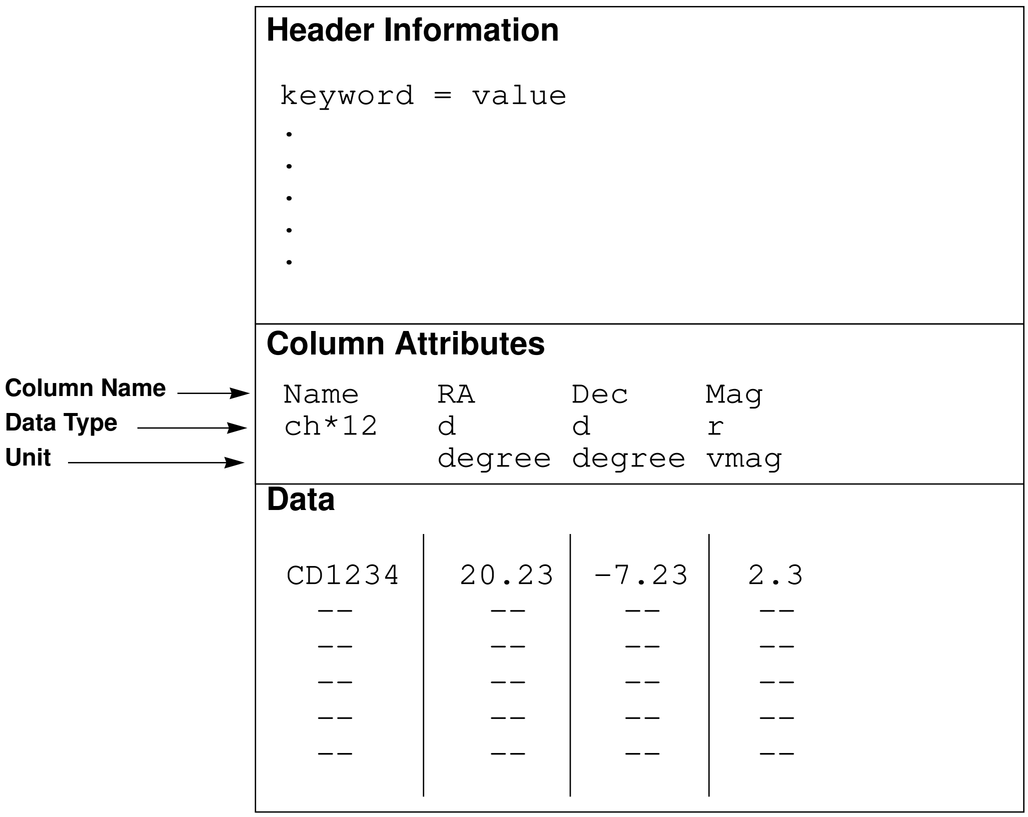

Figure 3.7: STSDAS Table Structure

CHAPTER 4: General Utilities

In This Chapter...

Many of the tasks in STSDAS were designed to meet a wide range of astronomical data analysis needs, and can be used on essentially any astronomical data. Other tasks, while being general, are auxiliary in nature: converting data from one format to another, creating or manipulating tabular data, editing images or image headers, displaying graphics and images, etc. Most of these auxiliary tasks are found within the **fitsio**, **ttools**, and **tbplot** packages, which comprise the TABLES external package. The TABLES package is bundled separately from STSDAS to make it easier for other IRAF external packages (like NOAO, XRAY, and EUV) to make use of these utilities without requiring that STSDAS be installed. However, STSDAS users will find **fitsio** and **ttools** listed in the STSDAS package tree as well: these are simply pointers back to the TABLES package, and are merely provided for convenience. STSDAS users need not worry about whether any given package is part of TABLES, since the TABLES package is loaded implicitly when loading STSDAS.

This chapter provides an overview of the general facilities provided in TABLES and STSDAS. The first two sections, on **fitsio** and **ttools**, describe tasks found within both TABLES and STSDAS; parts of the third section on graphics and plotting are also relevant for the TABLES package. The remainder of this chapter deals largely with HST-specific utilities which are only found in packages under STSDAS.

FITS Input/Output in fitsio

This section describes, in general terms, the FITS format and how it is used by tasks in the **fitsio** package. The last part of this section (beginning on page 62) provides basic tutorial-type information about how to read and write FITS files and tapes using **fitsio** tasks.

The FITS Format

The standard format for exchange of astronomical data is the FITS (Flexible Image Transport System) format, which is defined in a series of papers published in *Astron. Astrophys. Suppl.*, beginning in 1981.[1] FITS format is endorsed by the International Astronomical Union, and is being adopted as a NASA standard. The NASA Astrophysics Division, through the NASA Office of Standards and Technology (NOST), has adopted FITS "as a standard for the distribution of processed astronomical data from NASA Astrophysics missions and archives managed by or for the NASA Astrophysics Division."

The relationship between STSDAS image data, STSDAS tables, and the corresponding FITS data structures, is discussed in Chapter 3 on page 37.

FITS Capabilities in STSDAS/TABLES

The **fitsio** package in STSDAS or TABLES is used to convert STSDAS (and IRAF) images to and from FITS format.[2] It also converts ASCII or STSDAS binary tables to and from FITS format. The **stwfits** task lets you create one or more FITS files from a GEIS file, retaining the information in all groups together with the group parameter block. In **stwfits** there is a parameter called `gftoxdim` which, when set, converts the multigroup file to a FITS file that has an extra dimension to accomodate multiple groups; it appends a FITS TABLE extension that contains the group parameter block. In general, **stwfits** lets you create FITS extensions like TABLE, BINTABLE, and IMAGE.

1. See page 208 for a listing of documentation describing the FITS format.

2. Users should be familiar with STSDAS data formats (described in Chapter 3) before using the **fitsio** package.

It is also possible to produce a separate FITS file for every group in the multigroup GEIS source file. In this case, no table is appended; instead, group information is contained in the FITS header. If you want a separate FITS file for every group, then set the parameter `gftoxdim = no`.

A new way to produce a FITS file from a GEIS file is to use the `extension` parameter. This will produce one FITS file with an IMAGE extension from multiple input images. For more details, see the online help by typing `help stwfits`.

The **strfits** task can be used to convert a FITS file to a multigroup GEIS file, while retaining its group structure. If the parameter `xdimtogf` is set to `no`, then two files will be created: an image file having an extra dimension for the groups, and a table containing the group parameter block. This requires that you have a FITS file created with **stwfits** and the parameter `gftoxdim = yes`. To recreate the original multigroup GEIS file, set the parameter `xdimtogf = yes`.

The default group parameter block (GPB), which is located at the end of each group in the GEIS data file, contains the information shown in Table 4.1. These values are taken from the header file which, for each parameter (*n*), contains the keywords PTYPE*n*, PDTYPE*n*, and PSIZE*n*. PTYPE*n* is the parameter name, PDTYPE*n* is its data type, and PSIZE*n* is its size in bits.

Keyword	***Contents***
DATAMIN	Minimum pixel value in group
DATAMAX	Maximum pixel value in group
CRPIX*n*	Location of the reference pixel for the *n*th axis
CRVAL*n*	Value of the *n*th world coordinate at the reference pixel
CD*m_n*	Matrix describing scale and orientation of the axes

Table 4.1: FITS Default Group Parameter Block

If the FITS file does not follow the default standard, then a template file with the group parameter structure is needed to convert the FITS file to a GEIS file. The template file is a GEIS header file containing the keywords PTYPE*n*, PDTYPE*n*, and PSIZE*n* for each group parameter, plus the keywords PCOUNT and PSIZE. The file name of this image header can be passed to the **strfits** task using the `template` parameter.

The **stwfits** task offers the following capabilities:

- Writes multigroup images in stack mode. (Groups are converted to an extra dimension.)
- Writes multigroup images as one FITS IMAGE extension per group.
- Produces standard FITS TABLE or BINTABLE extension from STSDAS tables.
- Writes FITS data in FITS IEEE standard.
- Writes a FITS file from an ASCII file.

strfits offers the following capabilities:

- Reads all FITS files produced by **stwfits**.
- Produces STSDAS images with a user-defined group parameter block.

Scaling Data

The original FITS output standard calls for all floating point data types to be converted to 32-bit integer data. Scaling is done by **stwfits** when setting the parameter `scale=yes` and is automatic when `autoscale=yes`, i.e., the values of `bscale` and `bzero` are determined by the program. For data having a large dynamic range, this scaling loses precision when FITS files are recovered to disk images. However, the IEEE FITS standard allows both single and double precision data to be stored in floating-point format so that no precision is lost. To use the new IEEE FITS standard, set the `ieee` parameter to "yes" in the **stwfits** task.

Tutorial: Reading and Writing FITS Files

This section describes basic techniques for handling FITS-format data tapes, especially those with HST image data. Topics in this section include:

- The basic process of using tape drives within IRAF.
- How to read HST data tapes using the STSDAS **strfits** task (page 64).
- How to reconstruct a GEIS image if your tape contains several files with groups of an image in separate files (page 65).
- How to write HST data to tape using **stwfits** (page 66).

Using Tape Drives

Before reading or writing a tape, you must:

1 • Find out what the IRAF alias is for the tape drive you want to use (use the **devices** task).

2 • Physically mount (load) the data tape on the tape drive.

3 • Allocate the tape drive.

The IRAF **devices** command will list the available tape drives on your system along with their IRAF aliases. You can then use the **devstatus** command to see if somebody else is using the tape drive. Once you know the IRAF name of an available drive, you should mount your tape on the drive and then allocate it using the **allocate** command. When you are finished using the tape drive, you should release it using the **deallocate** command. Figure 4.1 shows an example of how to use these three commands.

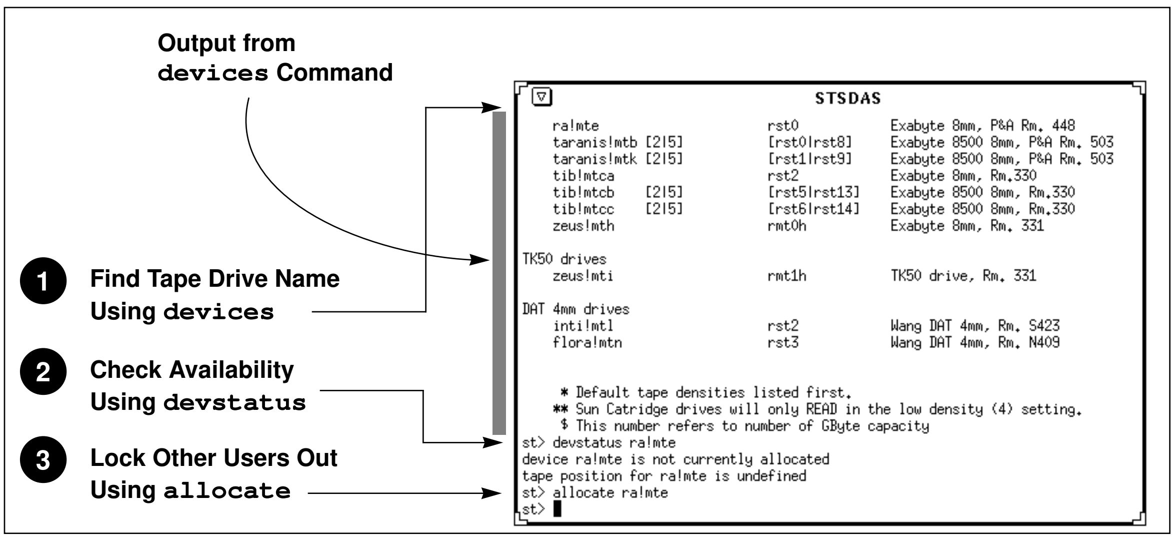

Figure 4.1: Finding and Allocating a Tape Drive

If you are not actually working from the host console to which the tape drive is attached, you must use the IRAF networking syntax, for example, `ra!mta` to indicate that the drive is attached to host ra.

Reading FITS Tapes and Files

To read an HST FITS-format data tape, you must:

1 • Allocate a drive and mount your tape as described above.

2 • Set the `imtype` environment variable which defines the type of image that you will be using.

3 • Set the **strfits** parameters.

4 • Read the tape using **strfits**.

5 • Dismount and deallocate the tape drive.

The `imtype` environment variable specifies the type of image that you are reading from tape, you will usually set this to "`hhh`", which specifies multigroup GEIS format.[3] For example:

```
fi> set imtype="hhh"
```

Always use the STSDAS **strfits** task when reading HST data sets. The STScI pipeline uses a special convention for mapping GEIS files to FITS format. While other FITS readers may be able to read some of the data, they are unlikely to properly reconstruct the entire data file.

Type `epar strfits` to edit the **strfits** parameters as shown in Figure 4.2. The parameters you should verify and set (if needed) are:

- `fits_file` - The IRAF name of your tape drive.
- `file_list` - Specification of the file numbers you want read off the tape. If you don't know what is on the tape, use the **catfits** task to display a catalog listing for the tape.
- `xdimtogf` - Be sure to set this to "yes" to specify that you are reading multigroup GEIS files that have been stored as a single FITS image with one extra dimension. (The default is "yes".)
- `oldirafname` - Set this to "yes" to restore the original file names.

3. You can always override the value of `imtype` by specifying the extension when you set your file name parameter.

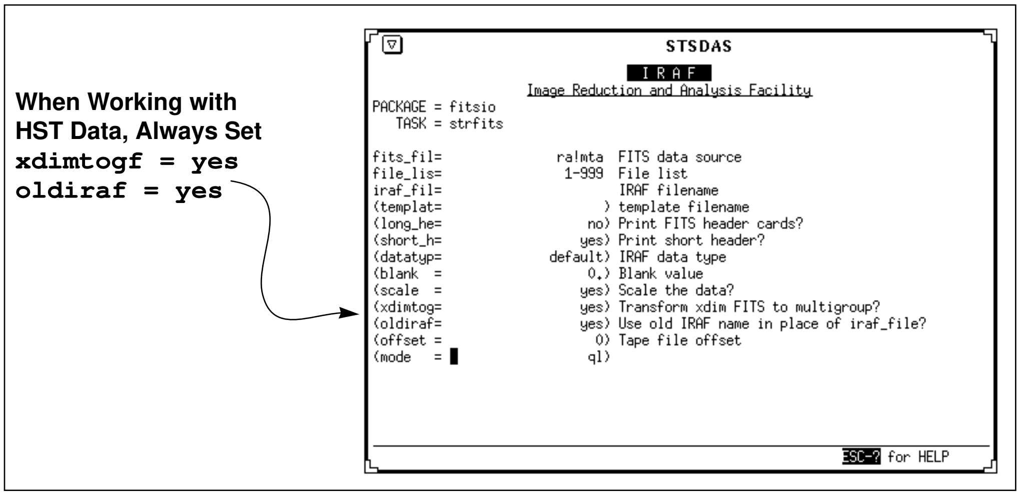

Figure 4.2: Setting strfits Parameters

Reconstructing GEIS Images From Multiple FITS Files

If you requested that your data be shipped to you with each group of the multigroup GEIS files written to separate files, you can reconstruct the original GEIS image and use it with the STSDAS calibration and analysis tools. To reconstruct the image, you must:

1 • Set the `imtype` environment variable to “`hhh`”.

2 • Find an appropriate template file containing the correct group parameters for the instrument.

STSDAS is shipped with appropriate template files in the `scidata$` directory. If, for example, you are using FOS data, your template file would be `scidata$fos.hhh`.

3 • Run the **strfits** task to reconstruct the first group of the image, defining the number of groups and the template file to use. (See the example in Figure 4.3).

4 • Run the **strfits** task once for each additional group in the file.

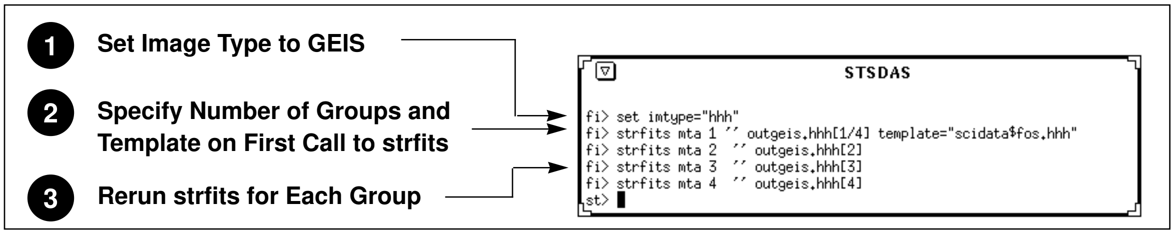

Figure 4.3: Reconstructing a Multigroup GEIS File

Writing FITS Tapes and Files

If you wish to write HST data sets to a FITS tape, you must:

1 • Allocate a tape drive and mount your tape as described on page 63.

2 • Set the **stwfits** parameters (type `epar stwfits`).

3 • Write the files to tape by typing `stwfits`.

Figure 4.4 shows an example of setting the **stwfits** parameters using the **epar** parameter editor. If you are writing your data to a new tape, there are three parameters you must set:

- `iraf_file` - The name of the file(s) that you want put on tape.
- `fits_file` - Set this to the IRAF name of the tape drive on which your tape is mounted (note that you must use the networking syntax if you are on a remote workstation).
- `newtape` - Set this to “yes” if you are writing a new tape, or “no” if you want the files appended to an existing tape.

If you are using an Exabyte tape drive, you should also set `blocking = 10`. Setting the blocking factor to 10 may be useful for other tape devices as well—you could save a lot of tape if you’re using a 9-track tape at 6250 bpi.

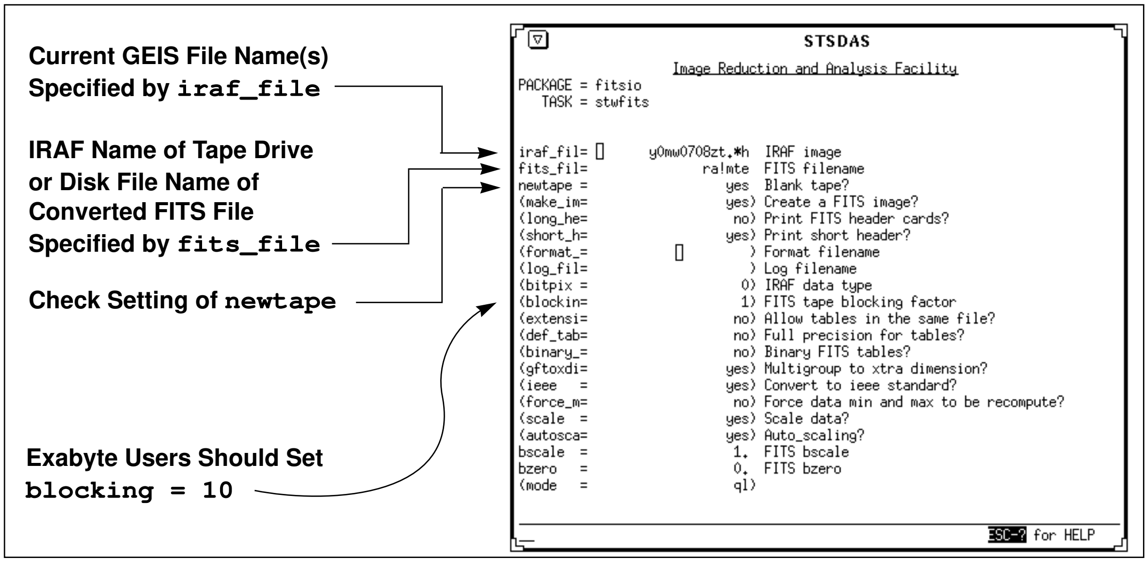

Figure 4.4: Setting stwfits Parameters

Once your parameters are set, run the **stwfits** task to write the files to tape and then dismount the tape and use the **deallocate** command to release the drive.

You can use **stwfits** to convert your multigroup GEIS files to FITS format so that you can work with the files using third-party software. In this case, you will write the files to disk rather than to a tape. To do this:

1 • Use **epar** to set the **stwfits** parameters. You need to set `iraf_file` to the name(s) of your GEIS files and set `fits_file` to the name(s) of the converted FITS-format file.

2 • Run **stwfits** as before.

For more information about using the **fitsio** tasks, see the two online help discussions for FITS examples and the GEIS format. Type `help fits_exampl` and `help geis`.

For more information about the **fitsio** package, see the online help files. Type `help fits_exampl` and `help geis`.

STSDAS Tables: the ttools Package

In addition to the image data format used for all HST data sets, STSDAS (through the TABLES package) supports a table format to handle tabulated information of different data types. It is also used to accommodate output from various analysis tasks and to facilitate communication between packages. STSDAS tables are binary files with data stored in rows and columns. Each table has a set of columns, containing various data types (i.e., character, integer, floating point, and boolean). All data values in a particular column must have the same data type. Virtually any number of rows can be in a table, and there is no restriction that the columns must fit across a terminal screen. Thus, it is possible to have a table with hundreds of columns.

Each column has the following attributes:

- Name (up to 19 characters, case insensitive)
- Data type (real, double, integer, short integer, boolean, or text)
- Format specification for displaying the contents of the column
- Units, a string of up to 19 characters (default is null)

Header parameters may also have data types of real, double, integer, boolean, or text. Header keywords are limited to eight characters (for FITS compatibility) and are converted to upper case.

Data in STSDAS tables are accessed using the name of the column. There is no need for the user to know the position of a column within a table, and the column names can be easily viewed using tasks in the **ttools** package. Because table entries are accessed by column name, information in a table can be easily shared by different applications: users (or programmers) need only know the name of the column containing the appropriate information.

Each STSDAS table is stored in a single file. By default, tables have the filename extension `.tab`. While you may give tables any filename and extension, it will be most convenient to use the default `.tab` extension, otherwise you will have to specify the extension explicitly when using the various table manipulation tools. Unless a table is opened read/write, `STDIN` and `STDOUT` are acceptable names for input and output text tables.

The table I/O software can also read and write plain ASCII tables. These are simply text files in row and column format. Column names in text tables are accessed by column number but using a character string starting with "`c`" and followed by the column number: `c1`, `c2`, `c3`, etc. While the file name of a binary table must have an extension, the name of a text table need not.

Users will find many opportunities for using STSDAS tables. Each calibration pipeline task (**cal*xxx***) reads from STSDAS tables to get the necessary calibration constants. The IRAF **daophot** package can produce results in either IRAF internal data base format or STSDAS table format. Tables produced by tasks such as **daophot** can then be analyzed using facilities in the **ttools** package.

The ttools Package

The **ttools** package contains a rich set of tasks for inspecting, editing (both with a text editor and with a graphic-based editor), and processing the information in tables. These tasks can be grouped as follows:

- ***Creating Tables:***
  - **tcreate** will generate an STSDAS table from plain ASCII tables. Users must provide the ASCII text for the table and a description of the columns.
  - Using **tedit**, users can interactively create a new table, defining column names on the fly and then inserting values into the table.
- ***Reviewing and Editing Tables:***
  - **tprint** will generate an ASCII listing of table contents (an option is provided to generate TeX or LaTeX output).
  - **tread** functions the same way as **tedit**, except that it runs in read-only mode.
  - **tupar** lets a user list and update header parameters in a table.
  - **gtedit** is a table editor with a graphical front-end. With **gtedit**, one can plot, for example, one column as a function of another and, using the graphics cursor, mark rows for deletion simply by pointing at points on the graph.
- ***Getting Information About Tables:***
  - **tinfo** tells you the size of a table (number of rows, number of columns, number of header keywords, etc.).
  - **tlcol** displays the names and other attributes of the columns in a table.

- ***Data Base Functions:*** The tasks **tquery**, **tsort**, **tproject**, **tselect**, **tdiffer**, **tjoin**, **tproduct**, and **texpand** provide various functions to use STSDAS tables as if they were small relational data bases. Information that you have in one table (or several tables) can be combined according to certain selection criteria, sorted, and compared.
  - **tselect** allows you to extract a subset of the rows in an existing table, subject to a specified selection criterion.
  - **tsort** sorts the rows in a table by the values in one or more columns.
  - **tproject** creates a subset of a table by retaining only the specified columns.
  - **tquery** combines the actions of **tselect**, **tsort**, and **tproject**; it also functions like the query mechanisms used in standard data base applications.
  - **tdiffer** can be used to create an output table that includes only the rows that differ between two input tables.
  - **tjoin** combines input from two tables into a third table based on a test for equality of values in a common column (for example, given a table of object names and magnitudes, and a second table of object names and colors, make a table that, for each object, contains columns with the magnitudes and colors—but only if the object names match, if not, the row is not copied to output).
  - **texpand** uses a set of user-defined rules to convert specified elements in a row, creating one or more modified versions of the row in the output file.
  - **tproduct** creates the Cartesian product between two tables (every combination of rows in the two tables). This can quickly make a big table!
- ***Statistics and Analysis:***
  - The **thistogram** task is used to compute a histogram for a given column in an STSDAS table. The output is directed to the IRAF standard output stream STDOUT or into a column in another table. The **sgraph** task can plot either type of output.
  - **tlinear** computes a linear regression on the data in two specified columns in an STSDAS table. The columns are used to specify the X and Y coordinates of the data to be fit. The output can be directed to a table or to STDOUT.
  - The data in columns of table can be resampled using the task **trebin**. The resampling is done over a user-specified range of the inde-

pendent variable, and the output will be equally sampled over this interval. Four options are available for computing the sampled data values: nearest neighbor replacement, linear interpolation, third order polynomial interpolation, and spline interpolation.

- The task **tstat** is used to determine the standard statistical parameters—mean, standard deviation, median, minimum, and maximum—for a column in an STSDAS table. The output can be directed either to STDOUT or to one row of another table.

- ***Performing Arithmetic on Table Columns:***
  - The STSDAS task **tcalc** is a general purpose table calculator. You can specify virtually any mathematical expression (using standard arithmetic and functional operators) involving columns in a table, computing a new table column.
  - **tintegrate** computes the integral of the data in a table column with respect to a second column. The integration is simple—using the trapezoid rule—and the output is written to both STDOUT and task parameters.
- ***Converting Keywords, Parameters, and Entries:*** These tasks (**keypar**, **keytab**, **parkey**, **partab**, **tabpar**, and **tabkey**) are used to copy parameter values from one location to another location. The values can originate either as CL parameters (such as a task parameter), keyword values in a table or image header, or entries in a particular column and row of an STSDAS table. The six tasks discussed in this section provide for all possible of combinations of input and output destinations, with `key` indicating a header keyword, `par` indicating a CL parameter, and `tab` indicating an element from a table. For example, the **keypar** task name indicates that the program would convert keywords to tables.
- ***Changing Column Names or Table Sizes:*** These routines let users modify the structure of STSDAS tables.
  - **tchcol** lets users rename a column or change its display format or units.
  - **tchsize** is used to modify the allocated sizes of internal table structures, such as the size of the header area, the space allocated for column descriptions, the length of the table row, and the number of rows for which space is reserved. Using this task improperly can be counterproductive, so make sure you understand the task parameters before you use this facility.

- Copying and Deleting Tables:
  - **tcopy** copies an input table to an output table; it differs from a standard file copy operation only in that it automatically appends the standard table extension `.tab` and verifies that the file is a table.
  - **trename** and **tdelete** function like **rename** and **delete** except that they will assume a `.tab` extension if none is specified (and will verify that the file is a table).

Several other tasks can be found in the **ttools** package but are not described here. These include: **tcheck**, **tdump**, **tmerge**, **imtab**, and **tabim**.

Graphics and Plotting

This section is divided into two parts: the first describes, in general terms, graphics tasks available to STSDAS/TABLES users; the second part (beginning on page 74) is a tutorial showing how some of these tasks can be used. If your site has only TABLES, but not STSDAS, installed then you will only be able to use the **igi** and **sgraph** tasks, and their related parameter sets **axispar**, **dvpar**, and **pltpar**, in addition to the traditional graphics tasks distributed with the IRAF system.

Graphics Tasks Overview

STSDAS users can often use the general facilities in the IRAF **plot** package. However, the **plot** package cannot plot data from STSDAS tables, and will not plot data from multi-group STSDAS format data files very effectively. Thus, STSDAS provides its own plotting package, **stplot**, as an adjunct to the functions provided in IRAF.

The **sgraph** task contains all of the functions of IRAF's general purpose plotting (i.e., the **graph** task in the **plot** package), but has added capabilities. In addition to plotting from IRAF or STSDAS images and text files (STDIN), **sgraph** will plot values from STSDAS tables. One may plot a single column against row number or one column against another column in the same table or another table. Lists (text files) and image sections can be mixed in the input parameter. If the curves are of different lengths, the plot will be scaled to the longest curve and curves will be left justified. If an image section has more than one dimension, the average projection will be computed and plotted along a designated axis. A unique dash pattern (up to 4) is used for each curve (by default).

The **depind** task prints pairs of dependent and independent pixel values from two one-dimensional images to STDOUT. This may be piped to **sgraph** or **graph** to plot one image line (spectrum) against another. A useful application of this task is plotting fluxes in one file against wavelengths in another file.

The **skymap** task interprets an STSDAS table as a catalog of coordinates and brightness to produce a star chart. That is, it plots symbols whose size depends on brightness (i.e., magnitude) and whose position on the plot is a projection of the celestial coordinates. There is an interactive interface which allows a user to roam the catalog, change the chart scale, and perform other actions.

Two tasks deal with group format STSDAS images and can be used to plot more than one group member on a single graph. The **grplot** task overplots selected group members on a single plot. The **grspec** task uses the **specplot** task in the NOAO **onedspec** package to allow interaction with a plot of several group members.

The **fieldplot** task is a general purpose plotting task to draw arrows representing directions and magnitudes (vector fields). Data are read from a text file, or from STDIN, to represent the coordinates of a point and the field vector. The vector may be either a pair of projected magnitudes or the absolute magnitude and direction. The magnitude may be scaled arbitrarily.

histogram plots data in a histogram format. The data can be read from an STSDAS table, STSDAS or IRAF image, or text file.

The **igi** task is an interactive interpreter for producing arbitrary plots. Although **igi** is a complete graphics program in itself, it operates within IRAF and is layered on the same low-level graphics facilities.

newcont is an alternative to the **contour** task in IRAF's **plot** package. It implements a different algorithm for computing contours that can produce more accurate results, but at some cost in efficiency. It also includes additional capabilities, such as choosing arbitrary contour levels and contours with different line types.

siaper draws HST science instrument apertures and may be used to display the HST field of view. You can overlay the plot on a displayed image to help position targets.

wcslab draws axes, ticks, and labels in the coordinate system defined in the parameters of a two-dimensional image. You can overlay these axes on a displayed image. (Note that this is a different task from the **wcslab** task found in the IRAF **plot** package).

The **igi** and **sgraph** tasks (and their associated parameter sets) are included in the TABLES package in the **tbplot** package.

Tutorial: Plotting and Other Graphics

This section explains how to generate plots in the IRAF and STSDAS environment. This section will explain:

- Which tools you should use to create various types of plots and what capabilities those tasks include.
- Tasks for plotting spectra, such as **fwplot** (page 76) and **splot** (page 77).
- Tasks for plotting image data, such as **implot** (page 77) and **contour** (page 78).
- An overview of **igi**, the Interactive Graphics Interpreter (page 79).
- How to produce PostScript files from your plots (page 80).

Basic Plotting Tools

FOS and GHRS observers can plot spectral data using the **splot** and **fwplot** tasks. WF/PC and FOC observers can plot image data using the **implot** and **contour** tasks.

There are a few basic plotting tasks that you should investigate (roughly in increasing order of generality):

- **fwplot** - Located in the STSDAS **hst_calib.ctools** package, this task will quickly make a *simple flux vs. wavelength plot* of FOS or GHRS spectra, which is useful for getting a quick first look at the data.
- **splot** - Located in the NOAO **onedspec** package, this task will let you plot and analyze simple *one-dimensional* spectra (used with **mkmultispec**).
- **implot** - Located in the **plot** package, **implot** will plot the values in an image along a given X or Y position (*row* or *column*). Contour plots can be drawn with the **contour** task.

- **sgraph** - Located in the **tables.tbplot** package, **sgraph** is a general-purpose data graphing task that accepts input from text files, IRAF images, and STSDAS tables.
- **igi** - The Interactive Graphics Interpreter (**igi**) in the **tables.tbplot** package is a full-featured, command-driven data graphing task.[4]

Plotting tasks may be interactive or non-interactive. The **fwplot** task is an example of a non-interactive graphing task which draws a plot and returns to the CL. It produces a plot with no further control other than by rerunning the task. Interactive plotting tasks such as **splot** permit immediate interactive control over the operation of the task by entering *cursor mode* commands typed from the keyboard.

Plots produced by non-interactive tasks on a graphics display may still be modified to some extent using cursor mode, although there is no control over the task itself. Type `=gcur` to enter cursor mode from the CL. In cursor mode, a number of commands become available. The most important of these are:

- **Print** plots by pressing [=].

  This sends the plot to a *buffer*; to send all plots in the print buffer to the printer, type `:.gflush`.

- **List commands** by pressing [?].

From the CL prompt, type `help cursor` for an explanation of how the graphics facility works and how to use common cursor commands.

Printing Plots

There are two ways to print a plot:

- Run the task using the `device` parameter to specify the plot printer.
- "Snap" the displayed plot from cursor mode.

Device Parameter

To send the plot to the printer initially, set the task's `device` parameter, for example:

```
cl> set implot.device=stdplot
```

You can define the CL variable `stdplot` in your `loginl.cl` or `loginuser.cl` file; you can set it to be any printer known to IRAF. When you run the plotting task, there will be no visible plot on the screen,

4. A complete *IGI Reference Manual* is available from the STSDAS Group.

instead, it will be printed on the specified printer. Use the **gflush** command to send the plot to the printer.

Remember to use **gflush** after sending a plot to a printer.

Snap

You can print any plot drawn on the screen using the cursor mode command `:.snap` or it's shortcut, the = key. *You must be in cursor mode for this to happen.* Interactive plotting tasks put you in cursor mode. You can always enter cursor mode by typing =`gcur` at the CL prompt.

Plots will be printed on the printer defined by the environment variable `stdplot`. (See "Setting Environment Variables" on page 18).

Using fwplot

GHRS and FOS observers can quickly produce a flux vs. wavelength plot of a spectrum using the **fwplot** task in the STSDAS **ctools** package. This takes a single argument—the name of the calibrated flux (`.c1h`) file.

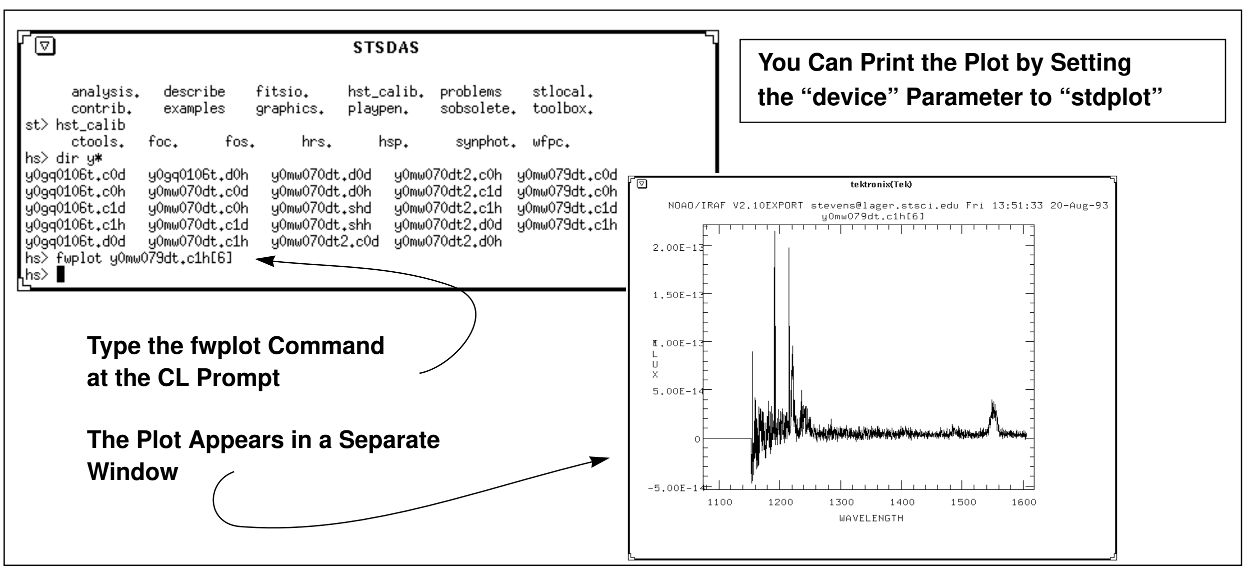

Figure 4.5: Simple Flux Versus Wavelength Plot of Spectral Data

Using splot

The **splot** task is used to plot one-dimensional data. Although you can use the task directly on your calibrated HST spectra, most observers find **splot** most useful after combining the wavelength and flux information using the **mkmultispec** task. When running **mkmultispec**, exclude unexposed pixels using the range syntax described in “Image Sections and Groups” on page 28.

Figure 4.6 shows an example of how a typical FOS observer might combine the wavelength and flux using **mkmultispec** and then plot the data using **splot**.

Reminder: Use the **help** command (or **phelp**) to find out more about these tasks.

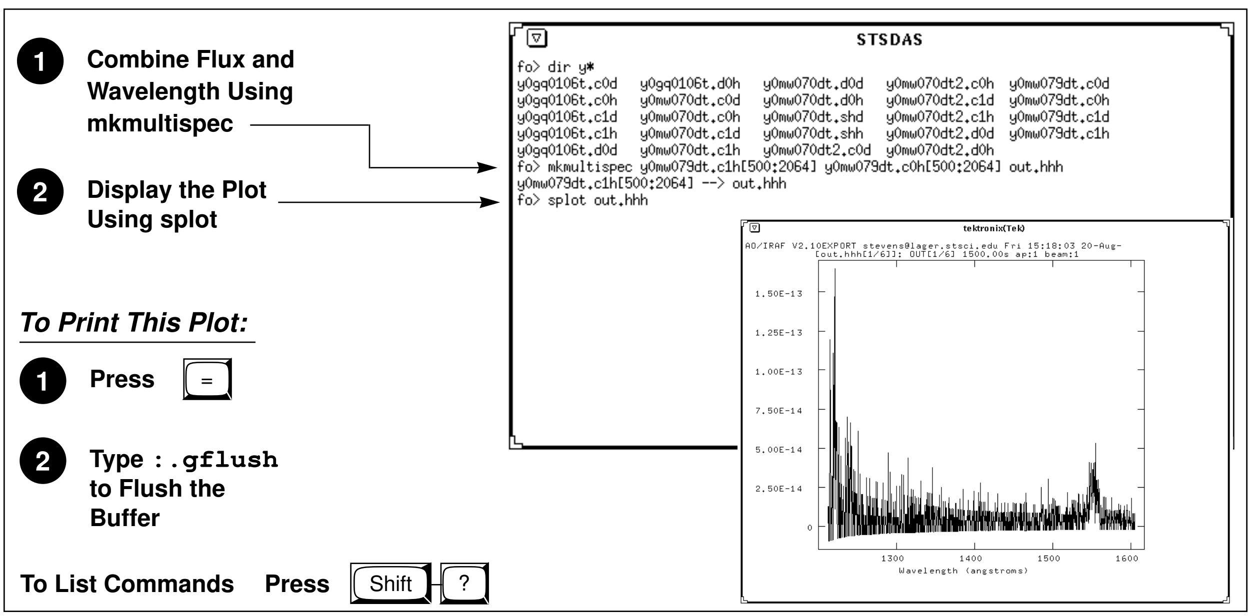

Figure 4.6: Using splot with Combined FOS Flux and Wavelength

Using implot

The IRAF **implot** task (in the **plot** package) will let you interactively examine an image by plotting data along a given *line* (X axis) or *column* (Y axis). When you run the task, a large number of commands are available, in addition to the usual cursor mode commands common to most plotting tasks, a complete listing of commands is found in the on-line help, but the most common are listed in Table 4.2.

Keystroke	*Command*
Shift – ?	Display on-line help
L	Plot a line
C	Plot a column
Q	Quit implot
Space	Display coordinates and pixel values

Table 4.2: Basic implot Commands

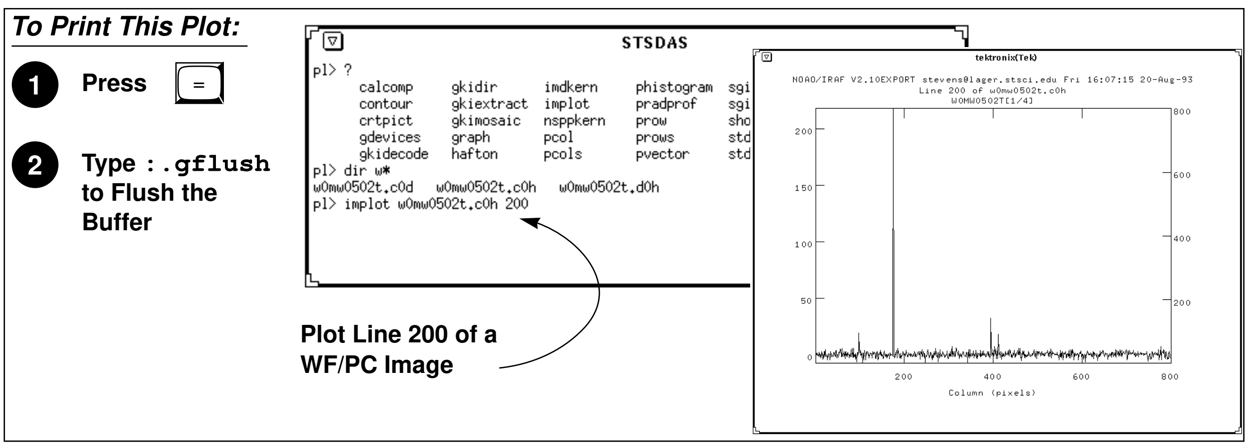

Figure 4.7: Plotting Image Data with implot

Contour Plots

Contour plots of image data can be created using the **newcont** task in the STSDAS **stplot** package. This is similar to the **contour** task in the IRAF **plot** package, but uses a different contouring algorithm and includes additional capability such as labeling in world coordinates. Figure 4.8 is an example of how to use the task with a WF/PC image.

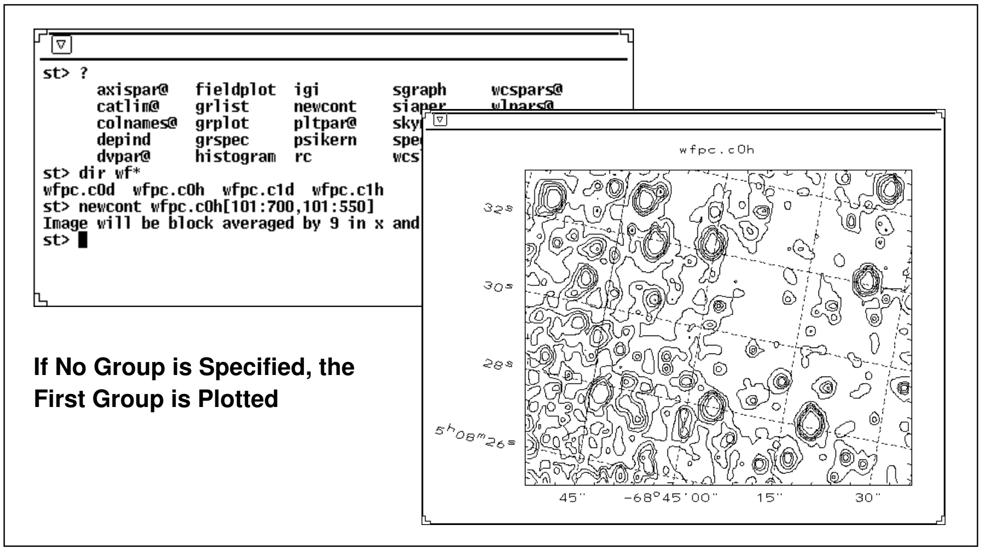

Figure 4.8: Contour Plots Using newcont

Using igi

As your plotting needs grow more sophisticated—and especially as you try preparing presentations or publication-quality plots—you should investigate the Interactive Graphics Interpreter, or **igi**. This is a task in the STSDAS **stplot** package that can be used to draw axes, error bars, labels, and a variety of other features on plots. Options are available to use a number of different line weights, font styles, and feature shapes, enabling you to create complex plots. Figure 4.9 shows a very simple example plot created in **igi** and the commands that were used to do it.

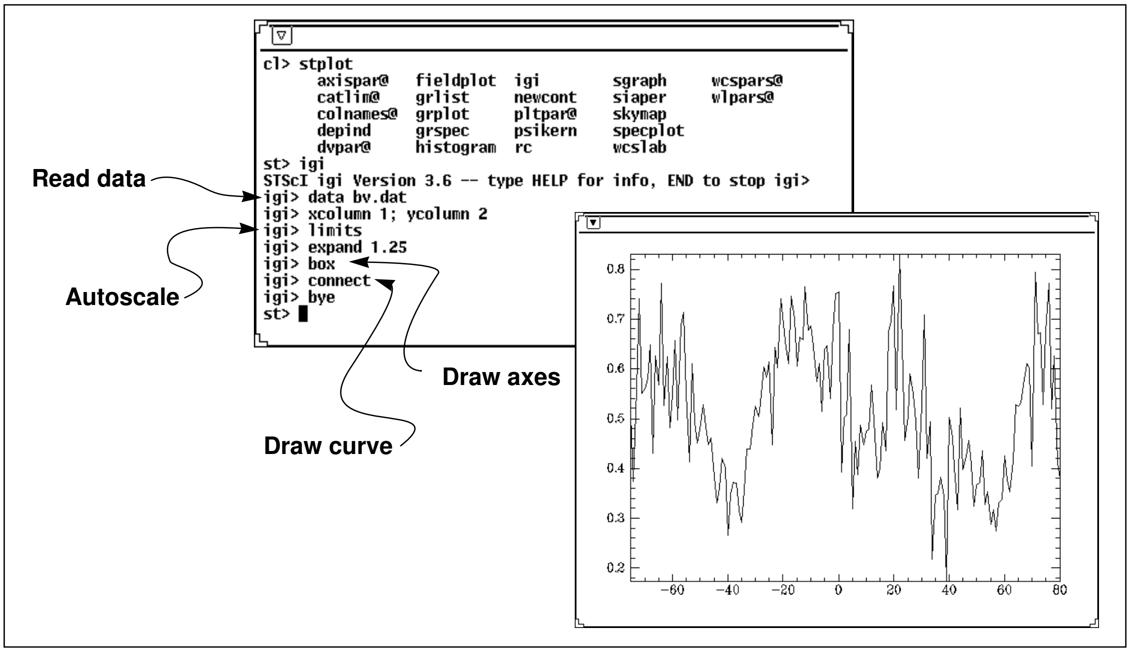

Figure 4.9: Sample igi Plot

In addition to standard graphing facilities, **igi** includes some data manipulation capabilities and programming capabilities, such as macros. There are also commands for rendering grayscale and pseudo-color images.

Because **igi** is a complete graphics environment in itself, a full description is well beyond the scope of this document. You can learn more about **igi** by reading the on-line help (`help igi` from within IRAF) or by reading the *IGI Reference Manual*, available from the STSDAS Hotseat.

PostScript Plots

STSDAS tasks can take advantage of many PostScript capabilities using the PostScript kernel, or *psikern*. Once installed, psikern provides PostScript capabilities such as the ability to work with color, and to do grayscale image rendering. These features are provided transparently—you simply specify the name of the printer (as known to IRAF) that you wish to use. This allows you to print plots directly to PostScript printers or to save them in Encapsulated PostScript (EPS) files which can then be imported into documents or rendered on the screen with a PostScript viewer such as PageView or GhostScript.

Not all tasks make use of these extended capabilities, but some of the **graphics.stplot** tasks do let you specify colors and render gray scale images. In particular, **igi** takes advantage of all of the added capability of psikern, including selection of color, filled-area patterns, and continuous-tone monochrome and color images. Other tasks such as **newcont**, **skymap** and **wcslab** take advantage of color. Any graphics task in STSDAS and IRAF can make use of improved font support.

There are two commonly-used ways to create a PostScript file of your plot (see the on-line help; type `help psikern`):

1 • With the plot displayed, from cursor mode, type:
`:.snap psi_port` (or `psi_land`)

Where `psi_port` prints in portrait mode (i.e., tall orientation for the paper) and `psi_land` prints in landscape mode (i.e., wide paper orientation).

2 • Before running your **plot** task, set the `device` parameter to `psi_port` or `psi_land`.

The system will automatically name the file, and usually store it in a temporary directory. When you type **gflush**, you will see the file name as created by **psikern**. You should then move the file to an appropriate directory where it won't be deleted. For example:

```
st> sgraph dev$pix device=psi_land
st> gflush
/tmp/psk4575a
st> copy /tmp/psk4575a home$plot.ps
```

The toolbox Package

This package describes utility programs in the STSDAS toolbox package, which is divided into six subpackages for examining, editing, reformatting, and operating on images and tables. These subpackages include:

- General utilities in the **tools** package
- Tools for examining and editing image headers in the **headers** package
- Tools for working with images in the **imgtools** package
- Tools for converting images and tables from one machine binary format to another in the **convfile** package

General Utilities: tools

The **tools** package is a diverse set of general utility tasks. Tasks of astronomical interest include **epoch**, which converts between different date formats, and **tprecess**, which precesses images, tables, or lists of coordinates. The **fparse** task parses a filename into its component parts and is useful when writing scripts that perform filename manipulation. The **ddiff** task compares directory tree structures and produces a list of their differences; this can be useful for locating missing files. The **mkapropos** task creates the data base used by the **apropos** task. Tasks used for renaming newly-delivered reference files for the Calibration Data Base (CDBS) include **uniqfile**, **uniqid**, **uniqname**, and **uniqtab**.

Working with Image Headers: headers

The **headers** package contains tasks for viewing and changing the contents of an image header. The **iminfo**, **hcheck**, and **hdiff** tasks are used to view an image header. **iminfo** prints out a nicely formatted display containing the most important information from the image header. It is mostly intended for HST images, as it searches for instrument-specific keywords. The **hdiff** task prints the differences between two image headers. It can be used to check for disagreements between an image's header and that of its data quality mask. **hcheck** is used to catch inconsistencies between header keywords within the same header. The user creates a file containing conditional print statements that express the consistency relationships and the specified header keywords are printed when the consistency relationships are violated.

The remaining tasks in the **headers** package change the contents of the image header. The **eheader** task is an interactive editor for the image header. **groupmod** allows adds or deletes group parameters from an image. **stfheader** adds history information from a file to an image header. **chgkeywrd** renames header keywords according to a set of replacement names contained in a file. The **imagree** task modifies the header keyword containing the image name to agree with the actual image name. And **imkeyname** performs the inverse operation, renaming the image to agree with the header keyword.

Working with Images: imgtools

The **imgtools** package contains tasks that manipulate the pixels of an image. Several tasks can be used to create a new image. **imcalc** creates an image as a mathematical function of one or more input images. **gcopy** copies all the groups of an input image to an output image. (Unlike **imcopy** in the IRAF **images** package, which will only copy a single group.) **moveheader** combines the pixels of an input image with the header of a second input image to produce a third image. **iminsert** overlays a first image on a second to produce an output image. **improject** produces an output image of reduced dimensionality from an input image by either summing or averaging over the reduced dimension. **stack** performs the opposite operation, by combing several images to produce one of higher dimensionality. **rbinary** converts a binary file with contiguous pixels into an image. **xyztoim** interpolates on a table with position and intensity information to create a two-dimensional image. **xyztable** converts the same information to a new table. **mkgauss** creates an image of a two-dimensional Gaussian function.

Several other tasks update an existing image. **pixedit** will change individual pixels in an image. **boxinterp** fills a rectangular area with values interpolated from the surrounding area. **imfill** sets pixels in an image to a fill value whenever a mask file flags the pixels as bad.

Two tasks are intended specifically for mask files. **addmasks** combines two or more masks into an output mask. **copymask** converts a mask from one format to another.

The remaining tasks report information about an image. **listarea** prints a section from an image. **pixlocate** prints the locations of pixels falling inside or outside an intensity range. And **xy2rd** translates the coordinates of a pixel in a two-dimensional image to right ascension and declination.

Converting Files: convfile

The **convfile** package contains tasks for converting the machine-dependent file formats used for storing GEIS images and binary STSDAS tables to formats used by other machines. This conversion is required if the files are being moved to a machine where the byte ordering is different, as was mentioned in Chapter 3. Note that this conversion is *not* required for files that are stored in machine-independent format such as FITS files and ASCII files.

Tasks in this package include:

- **sun2vax** – Converts images from Sun binary format to VAX format.
- **vax2sun** – Converts images from VAX binary format to Sun format.
- **tconvert** – Convert tables between Sun and VAX format. The type of conversion to be performed is specified by the `direction` parameter.

Tasks in the **convfile** package are often used in conjunction with IRAF networking so that the file format is converted as the file is copied across the network to a machine with a different architecture (see "Networking Syntax" on page 34). For example, if you were running IRAF on a Sun and had tables on the remote VAX host `abcvax` in the directory `disk$scratch:[caldir]`, you could copy the tables to your current directory on the Sun by using the following commands (note that the `direction` parameter is explicitly set to "vs"):

```
co> set scrdir = "abcvax!disk\$scratch:[caldir]"
co> tconvert scrdir$*.tab . vs
```

CHAPTER 5:

General Data Analysis Facilities

In This Chapter...

In addition to the instrument-specific applications and the general tools described in the previous two chapters, STSDAS includes seven general data analysis packages that can be used on essentially any astronomical data. These packages are grouped under the **analysis** package and include image restoration, various fitting routines, fourier analysis, access to the Guide Star Catalog, and statistics. This chapter provides an overview of the generic analysis facilities provided by STSDAS. Some of the features found in **analysis** subpackages include:

- ***Feature Fitting:*** There are two packages that contain general tasks for fitting image features: **isophote** and **fitting**. Both are layered on the IRAF **icfit** utilities. The **isophote** package fits galaxy brightness profiles in two-dimensional images with elliptical isophotes. The **fitting** package fits elliptical Gaussians to two-dimensional images and can also fit one-dimensional Gaussians, Planck or powerlaw functions, and composite profiles to spectra or images.

- ***Fourier Analysis:*** The **fourier** package contains tasks for forward and inverse Fourier transforms as well as tasks for computing power spectra, cross-correlations, convolutions etc.
- ***Guide Star Analysis:*** The **gasp** package contains tasks for searching the Guide Star Catalog and extracting portions to STSDAS tables. Tasks can derive a plate solution for an image, determine celestial coordinates of objects, and write World Coordinate descriptors to an image header based on the plate solution.
- ***Image Restoration:*** The **restore** package contains several tasks for restoring images, especially those produced by HST with its aberrated mirror.
- ***Statistics of Censored Data:*** Statistical methods for the analysis of censored data are implemented in the **statistics** package, which was adapted from the ASURV v1.2 package developed at Pennsylvania State University. Upper and lower detection limits can be applied, excluding from analysis data meeting certain statistical criteria.
- ***Time Series:*** One-dimensional photometric time series analysis is provided by the **timeseries** package. Tools are provided for coadding data over phase intervals and transforming flux measurements to instrument or standard magnitude systems. Auxiliary tools are available for other functions, such as calculating light travel time corrections to solar system barycenter, and subtracting sky background.

Other tasks that may be useful to the HST observing community, but that are not yet complete are provided in the **playpen** package. Tasks that may be useful but that were written outside the STSDAS group are provided on a non-supported basis in the **contrib** package.

Image Restoration

In this section we will describe:

- A general overview of image restoration
- Algorithms and tasks in the STSDAS **restore** package (page 87)
- How to prepare images for restoration (page 90)
- How to find and use PSF images (page 92)

Overview of Image Restoration

Image restoration[1] (or *deconvolution*) techniques are often needed to compensate for the error in HST's primary mirror. Deconvolution methods will likely continue to be important in the analysis of HST data after the installation of WF/PC-2 and COSTAR. Image restoration can remove diffraction spikes, reduce confusion, improve dynamic range, and provide modest gains in resolution. In some cases, significant super-resolution might be attained.

All restoration methods are likely to produce better results given higher S/N data. There is no magic number that can be given, however, to say what S/N ratio is adequate. The quality of a restoration depends not only on the S/N ratio, but also on the nature of the object (extended or point-like), the background (strong DC or complicated background structure present vs. empty background), and the quality of the point spread function.

Algorithms and Methods in STSDAS

Several restoration algorithms are provided in the STSDAS **restore** package. These algorithms, as implemented in STSDAS, are described in some detail in Chapter 12 of the *HST Data Handbook*. Summary descriptions of the following algorithms are provided in the next subsections:

- Richardson-Lucy algorithm: the **lucy** task
- Maximum Entropy Method: the **mem** task
- σ-CLEAN algorithm: the **sclean** task
- Wiener filter and Fourier inverse filter: the **wiener** task
- Adaptive filters: the **hfilter** and **adaptive** tasks

All of these tasks are described in detail in the on-line help for each task.

Richardson-Lucy Algorithm

The STSDAS **lucy** task generates a deconvolved image from an input image and point spread function (PSF). The output image will be the same size as the input image unless an optional larger output size is specified. The PSF image must contain the same number of dimensions as the input

1. A list of references on the image restoration topics described in this section is provided in Appendix B on page 209.

image, but the actual dimensions do not need to match. If a background (sky) image or mask image are given, their dimensions must match those of the input image. If a model image is specified, its dimensions must match the output image size. When the optional background image is specified, the **lucy** task will require six real and two complex arrays, i.e., a 512x512 image will need 10 MB of disk space.

One common question about the R-L method is when to stop the iterations. The usual rule of thumb is to run the R-L iterations until the χ^2 per degree of freedom (χ^2 / number of pixels) reaches 1.0. This only provides a global measure of the quality of the fit, however, since the rate of convergence varies for objects of different brightness and different spatial scales. For stellar fields with little or no background the R-L iterations can (and should) be run a long time—many hundreds or even thousands of iterations are appropriate. It would be possible to run the iterations to true convergence, i.e., such that the differences between successive image estimates are smaller than the numerical accuracy of the computation.

Maximum Entropy Method

The Maximum Entropy Method (MEM) is very versatile, and can be given constraints on the solution that make it as powerful, if not more so, than the R-L method. Owing to the flexibility of the method, MEM is somewhat more difficult to use and is generally more computationally intensive than R-L. Many people may be familiar with the *MEMSYS 5* package from Maximum Entropy Data Consultants, Ltd., and the front-end program *MEM* developed by Nick Weir at Caltech. STSDAS now includes a non-commercial MEM implementation in the **mem** task, written by Nailong Wu, which gives comparable results to *MEMSYS 5* and has a computation time similar to the R-L method. The user interface has also been simplified, with inputs very similar to the other restoration tasks in STSDAS. The task generally takes as input three images:

- The degraded HST image
- A point spread function image
- A model image

Two input parameters are specified: `noise` and `adu`. The output image will be the final converged MEM image, or an interim result that you can use as a model for the next iteration.

Generally speaking, R-L is more robust than *MEM*—that is, even though in some cases *MEM* methods may give better results, the R-L method is more stable. In some cases *MEM* will not converge to a solution. MEM tends to give the best spatial resolution but somewhat less reliable photometry (although recent tests comparing the STSDAS **mem** and **lucy** tasks show very comparable photometric linearity). *MEM* implementations are often more computationally intensive.

σ-CLEAN

In the standard CLEAN algorithm the image is searched for the brightest pixel, and a fraction (the *loop gain*) of the source is subtracted at this location by scaling the PSF accordingly. This process is repeated until the noise level of the background is reached. At the same time, a list is kept of each source removed from the data. At the end of the process, these sources are added back to form a *clean map* using a modified PSF such as a narrow Gaussian profile. Keel's modification to CLEAN is to employ a noise model for the image so that rather than simply locating the brightest pixel on each iteration, one finds the most significant pixel (taking into account read-out noise, sky background, and Poisson noise).

Our experience is that the σ-CLEAN method (the **sclean** task) requires substantial computing time and often does not have photometric linearity as good as the R-L method, most likely because iterations are stopped prematurely.

Wiener Filter

The **wiener** task applies a Fourier non-iterative deconvolution filter to two-dimensional images. The filter includes the familiar inverse, Wiener, geometric mean, and parametric forms. A general description of the method is provided on-line by typing `help wiener opt=sys`.

Adaptive Filters

The **hfilter** task is a simple adaptive low-pass noise filter based on the H-transform. The *H-transform* is a form of image coding based on the Haar transform, which keeps a tight one-to-one relationship between pixels in the transform space and data pixels in the image space. Using the H-transform, it is possible to design filters that adapt themselves to the local signal-to-noise and spectral properties of the data.

The **adaptive** task reduces the noise in parts of the image without reducing resolution of sharp features. The local S/N ratio as a function of decreasing resolution is evaluated as follows: at any given point, mean gradients and curvatures over different scale lengths are compared to expected noise values. The order for which the S/N ratio exceeds a specified level indicates the local resolution scale length of the signal, which determines the size of the filter applied at that point.

Image Preparation

You must prepare your images to get the best possible restoration results. Some techniques will require you to repair bad pixels (such as cosmic ray hits, data dropouts, etc.), while others will allow you to apply a mask or a weighting array. The following problems and solutions are described in this section:

- Cosmic ray removal and bad pixel masking
- Saturated pixel removal or masking
- Image boundary or edge considerations
- Combining images of different resolutions

Cosmic Ray Removal and Bad Pixel Masking

When exposures are split into several shorter length exposures it becomes much easier to identify cosmic rays and other data defects that do not occur in the same pixel on each exposure. Given the undersampling problems of the WF/PC, cosmic ray identification is a significant problem; faint stars can be difficult to distinguish from low energy cosmic ray hits. If hard cosmic rays hits are left in the data, the restored image will usually end up having a PSF-like artifact introduced at the location of the cosmic ray. Thus, it is important to first combine CR-SPLIT images using a task such as **wfpc.combine** to remove the cosmic rays. If only a single data frame is available try using the **cosmicrays** task in the **noao.imred.ccdred** package. This task can repair the cosmic ray hits and also create an output list of the affected pixels, which can be converted to a mask for use in the **lucy** task.

The best approach is to generate a mask for each image which flags the cosmic rays so that they are ignored by the restoration algorithm. The **lucy** task accepts such a mask and will give flagged pixels zero weight. If the algorithm does not accept input masks, replace the bad pixel with an interpolated value. This gives a statistical significance to the pixel that is not warranted, but cannot be avoided for simple algorithms such as the Wiener filter.

If CR-SPLIT images are not registered the cosmic ray removal routines will not perform well. Moreover, it is dangerous to resample the data in order to align the images given the undersampling of the WF/PC—a single bad pixel in the image can end up affecting a number of adjacent pixels after rebinning, and the accuracy of the rebinning is poor under-sampled data in general. Registration within 0.2 pixel is generally acceptable.

Saturated Pixels

Saturated pixels can be handled in several ways. If you have a shorter exposure of the same image at the same pointing (i.e., properly registered) you can patch in and replace the saturated pixels by scaling the shorter exposure image to the longer exposure time. The statistics of the patched in data points will not be the same, of course, and a new version of the STSDAS **lucy** task will allow the user to specific a weighting image to adjust for the different exposures.

Another approach is to mask out the saturated pixels and treat them as totally unknown. The **lucy** task in STSDAS will fill in the saturated pixels as best it can, given that the wings of the PSF appear in the data and are not saturated. The restoration will have some difficulty in restoring the full intensity of the saturated object, and a third approach is to use the result from, say, 100 R-L iterations as an input model, and restart the restoration ignoring the mask used initially. We have done some experiments along these lines but have not completed this work.

Another approach is to estimate or fit a PSF model to the region that is saturated using the wings of the PSF to determine the fit. Subtract the fit from the data, then restore the residual using your fit as a sky model (see the input parameters and help file for the **lucy** task). Note that this approach requires a very good model for the PSF.

Boundary Extension

Since restoration algorithms typically use Fast Fourier Transforms (FFTs), it is assumed that the image is a periodic or cyclical function. If there is a discontinuity from one edge of the image to the opposite edge, *ringing* (a pattern of high frequency ripples) will be introduced into the restored image. In order to avoid these problems, the image to be restored must be embedded in a larger image and the data must be extended in such a way as to make the image periodic with a smooth transition from one edge to the next (i.e., continuous first derivative). A task is under development in STSDAS (called **edge**) which will extend the image boundaries appropriately.

The STSDAS implementation of the R-L algorithm handles image edges slightly differently. The restored image can be chosen to be larger than the original data, and the padding region is simply given zero weight in the iteration. The restored image is allowed to have flux in the padding region, but the flux is based only on the known pixel values (e.g., a partial PSF pattern from a star just outside the image boundary).

Combining Images of Different Resolution

Images of varying resolution can be combined into a single image using an algorithm developed by Leon Lucy and Richard Hook of the ST-ECF. The task **acoadd** in the STSDAS **contrib** package is their implementation of this method.

PSF Preparation

You can use either an observed or a model PSF. Model PSFs are produced by the Telescope Instrument Modelling (TIM) and TinyTim software.

TIM and *TinyTIM* model PSFs are reasonably good in the optical part of the spectrum, but disagree with observed PSFs more and more as you go further into the UV. If possible, you should try using both an observed and a model PSF and compare the results, looking carefully for artifacts (especially features that resemble the PSF structure) in the restored images.

The best source for an observed PSF is in the same image *if* the PSF star is not too far from the object you want to restore (i.e., for WF/PC, within ~100–200 pixels). A PSF star near the edge of the image will not be suitable for restoring an object at the center of the image since vignetting causes serious distortions of the PSF at the field edge.

Other observed PSFs are available on STEIS in the WF/PC and FOC PSF Catalogs; which are described in files available via FTP, gopher, or WWW on the machine stsci.edu (see the directory instrument_news/wfpc /wfpc_psf_lib_07_08_93 and instrument_news/foc/ psf_files/psf_info.ps). The FOC PSFs are actually on-line on stsci.edu, and the WF/PC PSFs are stored in the HST archive. You may also need to search the HST archive to find exposures of stars in the same filter and position in the field (for WF/PC); unfortunately, there are no keys in the Archive database identifying fields with good PSF stars. When using an observed PSF, it is important that it be well-exposed. A noisy PSF will add noise to your restored image. An observed PSF must also be cleaned of cosmic rays and other image flaws and have any background offsets removed. It is advisable to apodize a PSF that has been cut out from a larger image. This means that you multiply the PSF subimage by a function, such as a truncated Gaussian, which smoothly tapers the pixel values to zero at the edge but does not alter the profile of the PSF itself. Tools are being developed within STSDAS to automate the process of PSF extraction.

Either TIM or TinyTIM may be used to generate model PSFs. TinyTIM is easier to use. TinyTIM is available on all STScI science computer systems, and is available via anonymous FTP on stsci.edu (software/tinytim).

The Image Restoration Project has contracted with David Redding at the Jet Propulsion Laboratory to develop a more sophisticated PSF modeling and prescription retrieval code. This software will be made available to the HST user community as soon as possible.

There can be advantages to using subsampled PSFs for image restoration, especially for WF/PC images. Virtually all WF/PC images are undersampled, and the quality of an image restoration can be improved if the restoration is computed on a higher density pixel grid (e.g., 2×2 or 3×3 subsampling). Subsampling is most effective for data that is most severely undersampled, such as WF images in the UV. Bear in mind that increasing the size of the image and PSF by subsampling will increase the computation time and memory requirements for deconvolution. To use a subsampled PSF you must also subsample the data (and model, if you have one) to the same scale pixel grid. You can use block replication to subsample the data.

Interactive Curve Fitting

The STSDAS **fitting** package contains tasks that perform curve fitting operations on images, STSDAS table columns, or lists. Both linear and non-linear functions are supported. When fitting one-dimensional functions with input data from two- (or more) dimensional images, the data are projected to a one-dimensional vector before fitting the function. The fitting tasks write their results to STSDAS tables. A common table format is used by all tasks in the package, with the same column headers and formats. Table 5.1 lists the tasks available in the **fitting** package.

Task	*Function*
convert	Convert ASCII data base format to STSDAS table format.
function	Evaluate functions over images, tables, or lists.
gfit1d	Interactive 1-D linear curve fit to images, tables, or lists.
i2gaussfit	Iterative 2-D Gaussian fit to noisy images (script).
nfit1d	Interactive 1-D non-linear curve fit to images, tables, or lists.
ngaussfit	Interactive 1-D Gaussians fit to images, lists, or tables.
n2gaussfit	2-D elliptical Gaussian fit to images.
prfit	Print contents of fit tables created by fitting tasks.

Table 5.1: Tasks in Fitting Package

The linear function fitting task is **gfit1d**. It fits Chebyshev and Legendre polynomials, and linear or cubic splines, using Cholesky factorization to solve the standard least-squares normal equations. Non-linear function fitting is split between three tasks. The first, **nfit1d**, supports six different functional forms:

- Power law
- Planck (blackbody) function
- Sum of two Planck functions
- Sum of a power law and a Planck function
- Galaxy brightness profile (bulge + disk)
- User-defined functions

The second non-linear function fitting task, **ngaussfit**, is used to fit multiple Gaussians to one-dimensional data. The third task, **n2gaussfit**, is a simple non-interactive tool for fitting a two-dimensional Gaussian to image data. The task **i2gaussfit** is a script useful for running **n2gaussfit** with noisy data.

The task **function** performs the inverse operation, that is, it reads tables containing fitting results and builds images, STSDAS tables, or lists by evaluating the function over the input data coordinates. Another task, **prfit**, will list or print the fitting table contents in a human-readable format.

The **gfit1d**, **nfit1d**, and **ngaussfit** tasks allow the use of error bars in their fitting operations. These error bars, giving the inverse weighting for each data point, may be used for data from either list or table input. Symmetric error bars are read from a single column of error values, while asymmetric (individual lower and upper) error bars may be read from two error columns.

The **gfit1d** task uses the IRAF **icfit** suite of interactive curve fitting tools to perform fitting operations. Interactive graphical curve fitting begins by graphing the data points and the first guess for the function (if any). When the graphics cursor appears the user may modify the graph, the function coefficients, and the control parameters in a number of ways using both cursor mode keystrokes and colon commands. The user then starts the fitting process. The fit can be recomputed as many times as desired before storing the final results in the output table.

The **nfit1d** and **ngaussfit** tasks use a suite of interactive non-linear fitting tools, called **ncfit**, which is based closely on the IRAF **icfit** package. Non-linear fitting is performed by a least-squares downhill simplex minimization algorithm, which makes the tasks robust, but at some cost in computational speed. Initial guesses for the function coefficients are needed to fit non-linear functions by this method. Guesses can be specified through task parameters, from a previously generated table of results, or interactively after task startup. The **n2gaussfit** task uses the same numerical techniques, but is non-interactive. Therefore you must specify all initial guesses via task parameters and cannot modify any of the fitting parameters during the fitting process.

The **ngaussfit** task is really just a special case of the more general **nfit1d** task where the functional form is restricted to be Gaussian. **ngaussfit**, however, contains some unique features that allow the user to constrain certain parameters of the fits. For example, when fitting multiple Gaussians the coefficients for all components may either be expressed in absolute units, or you may specify only the first component in absolute units and subsequent components relative to the first (amplitudes as a factor of the first and centers as an offset from the first). This is useful, for example, for constraining spectral line ratios by atomic physics values, or spectral line positions by their difference in wavelength.

The **function** task evaluates a one- or two-dimensional function that was previously computed by one of the fitting tasks. The function can be evaluated over pixel coordinates in an image section, or over data values in a table column or an ASCII list file. The output will be a one- or two-dimensional image, table column, or list file, depending on the type of input used.

Examples of Using Fitting Tasks

Here's an example of how you can use the **ngaussfit** task to simultaneously fit several features in an emission-line spectrum. Using all of the default parameter settings for both **ngaussfit** and the **gausspars** pset we initiate the task by typing:

```
fi> ngaussfit n4449.hhh linefits.tab
```

This reads spectral data from the image `n4449.hhh` and stores the results of our line fits in the STSDAS table `linefits.tab`. When the task starts up, it will produce a plot of the spectrum in the graphics window and leave you in interactive cursor mode. At this point you can use the IRAF graphics windowing keystroke command W to window the graph, if desired, to the particular region of interest within the spectrum. Once that is done, you need to define the spectral baseline (continuum) by using the cursor to define the two endpoints of the baseline and typing B at each location. The baseline will then be overplotted on the spectral data. If the region containing the features to be fitted is a small part of the entire spectrum, then you should also define a sample region within which the fitting will be performed. This will exclude from the fit all data points outside of the sample region and will greatly reduce the processing time of the fitting. Use the S keystroke to define the two endpoints of the sample region.

The Gaussian components are described by three coefficients: the amplitude (above the baseline), the center location, and the FWHM. You can interactively define the initial location and amplitude for each feature that you wish to fit by simply placing the cursor at the peak of each feature and typing P. When you've defined all the features you want, press Q to exit the feature-definition loop. At this point the initial guesses for the FWHM of the defined features will be set to a value of 1.0 and will be set to be held fixed during the fitting. If this value is a reasonable initial guess, then all you need to do is change the coefficients from being fixed to variable using the `:var` colon command. If you type `:show` you will see a list of the current values for all coefficients. Coefficients 1 and 2 are the baseline zero point and slope, respectively, and then the coefficients for each of the Gaussian features follow in groups of 3. The first is the amplitude, the second is the center position, and the third is the FWHM. If we have three features defined, for example, coefficients 5, 8, and 11 will be the FWHM of each. To let the FWHM coefficients vary during the fit, issue the colon commands `:var 5`, `:var 8`, and `:var 11`. If you also want to set a different value for the initial guesses for these coefficients, use the `:coef` colon command: e.g., `:coeff 5 6.5` will set coefficient number 5 (the FWHM of our first feature) to a value of 6.5.

Now execute the fitting process by typing F. When the fit has been computed you can replot the solution using the R keystroke and can examine the results using the `:show` or `:vshow` commands. Because the computation of coefficient error estimates is quite CPU-intensive, the default mode of operations is to not perform error estimation. If you want to estimate errors it is most efficient to first iterate to a good fit with error estimation turned off, then issue the command `:errors` to turn error estimation on and reiterate on the fit one more time by pressing F again. Also note that by default the baseline coefficients are held fixed; you may want to allow these to vary (use the `:var` command) in order to achieve a better baseline fit than what you're able to define using your eye. Figure 5.1 shows the results of fitting the Hβ and [OIII] emission features in our spectrum of NGC 4449. Figure 5.2 lists the resulting coefficients and error estimates (in parentheses).

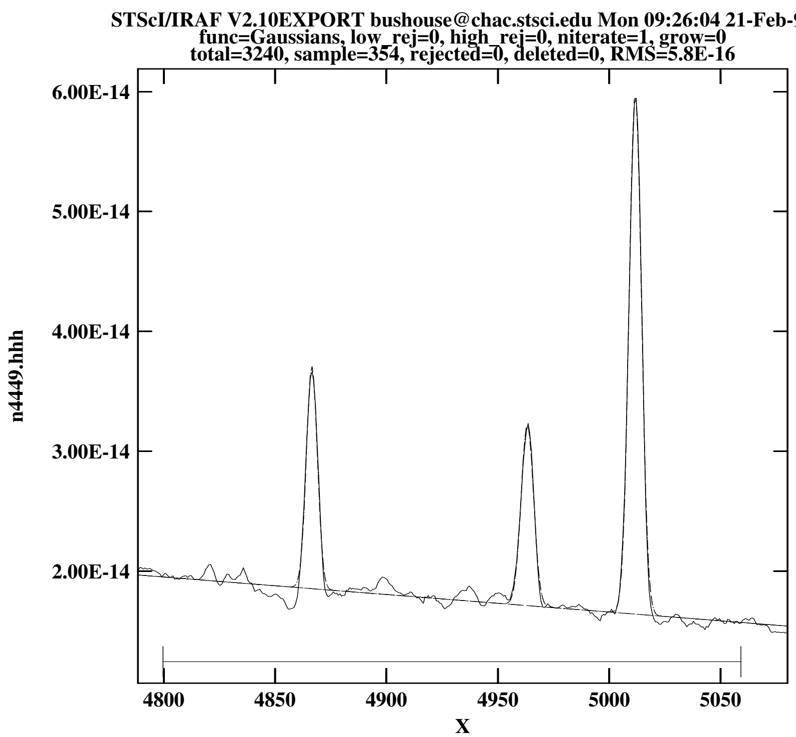

Figure 5.1: Fitting Hβ and [OIII] Emission Features in NGC 4449 Spectrum

```
function = Gaussians (fix)
coeff1 = 8.838438E-14    (0.)            - Baseline zeropoint (fix)
coeff2 = -1.435682E-17   (0.)            - Baseline slope (var)
coeff3 = 1.854658E-14   (2.513048E-16)  - Feature 1: amplitude (var)
coeff4 = 4866.511        (0.03789007)    - Feature 1: center (var)
coeff5 = 5.725897        (0.0905327)     - Feature 1: FWHM (var)
coeff6 = 1.516265E-14   (2.740680E-16)  - Feature 2: amplitude (var)
coeff7 = 4963.262        (0.06048062)    - Feature 2: center (var)
coeff8 = 6.448922        (0.116878)      - Feature 2: FWHM (var)
coeff9 = 4.350271E-14   (2.903318E-16)  - Feature 3: amplitude (var)
coeff10 = 5011.731       (0.01856957)    - Feature 3: center (var)
coeff11 = 6.415922       (0.03769293)    - Feature 3: FWHM
rms           = 5.837914E-16
grow          = 0.
naverage      = 1
low_reject    = 0.
high_reject   = 0.
niterate      = 1
sample        =  4800.132:5061.308
```

Figure 5.2: Coefficients and Error Estimates

Here's an example showing how the **gfit1d** and **function** tasks can be used to fit and evaluate radial surface brightness data for a galaxy. We'll assume that the surface brightness (magnitudes per unit area) as a function of radius has already been measured using the **ellipse** task from the STSDAS **isophote** package (see "Isophote Fitting" on page 106). The **ellipse** task has produced an STSDAS table called `a226iso.tab`, which contains (amongst others) columns of radius (to the 1/4 power), surface brightness, and lower and upper error estimates of the surface brightness. The labels for these columns are "a**1/4", "mag", "eml", and "emu", respectively. We set the **gfit1d** task parameters as shown in Figure 5.3.

```
input   = a226iso.tab a**1/4 mag  >data to be fit
output  =        a226gfit.tab  >output table
(axis   =                 1) >axis to project onto if we are fitting 2D image
(sample =                   *) >sample points to use in fit
(xmin   =               INDEF) >limits for normalization
(xmax   =               INDEF)
(errcolu=            eml emu) >column name for error column if fitting table
(functio=             spline1) >fitting function
(order  =                   1) >order of fitting function
(ps     =                 yes) >output as power-series polynomial ?
(interac=                 yes) >set fitting parameters interactively?
(naverag=                   1) >number of points in sample averaging
(low_rej=                  0.) >low rejection in sigma of fit
(high_re=                  0.) >high rejection in sigma of fit
(niterat=                   1) >number of rejection iterations
(grow   =                  0.) >rejection growing radius in pixels
(device =            stdgraph) >graphics output device
(cursor =                    ) >graphics cursor input
```

Figure 5.3: Parameters for gfit1d

The `input` parameter specifies the input table name, as well as the two columns from which the independent and dependent data will be read. The `errcolumn` parameter is used to specify the two column names from which to read the lower and upper error bar values. When we execute the task we get a plot that looks like Figure 5.4.

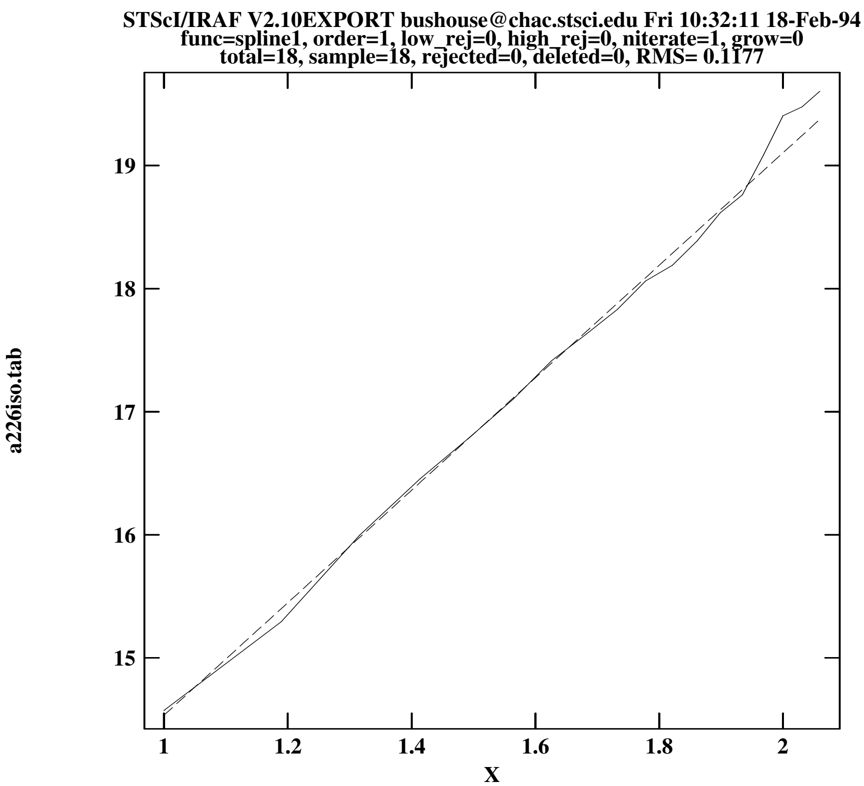

Figure 5.4: Sample Plot from gfit1d

The solid line in the graph connects the input data points, while the dashed line shows the fit. Upon exiting the task, the function coefficients are written to the STSDAS table `a226gfit.tab`.

Now, so that later we can make a nice plot of the input data with the fitted function overplotted, we'll use the **function** task to evaluate the function coefficients stored in `a226gfit.tab` at the list of radius values in our original data table `a226iso.tab`. We set the parameters for **function** as shown in Figure 5.5.

```
input1  =     a226iso.tab  >input file to which to apply the function
input2  =    a226gfit.tab  >fit table
output  =           gfit  >output file (or output column if input1 is a table
row     =              1  >row to use from fit table
(column =         a**1/4) >name of column to use if input1 is a table
(xandy  =            yes) >if list file should x and function(x) be output
```

Figure 5.5: Parameter Settings for function Task

This will evaluate the function over the list of radius values in the column "a**1/4" and create a new column in `a226iso.tab` called "gfit" that will contain the value of the function at each radius.

Finally, we can use the **sgraph** task to produce our final plot. First, we plot the original surface brightness vs. radius data using the following command:

```
cl> sgraph "a226iso.tab a**1/4 mag" point+ yflip+ \
>>>   wb=20.3 wt=14.4
```

This command tells **sgraph** to take its `x` and `y` data from columns "a**1/4" and "mag", respectively, in table `a226iso.tab`, plot in point mode (as opposed to connected-line mode), flip the Y-axis (because magnitude units run backwards), and set the range of the Y-axis to be from 20.3 to 14.4 (to give a little extra room for the error bars when we plot them next). Now run **sgraph** a second time in append mode to overplot the error bars:

```
cl> sgraph "a226iso.tab a**1/4 mag" \
>>>  errcol="eml emu" mark=vebar erraxis=2 app+
```

This tells **sgraph** to read the lower and upper error bars from table columns "eml" and "emu", respectively, use a vertical error bar for the marker type, and the error axis is in the Y-dimension (`erraxis=2`). And finally overplot the fit to the data running **sgraph** a third time as:

cl> sgraph "a226iso.tab a**1/4 gfit" append+

which tells **sgraph** to read the X and Y data from columns "a**1/4" and "gfit", respectively, and plot in line mode. The resulting plot is shown in Figure 5.6.

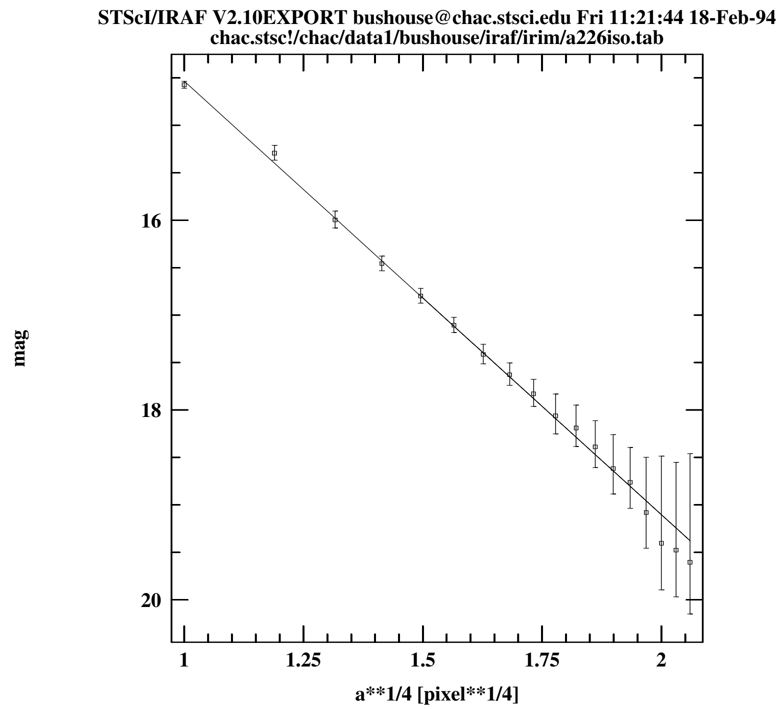

Figure 5.6: Producing Final Plot with sgraph

Fourier Analysis

The STSDAS package **fourier** contains tasks to compute forward and reverse Fourier transforms, auto-correlation and cross-correlation, power-spectra, and convolution. In addition, the package provides some test and diagnostic functions to help in understanding and visualizing Fourier transforms. The task **mkfunc** allows you to create your own data sets for experimentation, shift lets you shift them in phase, and **ftplot** provides a convenient graphics display of both input and output data (real and imaginary parts). **bracewell** demonstrates Fourier transforms from the catalog of standard transform pairs in Bracewell's book on Fourier transforms. **carith** does complex multiplication or division of images.

Tasks in this package can transform data arrays of arbitrary size, i.e., there is no restriction for the arrays to be a power of 2 in length. The tasks are layered on code that does a prime factor decomposition of the Fourier transform. Arrays that cannot be decomposed into prime factors will not be transformed very quickly (i.e., if your data file is 509 x 509 pixels, you should probably copy it to a 512 x 512 array; if it is 510 x 510, with prime factors 2, 3, 5, 17, it's okay). You can use the tasks **listprimes** and **factor** to check the size of your data set.

The Fourier transform tasks make a reasonable attempt to keep track of the coordinates attached to your input data, and a number of Fourier transform coordinate pairs are recognized: time/freq, lambda/wavenumb, ll-mm/uu-vv.

The **forward** task performs a Fourier transform, defined by:

$$F(s) = \sum_{x=1}^{N} f(x)\, e^{-2\pi i (x-1)(s-1)/N}$$

The **inverse** task performs an inverse Fourier transform, defined by:

$$f(x) = \frac{1}{N}\sum_{s=1}^{N} F(s)\, e^{2\pi i (x-1)\frac{(s-1)}{N}}$$

The **autocorr** task performs auto-correlation, defined by:

$$\frac{1}{N}\sum_{u=1}^{N} f\times(u) f(x+u-1)$$

The **crosscorr** task performs cross-correlation defined by:

$$\frac{1}{N}\sum_{u=1}^{N} f\times(u)\, g(x+u-1)$$

The **fconvolve** task performs convolution, as defined by:

$$\frac{1}{N}\sum_{u=1}^{N} f(u)\, g(x-u+1)$$

The **powerspec** task computes a power spectrum, defined by:

$$P(s) = \frac{1}{N} F(s) F^{*}(s)$$

A frequently asked question about tasks in the **fourier** package is, "Which pixel receives the DC signal when taking a forward transform or power spectrum? That is, which pixel is zero frequency?" Zero frequency is at the reference pixel, which is given by the value of the image header keyword CRPIX1 (or CRPIX1 and CRPIX2 for a two-dimensional image). This will be pixel 1 (or [1,1] for two-dimensional) if the `center` parameter was set to "no".

The coordinate parameters of the input image are used to compute appropriate coordinate parameters for the output image. This makes it straightforward to associate a "world coordinate" value with any pixel in the output image. The coordinates can be seen using **igi** or **splot** for a one-dimensional image. For two-dimensional images, the image can be displayed using SAOimage, and then coordinates of features can be found using the **rimcursor** task with the parameter setting `wcs="world"`. For an image produced by the **forward** or **powerspec** task, the world coordinate at a pixel is the frequency in physical units, such as cycles per second. As an example, consider an image N pixels long with pixel spacing of D seconds per pixel. The pixel spacing in the Fourier domain is $1 / (N \times D)$ Hz per pixel. Suppose the input image contains a periodic signal with period P pixels, which is $P \times D$ seconds. The periodic signal results in a peak in the Fourier domain at N / P pixels from the reference pixel. The world coordinate (frequency in Hertz, in this example) at a pixel is the offset from the reference pixel multiplied by the pixel spacing, or $(N / P) \times 1 / (N \times D) = 1 / (P \times D)$ Hz, as you would expect for a period of $P \times D$ seconds.

For a two-dimensional image the pixel spacing changes independently in each dimension. If the length and pixel spacing are $N1$ and $D1$ for the first image axis and $N2$ and $D2$ for the second image axis, then the pixel spacing in the Fourier domain is $1 / (N1 \times D1)$ for the first axis and $1 / (N2 \times D2)$ for the second axis. In general, therefore, the coordinate axes will not be orthogonal in the Fourier domain. Tasks such as **forward**, **inverse**, and **powerspec** compute the output CD matrix by multiplying the input CD matrix on the right by the following matrix:

$$\begin{bmatrix} \dfrac{1}{N1 \times D1^2} & 0 \\ 0 & \dfrac{1}{N2 \times D2^2} \end{bmatrix}$$

The interpretation of a power spectrum deserves some discussion. The power spectrum shows the variance of the input data. While the total variance is just the average of the squares of the data values, the power spectrum tells whether the variance is due primarily to low frequencies or high frequencies or what combination. The integral of the power spectrum over a given interval is the contribution to the variance attributed to frequencies within that interval. For discrete data, we can interpret that principle as follows. The sum of the squares of the pixel values of the input data is equal to the sum of all the pixel values of the power spectrum. (This includes *negative* frequencies.) So the contribution of a range of frequencies to the variance of the input data is the sum of the power spectrum over the range of pixel numbers corresponding to those frequencies, divided by the total number of pixels. We can translate between pixel number P and frequency F in the usual way using the coordinate parameters in the power spectrum, e.g., for one-dimensional data, F = `CD1_1` × (P – `CRPIX1`). The value of the power spectrum at zero frequency (i.e., at the reference pixel) is the square of the average of the input data multiplied by the number of pixels.

When the input data contain a periodic term of sufficient amplitude, it will show up as a feature in the power spectrum. The amplitude of that periodic term can be obtained by adding up the counts in the power spectrum over the feature, as follows:

$$\text{amplitude in time series} = 2 \times \sqrt{\left(\frac{\text{sum of counts}}{N}\right)}$$

where N is the number of pixels in the input data. The factor of two accounts for the mirror image feature at negative frequency, so the sum of counts is taken only over the feature at positive frequency. Of course, when taking the sum of counts over the feature, the background level of the power spectrum in the neighborhood of the feature should first be subtracted.

The units of a power spectrum can be determined from the units of the input data. An integral over a portion of a power spectrum, as described above, will have the same units as the variance of the input data, e.g., (counts/pixel)^2. In the particular case of a one-dimensional time series with 0.2 second per pixel, if one of the input data values is 20 (counts/pixel), that value could also be regarded as 100 counts/sec. So if the variance has been computed in units of $(\text{counts/pixel})^2$ by integrating over the power spectrum, the variance may be converted to $(\text{counts/sec})^2$ by dividing by $(0.2)^2$.

The Guide Star Catalog and Plate Solutions

One remarkable product of the Space Telescope project is the Guide Star Catalog (GSC). The GSC contains positions for some 20 million celestial objects, accurate to a few tenths of an arcsecond and about half a magnitude. This catalog is published on two compact disks, and can be read with software in the **gasp** package in STSDAS.

The STSDAS **gasp** package is a subset of the GASP (Guide Star Selection System Astrometric Support Package) software available at STScI. To use the **gasp** package, you need to have a CD-ROM drive available and a copy of the GSC. Because CD-ROMs are not quite standard devices on all computer systems, we only support access to the CD-ROM device on Sun systems at this time. Certain tasks in the **gasp** package do not use the GSC CD-ROM directly, and can be used on any system.

Tasks in **gasp** can be characterized as either low or high level. That is, their function is either specialized (low level) or several such specialized tasks are brought together in a higher level task. For example, **sgscind** searches an index table of the coordinates of the guide star regions, listing those regions that overlap the user's field, **extgst** extracts only the stars actually within the user's field from a list of regions, and **fttostt** converts one or more FITS format tables to STSDAS format tables. The higher level tasks are **regions**, which creates a set of STSDAS tables containing the guide star regions that overlap the user's field, and **targets**, which includes the functions of **regions**, and adds the functionality of **intrep**, which allows the user to determine a plate solution and obtain guide star coordinates for any arbitrary position on an image.

The images and the catalog coordinates interact with each other in this package. In some tasks, it is necessary to know the transformation between the world (celestial) and pixel coordinates.

To understand what the **gasp** tasks do, it is useful to know a bit about the catalog. The Guide Star Catalog is distributed as a set of two compact disks. The catalog is subdivided into 9537 regions, bounded by circles of right ascension and declination. All regions subtend 7.5 degrees of declination and varying widths in right ascension. Data for each region are stored as separate files organized into subdirectories. Each subdirectory contains all of the regions for one 7.5 degree wide zone of declination. The region files are distributed between the two disks, divided at a declination of -7.5 degrees. The description files and supporting tables are duplicated on both disks. All region files are in FITS table format. One of the supporting files is a FITS table giving the boundary coordinates of each of the regions. Searching this table allows a particular region number to be determined.

The **gasp** package includes two general purpose tasks, **pltsol** and **intrep**. These provide a general-purpose facility for determining plate solutions and transforming between plate and sky coordinates.

Isophote Fitting

The STSDAS **isophote** package contains tasks for elliptical isophote analysis. The main task is **ellipse**, which fits a set of elliptical isophotes over an image. The algorithm is closely based on the description given by Jedrzejewski (*Mon.Not.R.Astr.Soc.*, **226**, 747, 1987). The results of the isophote fitting can be written to an STSDAS table or to ASCII text tables. Table 5.2 lists the tasks available in the **isophote** package.

Task	*Purpose*
bmodel	Build a model image from isophotal analysis data
elapert	Create polygon files for use in apphot polygonal photometry
ellipse	Fit elliptical isophotes to image data
isoimap	Graph ellipses superposed over grayscale image display
isomap	Graph ellipses superposed over image contours
isopall	Graph summary of all results from isophotal analysis
isoplot	Graph results of isophotal analysis

Table 5.2: Tasks in isophote Package

The output table from the **ellipse** task, containing isophote information, is used as input to the remaining tasks in this package. The task **bmodel** reads the table and builds an artificial image with a photometric model of the original image. Task **elapert** is used to build a set of elliptical apertures, to be used with the **polyphot** task in the **noao.digiphot.apphot** package, for aperture photometry.

Task **isoplot** can be used to quickly examine, in graphic form, the run of any isophote parameter against ellipse semi-major axis. Task **isopall** builds a graphic mosaic simultaneously showing the run of nine isophote parameters against ellipse semi-major axis. Task **isomap** draws a contour map of the original image, and superposes on it the ellipses corresponding to the contoured levels. In a similar way, task **isoimap** superposes ellipses on the grayscale image display.

To use the **ellipse** task, the user specifies values for the semi-major axis length (`a0`), the X and Y axis center (`x0` and `y0`), the ellipticity (`eps0`), and the position angle of the semi-major axis (`teta0`). These parameters can be set in several ways:

- By setting the `a0`, `x0`, `y0`, `eps0`, and `teta0` parameters before running the task.
- By four X,Y coordinate pairs, read from a text file, which correspond to the extremities of the major and minor axes of the first-guess ellipse.
- Interactively using cursor keystrokes and commands.
- If none of the above are used, the task will prompt you for values.

Here's an example of how these parameters can be specified when running the task interactively. First, display the desired image in the image display window (i.e., using SAOImage or imtool in conjunction with the **tv.display** task). Now start the ellipse task using the parameter settings shown in Figure 5.7.

Except for these Values, Parameters are Set to Default Values

```
input   =         my_image.hhh  input image name
output  =          iso_fit.tab  output table name
(sdas   =                 yes)  output table will have sdas format ?
(interac=                 yes)  set fitting parameters interactively?
(device =                imdy)  graphics output device
(imcur  =                    )  image cursor input
(gcur   =                    )  graphics cursor input
(initfil=                    )  file with initial ellipse points
(samplef=                    )  files with intensity samples
(badpixe=                    )  bad pixel list
(harmoni=                    )  optional harmonic numbers
(x0     =               INDEF)  initial ellipse center
(y0     =               INDEF)
(eps0   =               INDEF)  initial ellipticity
(teta0  =               INDEF)  initial position angle
(hcenter=                  no)  hold center fixed ?
(heps   =                  no)  hold ellipticity fixed ?
(hteta  =                  no)  hold position angle fixed ?
a0      =                 10.   initial semi-major axis
mina    =                  1.   minimum semi-major axis
maxa    =                 18.   maximum semi-major axis
(astep  =                  1.)  step between successive ellipses
(linear =                 yes)  linear astep ?
(minit  =                   8)  minimun no. of iterations
(maxit  =                  50)  maximun no. of iterations
(lslope =                  0.)  limit for acceptable slope
(conver =                  0.)  convergency sensitivity control
(clip   =                  0.)  fraction of points to clip off
(maxrit =               INDEF)  max. semi-major axis for iterative mode
(integrm=           bi-linear)  area integration mode
(mag0   =                22.5)  magnitude of reference source
(refer  =                  1.)  intensity of reference source
(backgr =                  0.)  intensity of background signal
(star   =                  0.)  star removal radius
```

Figure 5.7: Sample Parameter Values

When the task starts up you will be left in graphics cursor mode, at which point you can set the necessary first-guess values for the ellipse parameters. First, position the cursor in the image display at the desired center location for the first ellipse and then press X. This will define parameters `x0` and `y0`. You can then either position the cursor at the desired starting radius and position angle and press A to redefine the `a0` and `teta` parameters, or you can enter a value from the keyboard by typing `:teta 0.0`, for example. Finally, the initial value for `eps` is defined by entering a value from the keyboard, such as `:eps 0.1`. Before attempting the fit for the first ellipse you can review the current parameter settings by typing `:show`. If none of the displayed parameters have values of "INDEF" you are ready to proceed with the fitting.

To proceed with the fitting, you can press F to compute a fit for the current isophote, press N to proceed to the next isophote, or press C to continue the fitting in non-interactive mode.

Using the initial ellipse values, the image is sampled along a first-guess elliptical path, producing a one-dimensional intensity distribution as a function of the ellipse eccentric angle. The harmonic content of this distribution is analyzed by fitting it to the following least squares function:

$$y = y0 + A1 \times \sin(E) + B1 \times \cos(E) + A2 \times \sin(2'E) + B2 \times \cos(2'E)$$

From the harmonic amplitudes (A1, B1, A2, B2), and from the local image gradient, correction factors can be derived for each sampling ellipse parameter (keeping `a0` fixed). The parameter that corresponds to the largest amplitude is varied, and the image is resampled. This process is repeated until any one of the following criteria are met:

1 • The largest harmonic amplitude is less than the internal variable `crit` times the rms residual of the intensity data around the harmonic fit. (The value of `crit` is defined below).

2 • A user-specified maximum number of iterations, `maxit`, is reached.

3 • More than 50% of the sampled points have the value INDEF. This usually happens because points on the elliptical sampling path that lie outside the image boundaries are given the value INDEF. However, badly flagged regions of an image can also cause the iteration loop to halt prematurely.

In any case, a minimum of `minit` iterations are performed. If the iterations stop because of reasons 2 or 3 above, then those ellipse parameters that generated the lowest absolute values for harmonic amplitude will be used. At this point, the image data sample coming from the best fit ellipse is fitted by the following function:

$$y = y0 + An \times \sin(n \times 'E) + Bn \times \cos(n \times E)$$

where $n = 3$ and $n = 4$. The amplitudes (A3, B3, A4, B4), when divided by the semi-major axis length and local intensity gradient, measure the isophote's deviations from perfect ellipticity—these parameters, all other associated isophote parameters, and some auxiliary information, are written to the output file.

After fitting the ellipse that corresponds to a given value of the semi-major axis (by the process described above), the axis value is increased by a factor of (1. + `astep`), and the process is repeated until either the semi-major axis value reaches the value `maxa`, or the last fitted ellipse has more than 50% of its sampled points flagged with INDEF values. The process then resumes from the first fitted ellipse (`a0`) inwards, in steps of (1./(1. + `astep`)), until the semi-major axis length reaches the value `mina`. At each step, the starting ellipse parameters are taken from the previously fitted ellipse that has the closest semi-major axis length.

On low surface-brightness regions (i.e., those having large radii), the small values of the image radial gradient can result in large corrections and meaningless values for the ellipse parameters. In this case, the iterative mode is disabled and geometric parameters keep their last stable values. This non-iterative mode is entered when one of the following criteria are met:

1 • The absolute value of the image radial gradient is less than `lslope` times the rms along the isophote.

2 • The ellipticity exceeds 1.0.

3 • The ellipse center crosses the image boundaries.

4 • The semi-major axis exceeds `maxrit`; `maxrit` can be ignored if its value is set to zero or INDEF.

Three methods are available for sampling an image around the elliptical path: bi-linear interpolation, and either mean or median over elliptical annulus sectors. These modes are selected using the task parameter `integrmode`. Bi-linear interpolation extracts a 1-pixel width sample from the image. When using this technique, and when the distance between successive ellipses exceeds 2 pixels, many pixels will never be sampled. Integration over elliptical annuli can be used, however, to extract all information from the image array. The width of a given annulus is set from halfway between the present ellipse and the previous one to halfway between the present ellipse and the next one. Annuli are divided in sectors, and the angular step between successive sectors makes their areas approximately constant over a given annulus. The angular span of a given sector is, however, restricted to a maximum of 12 degrees. Individual image pixels are considered to be inside a sector if more than half of the pixel area is inside that sector, otherwise they are not counted. The resulting sample point for that particular sector will be the arithmetic mean or the median of all the non-INDEF pixels inside the sector. The percentage difference between the true geometric area of the sector and the image area effectively sampled in that sector is a measure of the sampling technique geometric error. The average, along the elliptical path, of its absolute value for each sector, is output for each fitted isophote.

If any sector vertex is outside the image boundary, its corresponding sample point is flagged as INDEF. The value of `integrmode` is ignored when the distance between the present ellipse and the next ellipse is less than 2 pixels, or when the semi-major axis is less than 20 pixels; in these cases, bi-linear interpolation is always used. The mean and median methods execute 3-5 times slower than bi-linear interpolation.

The basic convergency criterion (i.e., the `crit` variable), is defined as follows. In the case of bi-linear interpolation, it has a fixed value of 4%. In the case of either mean or median over elliptical annulus sectors, however, the increased number of pixels counted inside each sector as the algorithm proceeds outwards, will tend to decrease the rms residual against which harmonic amplitudes are compared. This will, on one side, increase the sensitivity to low-amplitude harmonics, and on the other side will increase the number of iterations necessary for the criterion to be met. The task parameter `conver` can be used to decrease the algorithm sensitivity in the outer regions of images, where sectors are large. The value of `crit` is defined as:

$$crit = 0.04 \times \sqrt{S} \times CONVER$$

where S is the sector area (kept approximately constant along the ellipse) in units of pixel^2. Thus `conver=1.0` should, in principle, cancel the effect of the smaller rms residual that is caused by integration over larger areas, when compared with bi-linear interpolation. This will, however, retain all data for the full image area in the resulting sample. Larger values of `conver` will relax the criterion while smaller values will tighten it. If `conver = 0.0`, the `crit` parameter will be fixed at 4% regardless of the integration method used.

Errors for all parameters are obtained directly from the rms scatter of intensity data along the fitted ellipse.

For the sake of convenience, task output includes semi-major axis length raised to the 1/4 power, as well as isophotal intensities and errors expressed in a rough magnitude scale. This scale is derived from the parameters `mag0`, `refer`, and `dark`, and from the mean isophotal intensity `int`, according to the following relation:

$$m = \mathrm{mag0} - 2.5 \times \log 10\left(\frac{(\mathrm{int} - \mathrm{dark})}{(\mathrm{refer\text{-}dark})}\right)$$

Output directed to STDOUT is a table with one row for each semi-major axis length. Each row contains the following data: semi-major axis length, mean isophotal intensity and its rms, ellipticity and its error, position angle and its error, number of valid data points used in the fit, number of flagged data points extracted from the image, number of iterations, and stop condition code. The stop code can have the values listed in Table 5.3.

Stop Code	*Meaning*
`0`	Normal
`1`	Less than 50% of extracted data points are valid
`2`	Exceeded `maxit`
`3`	Singular matrix in harmonic fit, results may not be valid
`4`	Small slope, or ellipse diverged. All subsequent ellipses at larger radii have the same constant geometric parameters

Table 5.3: Stop Condition Codes

The last column contains the average, along the elliptical path, of the absolute value of the error (in percent) between the true geometric area of each sector and the image area effectively used for that sector.

If the output is in the form of an STSDAS table, it will contain one row for each value of the semi-major axis. The labelling of each column is described in Table 5.4.

Label	*Contents*
`a`	Semi-major axis length (pixel units)
`meanint`	Mean isophotal intensity
`rms`	Scatter around isophotal intensity
`eps`	Ellipticity
`eeps`	Ellipticity error
`teta`	Position angle (degrees)
`eteta`	Error in position angle
`x0, y0`	Ellipse center (pixel units)
`ex, ey`	Error in ellipse center
`slope`	Local radial intensity gradient
`eslope`	Error in gradient
`a**1/4`	(Semi-major axis length)**1/4
`mag`	Mean isophotal magnitudes
`em1, emu`	Lower and upper magnitude errors
`A3, B3`	Third harmonic deviations from an ellipse
`A4, B4`	Fourth harmonic deviations from an ellipse
`aA3, eB3`	Errors in third harmonic deviations
`eA4, eB4`	Errors in fourth harmonic deviations
`ndata`	Number of valid data points in ellipse
`nflag`	Number of flagged data points
`niter`	Number of iterations
`stop`	Stop condition code
`bigA`	Maximum (absolute value) among first and second harmonic amplitudes
`se`	Sector area averaged error

Table 5.4: Column Labels

If the main output is in the form of character-oriented tables, their names will be constructed by concatenating the string passed to output with suffixes 1, 2, 3 and 4. Their contents will be as follows:

File 1: Semi-major axis length, mean intensity of isophote and its rms, ellipticity and its error, position angle and its error, maximum among first and second harmonic amplitudes.

File 2: Semi-major axis length, coordinates of the center and their errors, gradient of image and its error, number of valid data points used in the fit, number of flagged data points extracted from the image, number of iterations, and stop condition code.

File 3: Semi-major axis length, higher order harmonic deviations from a perfect ellipse (n = 3 and 4) and their errors.

File 4: Semi-major axis length, (semi-major axis length)**(1/4), isophotal magnitude, lower and upper magnitude error bars, sector area averaged error.

The user can generate a set of files containing the intensity sample from the image corresponding to a fitted isophote. These are ASCII text files and are created only when a value is passed to the `samplef` parameter. The files will have the root name specified by `samplef` and the extensions `.001`, `.002`, `.003`, etc. Each file will contain one sample point (eccentric angle, intensity pair) per line, preceded by comment lines with identification data. These files are compatible with the **sgraph** task, allowing isophotal data to be quickly examined using graphics.

Figure 5.8 shows a subset of output table columns for the example **ellipse** run described earlier. The **sgraph** task can be used to easily examine these data, for example, the following command will produce a plot of position angle as a function of radius:

```
cl> sgraph "iso_fit.tab a teta"
```

Similarly, the following command will produce a plot of mean isophotal surface brightness (magnitudes per unit area) as a function of radius to the 1/4 power (Figure 5.9).

```
cl> sgraph "iso_fit.tab a**1/4 mag" yflip+ point+
```

Table iso_fit.tab Mon 11:32:35 21-Feb-94

a	a**1/4	x0	y0	meanint	mag	eps	teta
pixel	pixel**1/4	pixel	pixel				degrees
1.00	1.00000	30.05	33.95	1478.	14.6	0.0690	1.29
2.00	1.18921	30.02	33.84	770.	15.3	0.0690	-67.62
3.00	1.31607	30.04	33.70	398.	16.	0.0612	-67.62
4.00	1.41421	30.02	33.73	260.	16.5	0.0976	-48.33
5.00	1.49535	30.03	33.69	191.	16.8	0.1450	-50.65
6.00	1.56508	29.98	33.75	142.	17.1	0.1473	-51.34
7.00	1.62658	29.87	33.77	110.	17.4	0.1424	-50.05
8.00	1.68179	29.86	33.28	87.6	17.6	0.1277	-41.92
9.00	1.73205	30.04	33.49	72.1	17.9	0.1414	-28.30
10.00	1.77828	30.31	33.62	61.1	18.	0.1564	-43.41
11.00	1.82116	30.18	33.53	53.6	18.2	0.1934	-45.21
12.00	1.86121	30.13	33.18	43.8	18.4	0.1738	-49.85
13.00	1.89883	29.94	32.86	33.4	18.7	0.1265	-61.13
14.00	1.93434	30.23	33.08	29.8	18.8	0.1623	-55.11
15.00	1.96799	30.16	33.13	23.5	19.1	0.1408	-55.11
16.00	2.00000	30.16	32.68	19.6	19.3	0.1408	-43.31
17.00	2.03054	29.68	32.17	15.4	19.5	0.1408	-43.45
18.00	2.05977	29.40	33.05	13.2	19.7	0.1408	-43.45

Figure 5.8: Subset of Table Output from ellipse Task

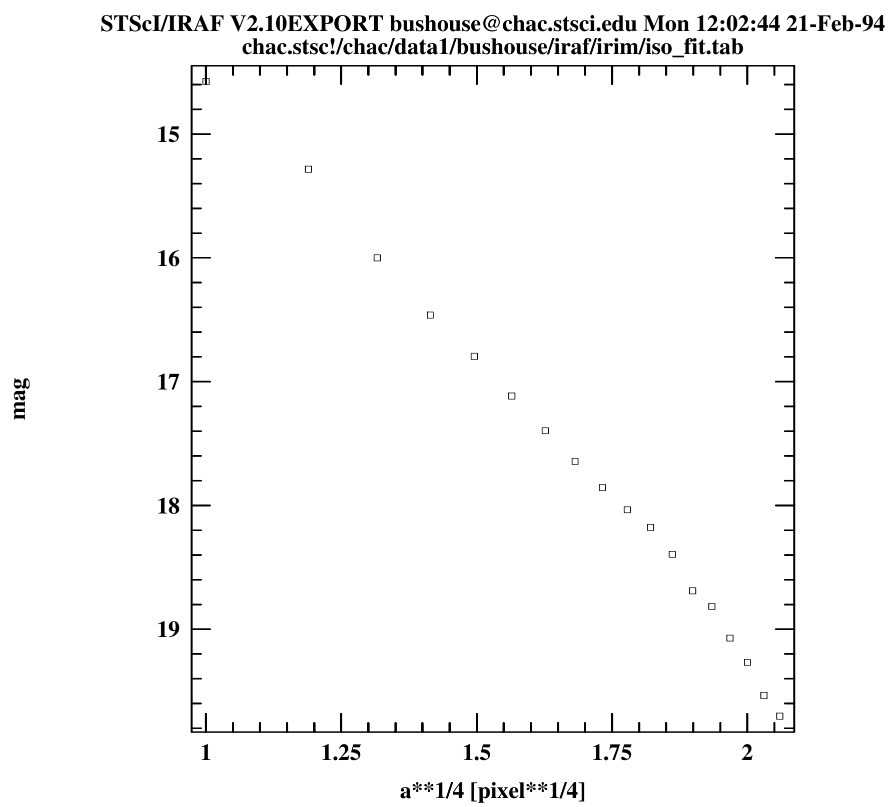

Figure 5.9: Plot of Isophotal Surface Brightness

Statistics

The STSDAS **statistics** package is a direct implementation of the survival analysis package *ASURV* developed by Eric Feigelson and Takashi Isobe at Pennsylvania State University.[2]

Observational astronomers frequently encounter the situation where they observe a particular property (e.g., far infrared emission, calcium line absorption, or CO emission) of a previously defined sample of objects, but fail to detect all of them. The data then contains upper limits as well as detections, preventing the use of simple and familiar statistical techniques in the analysis. Statistical quantities frequently sought include the mean value of the measured variable for the sample, its distribution function, comparison of this distribution with other samples, and the relation of the measured variable to other properties of the objects.

Astronomers recently recognized the existence of a variety of statistical methods, or have derived similar methods on their own, to deal with these problems. The methods are frequently collectively called “survival analysis” or the “analysis of lifetime data,” from their origins in actuarial and related fields. This revision of the survival analysis package provides all of the methods described in the papers in the *Astrophysical Journal* (Feigelson and Nelson, 1985; Isobe, Feigelson and Nelson, 1986).

No statistical procedure can magically recover information that was never measured at the telescope. However, there is frequently important information implicit in the failure to detect some objects which can be partially recovered under reasonable assumptions. We purposely provide several two-sample tests, correlation tests and linear regressions—each based on different models of where upper limits truly lie—so that the astronomer can judge the importance of the different assumptions.

There are reasons for confidence in the methods. First, the volume of scholarly effort devoted to the problem is impressive. A dozen books and scores of papers have been published during the last decade. Second, enterprises involving thousands of lives and billions of dollars (such as the insurance industry) appear to believe in the methods. Third, Monte Carlo simulations of a typical astronomical problem (flux-limited satellite observations of a magnitude limited population of uniformly distributed objects) showed several methods to be quite reliable in this context (Isobe et al., 1986). More simulations of astronomical problems are needed.

2. Detailed references for this section are listed on page 209.

There are also reasons for not applying these methods. We describe some of their known limitations in section about cautions and caveats.

Overview of Survival Analysis

Statistical methods dealing with censored data have a long and confusing history. It began in the 17th century with the development of actuarial mathematics for the life insurance industry and demographic studies. Astronomer Edmund Halley was one of the founders. In the mid-20th century, this application grew, along with biomedical and clinical research, into a major field of biometrics. Similar (and sometimes identical) statistical methods were being developed in two other fields: the theory of reliability, with industrial and engineering applications; and econometrics, with attempts to understand the relations between economic forces in groups of people. Finally, we find the same mathematical problems and some of the same solutions arising in modern astronomy as discussed above.

Let us give an illustration on the use of survival analysis in three disparate fields. Consider a linear regression problem. First, an epidemiologist wishes to determine how the longevity or "survival time" (the dependent variable) depends on the number of cigarettes smoked per day (the independent variable). The experiment lasts 10 years, during which some individuals die and others do not. The survival time of the living individuals is only known to be greater than their age when the experiment ends. These values are therefore "right censored data points." Second, a company manufacturing engines wishes to know the average time between breakdowns, as a function of engine speed, to determine the optimal operating range. Test engines are set running until 20% of them fail; the average "lifetime" and dependence on speed is then calculated with 80% of the data points censored. Third, an astronomer observes a sample of nearby galaxies in the far-infrared to determine the relation between dust and molecular gas. Half have detected infrared luminosities but half are upper limits ("left censored data points"). The infrared luminosities are compared to CO observations, which themselves may contain censored values.

Astronomers have dealt with their upper limits in a number of fashions. One is to consider only detections in the analysis; while possibly acceptable for some purposes (e.g., regression), this will clearly bias the results in others (e.g., estimating luminosity functions). A second procedure considers the ratio of detected objects to observed objects in a given sample. When plotted in a range of luminosity bins, this has been called the "fractional luminosity function" and has been frequently used in extragalactic radio astronomy. This is mathematically the same as actuarial life tables. But it is sometimes used incorrectly in comparing different samples: the detected fraction clearly depends on the (usually uninteresting) distance to the objects as well as their (interesting) luminosity. Also, error bars do not apply in these fractional luminosity functions.

A third procedure is to take all of the data, including the exact values of the upper limits, and model the properties of the parent population under certain mathematical constraints, such as maximizing the likelihood that the data fits the model. This technique, which is at the root of much of modern survival analysis, was independently developed by astrophysicist Yoram Avni (Avni et al., 1980; Avni and Tananbaum, 1986) and is now commonly used in X-ray astronomy.

Survival methods have the advantage of having been widely-used in a range of disparate fields so that their mathematical validity has been tested in other disciplines. Issues of great social importance (e.g., establishing Social Security benefits, strategies for dealing with the AIDS epidemic, MIL-STD reliability standards) are based on it. In detail, however, astronomers must study the assumptions underlying each model and be aware that some methods at the forefront of survival analysis may not be fully understood.

Overview of the Software

The statistical methods for dealing with censored data might be divided into a 2 x 2 grid: parametric vs. nonparametric, and univariate vs. multivariate. Parametric methods assume that the censored survival times are drawn from a parent population with a known distribution function (e.g., Gaussian, exponential), and is the principal assumption of industrial reliability models. Nonparametric models make no assumption about the parent population, and frequently rely on maximum-likelihood techniques. Univariate methods are devoted to determining the characteristics of the population from which the censored sample is drawn, and comparing such characteristics for two or more censored samples. Bivariate methods measure whether the censored property of the same is correlated with another property of the objects and, if so, to perform a regression that quantifies the relationship between the two variables. There are few truly multivariate methods for censored data.

We will concentrate on the nonparametric models because the underlying distribution of astronomical populations is usually unknown. Some methods (e.g., EM algorithm regression), however, are fully parametric while others (e.g., Cox regression, Buckley-James regression) are partially parametric. Most of the methods are derived and described in more detail in Schmitt (1985), Feigelson and Nelson (1985) and Isobe et al. (1986).

Two tasks are univariate in nature: **kmestimate** and **twosampt**. **kmestimate** gives the Kaplan-Meier estimator for the distribution function of a randomly censored sample. First derived in 1958, it lies at the root of non-parametric survival analysis. It is the unique, self-consistent, generalized maximum-likelihood estimator for the population from which the sample was drawn. When formulated in cumulative form, it has analytic asymptotic (for large N) error bars. The median is always well-defined (but not the mean when the lowest point is also the upper limit). It is identical to the differential "redistribute-to-the-right" algorithm, independently derived by Avni et al. (1980), but when formulated in integral form has simple analytic error analysis.

The second univariate task is **twosampt**, giving the variety of ways to test whether two censored samples are drawn from the same parent population. They are mostly generalizations of standard tests for uncensored data, such as the Wilcoxon and logrank nonparametric two-sample tests. They differ in how the censored data are weighted or "scored" in calculating the statistic. Thus, each is more sensitive to differences at one end or the other of the distribution. The Gehan and logrank tests are widely used in biometrics, while some of the others are not.

There are a number of bivariate methods, including two correlation tests and three linear regression analyses. Cox hazard model (correlation test), EM algorithm, and Buckley-James method (linear regressions) can treat several independent variables if the dependent variable contains only one kind of censoring (i.e., upper or lower limits). Generalized Kendall's tau (correlation test) and Schmitt's binned linear regression can treat mixed censoring (including censoring in the independent variable), but can have only one independent variable. correlation test We currently do not provide error analysis for Schmitt's method.

Cautions and Caveats

The Kaplan-Meier estimator works with any underlying distribution (e.g., Gaussian, power law, bimodal), but only if the censoring is "random." That is, the probability that the measurement of an object is censored can not depend on the value of the censored variable. At first glance, this may seem to be inapplicable to most astronomical problems: we detect the brighter objects in a sample, so the distribution of upper limits always depends on brightness. However, two factors often serve to randomize the censoring distribution. First, the censored variable may not be correlated with the variable by which the sample was initially identified. Thus, infrared observations of a sample of radio bright objects will be randomly censored if the radio and infrared emission are unrelated to each other. Second, astronomical objects in a sample usually lie at different distances, so that brighter objects are not always the most luminous. In cases where the variable of interest is censored at a particular value (e.g., flux, spectral line equivalent width, stellar rotation rate) rather than randomly censored, then parametric methods appropriate to "Type 1" censoring should be used. These are described by Lawless (1982) and Schneider (1986), but are not included in this package.

Thus, the censoring mechanisms of each study must be understood individually to judge whether the censoring is or is not likely to be random. The appearance of the data, even if the upper limits are clustered at one end of the distribution, is not a reliable measure. A frequent (if philosophically distasteful) way to avoid having to determine the nature of the censoring is to define the population of interest to be the observed sample. The Kaplan-Meier estimator then always gives a valid redistribution of the upper limits, though the result may not be applicable in wider contexts.

The two-sample tests are somewhat less restrictive than the Kaplan-Meier estimator, since they seek only to compare two samples rather than determine the true underlying distribution. The Gehan test assumes that the censoring patterns of the two samples are the same, but does not require that they be random. The logrank test results appear to be correct even when the censoring patterns differ somewhat. There is little known about the limitations of the other two-sample tests given here or the bivariate generalized Kendall's tau rank statistic.

The two bivariate correlation coefficients, generalized Kendall's tau and Cox regression, are both known to be inaccurate when many tied values are present in the data. Ties are particularly common when the data is binned. Caution should be used in these cases.

Cox regression, though widely used in biostatistical applications, formally applies only if the "hazard rate" is constant; that is, the cumulative distribution function of the censored variable falls exponentially with increasing values. Astronomical luminosity functions, in contrast, are frequently modelled by power law distributions. It is not clear whether a Cox regression is significantly affected by this difference.

There are several limitations to the three linear regression methods—**emmethod**, **buckleyjames**, and **schmittbin**—presented here. First, only **schmittbin** works when censoring is present in both variables. Second, **emmethod** requires that the residuals about the fitted line follow a Gaussian distribution. **buckleyjames** and **schmittbin** are less restrictive, requiring only that the censoring distribution about the fitted line is random. Both we and Schneider (1986) find little difference in the regression coefficients derived from these two methods. Third, there is considerable uncertainty regarding the error analysis of the regression coefficients of all three models. **emmethod** gives analytic formulas based on asymptotic normality, while Avni and Tananbaum (1986) numerically calculate and examine the likelihood surface. **buckleyjames** gives an analytic formula for the slope only, but it lies on a weak theoretical foundation. We do not provide Schmitt's (1985) bootstrap error analysis for schmittbin, partly because of the high computational expense for large data sets. Users may thus wish to run all methods and evaluate the uncertainty with caution. Fourth, use of **schmittbin** is questionable in certain cases with heavy censoring. The user must arbitrarily choose a bin size. If it is too small, many censored points at the end of the distribution will be changed to detected points. If the bins are too large, accuracy in the regression calculation is reduced. Fifth, the Buckley-James method has a defect in that the final solution occasionally oscillates rather than converging smoothly.

If we consider the field of survival analysis from a broader perspective, we note a number of deficiencies with respect to censored statistical problems in astronomy. Most importantly, survival analysis assumes the upper limits in a given experiment are precisely known, while in astronomy they frequently represent n times the RMS noise level in the experimental detector, where n=2, 3, 5, or whatever. Methods adopted to this situation are clearly needed. A related deficiency is the absence of weighted means or regressions. Survival analysis also has not yet produced any true multi-variate techniques, such as a Principal Components Analysis that permits censoring.

Finally, we note that this package is not unique. Nearly a dozen software packages for analyzing censored data have been developed (Wagner and Meeker 1985). Four are part of large multi-purpose commercial statistical software systems: SAS, SPSS, BMDP, and GLIM. These packages are available on many university mainframes. We have found BMDP to be the most useful for astronomers (see the appendixes in Feigelson and Nelson 1985; Isobe et al. 1986 for instructions on its use). It provides most of the functions in **kmestimate** and **twosampt** as well as a full Cox regression, but no linear regressions. Other packages (CENSOR, DASH, LIMDEP, STATPAC, STAR, SURVAN, SURVREG) were written at various universities, medical institutes and industrial labs, and have not been evaluated.

Timeseries

The **timeseries** package contains tasks for reducing one-dimensional time series data. While these tasks were written to reduce data obtained by the High Speed Photometer (HSP) aboard the Hubble Space Telescope, some tasks may also be used as general photometric reduction tasks. Each **timeseries** task is described below. Details can be found in the online help files for each task.

coadd: This task will co-add files containing observed count rates as a function of phase (for periodic variable stars, for example). This task is usually used in conjunction with the **orderphase** task. After sorting a time series observation according to phase (using the **orderphase** task), you can use **coadd** to get a smoother light curve.

delaytime: Calculate the difference in light travel time between the space telescope and the solar system barycenter. This task also calculates: (1) Earth to solar system barycenter (maximum of about 500 seconds), (2) geocentric distance to the spacecraft (maximum ~0.023 second), (3) parallax (maximum ~1.2 millisecond per arcsecond of parallax), (4) proper motion of the target (maximum 2.4 milliseconds per arcsecond of proper motion), and (5) relativistic effect due to time dilation caused by Earth's orbital velocity changes and gravitational red shift caused by Earth's orbital heliocentric distance changes (maximum ~1.7 millisecond).

orderphase: Sort the pixels in a file by ascending phase (for periodic variable stars, for example). This task needs input of the variability period and the zero phase epoch. A single-cycle light curve is produced by reordering all points according to their fractional phase. The output from this task can be used as a single-cycle light curve to evaluate the choice of period or to remove discrepant data. Before using this task, you may need to correct the input data for light delay time (using **delaytime** task), especially if the light delay time is comparable to the variability period.

filterphot: Convert count rates in the input time series to magnitudes and colors, and optionally transform to a standard filter photometry system. Four options are available:

- Instrumental magnitudes can be computed from intensity measurements of two time series input files.
- Instrumental colors can be computed from intensity measurements of two time series input files.
- Standardized magnitudes can be calculated from instrumental magnitudes and appropriate standard colors, by using three transformation coefficients.
- Standardized colors can be calculated from instrumental colors and appropriate standard colors by using up to three transformation coefficients.

polarimetry: Reduce count rates obtained from light going through polaroid filters oriented at four different angles to Stokes parameters. The input for this task is four time series files, one for each of the polarizers oriented at 0°, 45°, 90°, and 135° of the same bandpass filter. The normalized Stokes parameters are computed from the four measurements and are transformed from the instrumental coordinate system to the equatorial coordinate system. The degree of linear polarization and its position angle are then calculated from the normalized Stokes parameters.

subsky: Subtract background flux from target+background flux. This task can handle several background subtraction schemes: (1) point-by-point subtraction, (2) overall fit of background data by using a low-order polynomial function, (3) background flux calculated by averaging several background points closest to the time of target observation.

Other STSDAS Packages

Some STSDAS packages that are still in development contain useful tasks that may be robust enough for some purposes. Because these packages continue to be developed, the descriptions contained in this manual are only brief synopses of their functions. Refer to the online help for more current summaries of the package capabilities. These packages include the **testdata** and **playpen** packages. You may also want to investigate the **contrib** package, which contains programs developed by users outside the STSDAS group.

There are also two other packages that may be of interest to some users, but that we do not necessarily recommend using: **sobsolete** and **vdisplay**. The **sobsolete** is where tasks that are about to be retired from STSDAS are stored. By putting these tasks in their own package, users become aware that a task's functionality has either been replaced by another task or is no longer deemed useful (type `help sobsolete opt=sys` for more information). If you need and use a task in the **sobsolete** package, you should notify the HotSeat staff to find out why the task is being retired.

The **vdisplay** package is provided to support Gould DeAnza image display devices on VAX systems. These devices are not often used any longer, and this package will eventually be retired.

Test Data

The STSDAS **testdata** package will generate simple test spectra. Task **compspec** creates a one-dimensional image using a user-specified algebraic expression to compute the pixel values. The task **overscan** modifies an existing spectrum to simulate the effect of overscanning with the Faint Object Spectrograph.

Playpen

Tasks or packages that are being developed are often placed in the **playpen** package so that users can exploit the latest software even before it is fully integrated into STSDAS. While **playpen** may rightly be considered a testbed for new utilities, the tasks themselves have undergone the same testing and validation as other STSDAS tasks. Users should simply be aware that the functionality, the user interface, the parameters, and even the task names themselves may change with little or no warning. As tasks or packages are fully developed, they may be moved elsewhere within STSDAS—for example, to the **analysis** package—but this will usually occur during major releases, not minor patch releases.

The **playpen** package currently contains a prototype of the **registration** package, which allows images to be aligned based on common features or on FITS-style World Coordinate information stored in the image headers. This package currently accommodates sets of images with different pixel scales, central positions, and rotations. Accommodation of more complicated transformations is planned.

The **newredshift** task is a handy utility for modifying the redshift of one-dimensional spectra. The task operates by modifying the dispersion relation, which is stored in various keywords in the image header, and can handle different input and output redshifts, as well as logarithmic wavelength scales.

The **hsubtract** and **fill** tasks implement the Baade-Lucy algorithms for dealing with sky background in direct images. This method is designed to take care of the spatial high-frequency component in the background. It fills *pits* (pixels below the average background) by local flux reshuffling. The **hsubtract** task subtracts a reference background, either from a reference image or a computed background. These tasks, along with the **edge** task for modifying image edges by windowing or overlapped edge extension, are useful for conditioning images before using the image restoration tasks.

The **lgrlist** task is an enhancement of the **stplot.grlist** task. It expands a list of image names, including a range of group numbers, into an explicit list of individual image names with an explicit section and group number. This input list of images can include wildcards, @-lists, etc.

Two HST-specific utilities are also included in the **playpen** package. The **jimage** task uses a list of U2-U3 pointing coordinates and creates an image that reflect the *jitter* information. The **hstpos** task plots the position of HST at specified times on a Mollweide projection of the world. The task uses information contained in the headers of uncalibrated, GEIS-format HST data, and is most useful for determining whether a given observation was likely to be affected by terminator crossings, or by the South Atlantic Anomaly. The **geo2mag** task for converting geocentric coordinates to and from geomagnetic coordinates, while not HST specific, is also useful for gaining insight about the quality of HST data.

User-Contributed Software: contrib

The **contrib** package contains tasks or small packages that are contributed from the HST user community. They are included in the STSDAS distribution as a service to our users, but the responsibility for the code, the accuracy of the output, and advice for using this software rest with the authors.

The **acoadd** task optimally coadds images with differing resolution using an algorithm developed by Leon Lucy and Richard Hook (rhook@eso.org) at the ST-ECF in Garching.

The **gaussfit** task is a general utility for solving least-squares and robust estimation problems. It includes a programming language and an algebraic manipulator for analytically calculating the required partial derivatives. While this version of the program is outdated, it still provides some functionality beyond that found in the **stsdas.fitting** package. A more recent version (which was developed for the PC and Macintosh environments) can be obtained from the University of Texas at Austin.

The **redshift** package contains two tasks for determining redshifts and dispersion relations, using either the Fourier Quotient technique (Sargent et al., 1977, *ApJ*, **212**, 326) or the cross-correlation technique (Tonry and Davis, 1979, *AJ*, **84**, 1511). This package was developed by Gerard Kriss at Johns Hopkins University (gak@perseus.pha.jhu.edu).

The **specfit** task is very useful for fitting multi-component models to one-dimensional spectra. The components include polynomial, blackbody, and powerlaw continua, and Gaussian, Lorentzian, and user-defined line profiles. Users can select either the downhill simplex or the Levenberg-Marquardt method of solution. The task was written by Gerard Kriss at Johns Hopkins University (gak@perseus.pha.jhu.edu).

Finally, the **vla** package contains three IRAF scripts to reduce radio or optical imaging spectroscopy *data cubes* into a set of two-dimensional images representing various moments of the data cube along the velocity axis. These moments include the total intensity, the mean and peak velocity, and the profile dispersion (i.e., the width). These scripts were written by Diane Gilmore (dgilmore@stsci.edu) and Ron Allen at STScI.

CHAPTER 6:

Instrument-Specific Software

In This Chapter...

Several packages in STSDAS support the calibration of HST instrument data sets. This chapter provides an overview of the HST instrument-specific tasks available in STSDAS and introduces some of the concepts behind them. This chapter is not intended as a complete reference to the instruments or the calibration process.

The **hst_calib** package contains all of the tasks related to HST instrument calibration. One package, called **ctools**, contains general calibration support tasks that are common for two or more instruments. The **synphot** package contains tasks for simulating HST observations or doing photometry and spectral analysis. The individual tasks for each instrument are in separate subpackages, named after each instrument. These packages are:

- **foc** - Calibration and analysis tasks for the Faint Object Camera.
- **fos** - Calibration and analysis tasks for the Faint Object Spectrograph.

- **hrs** - Calibration and analysis tasks for the Goddard High Resolution Spectrograph.
- **hsp** - Calibration and analysis tasks for the High Speed Photometer.
- **wfpc** - Calibration and analysis tasks for the Wide Field/Planetary Camera and the Wide Field/Planetary Camera-2.

Each of these packages, includes a task that is identical to the software used to calibrate data in the RSDP (Routine Science Data Processing) pipeline in PODPS (Post Observation Data Processing System, part of SOGS). The calibration tasks in STSDAS produce the same results as the RSDP pipeline when the same reference data and calibration parameters are used, because the code is identical. Calibration algorithms are developed and tested in STSDAS, and then installed in RSDP without modification. (The calibration tasks are listed in Table 6.1). This section is intended only to provide an overview of the calibration support software for each instrument. Extensive discussions of practical uses of the software are deferred to the *HST Data Handbook*.

The *HST Data Handbook* describes, in detail, the procedures used to interactively run the STSDAS calibration software. It also includes descriptions of how to find and retrieve reference files and background information describing how the instruments work and limitations of the data and their processing.

The Fine Guidance Sensors (FGS) are not supported by the STSDAS software. Information about FGS data and its processing can be found in Chapter 27 of the *HST Data Handbook* on page 419.

The Off-Line Calibration Process

Most observers find that the data sets shipped after the observation were calibrated satisfactorily, however, there are several reasons why you may wish to recalibrate the data set, among these reasons:

- A better reference file may have become available
- You may wish to omit a calibration step that was done
- You may wish to use a different version of the calibration code

STSDAS tasks in various packages can be used to create new reference tables and files, change keywords, and modify data, thus redefining the parameters used by the calibration software, and creating a data set that is calibrated the way you want.

The flow chart presented in Figure 6.1 shows the basic process used to calibrate a data set produced by any instrument on the Space Telescope.

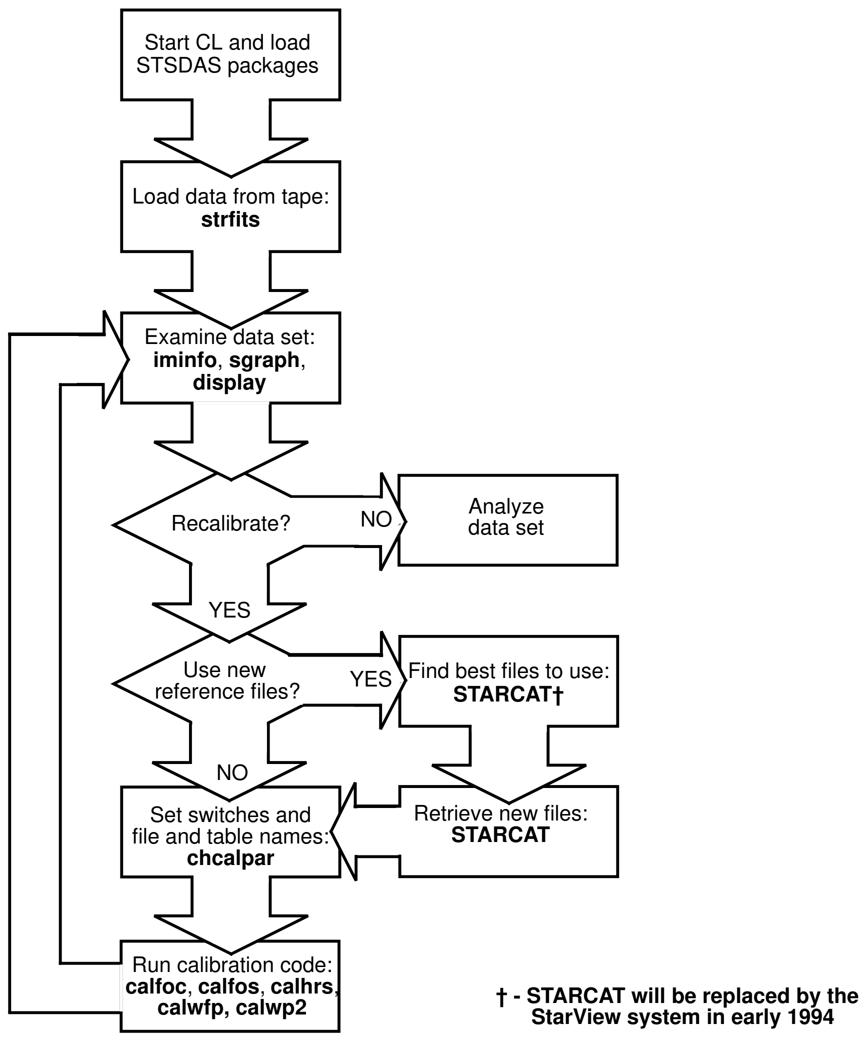

Figure 6.1: Basic Calibration Process for HST Data Sets

General Calibration Operations

Every HST observation creates a *raw* data set that is then processed through the RSDP pipeline process creating a *calibrated* data set. Every observer receives both of these data sets on their data tapes. Observers who are comfortable with STScI's standard calibration can immediately begin analyzing the calibrated data set.

Each STSDAS instrument package includes a calibration task (see Table 6.1). Each of these calibration tasks has a single input parameter—the root name of the raw data set to be calibrated—and a single output parameter, the root name of the output calibrated data set. All other parameters that affect the calibration, such as the names of reference files or tables, are specified in the header of the input raw data file. Various tools are available (such as **chcalpar** in the **ctools** package) to modify the header keywords. An overview of STSDAS tools that support the general calibration process can be viewed online by typing `help ctools opt=sys`.

Calibration Task	*Instrument*
calfoc	Faint Object Camera
calfos	Faint Object Spectrograph
calhrs	Goddard High Resolution Spectrograph
calhsp	High Speed Photometer
calwfp	Wide Field/Planetary Camera
calwp2	Wide Field/Planetary Camera-2

Table 6.1: STSDAS Instrument Calibration Tasks

Editing Headers with chcalpar

The **chcalpar** task is used to edit calibration header keywords for any HST dataset. The task works by loading keywords from your image into a pset for the instrument used, this pset is then edited with the **eparam** editor. When you are done editing keywords, you exit from **eparam** and are then asked to confirm your changes before the new values are written to your output image. As with all STSDAS tasks, details are found in the online help file (type `help chcalpar`). A detailed description of **chcalpar** also appeared in the Winter 1993 issue of the *STSDAS Newsletter* (Number 3, page 8), which is obtainable from the STSDAS HotSeat.

Reference Files

The calibration tasks use both *reference data files* (such as flat field images) and *reference tables* (with values such as a sensitivity constant that is applied to all data values). Copies of the reference files used to calibrate your observation are sometimes included with your data set if you received a tape from STScI. Copies of the reference files and tables can also be obtained from the HST Archive. To use the archive, you will need to request an account on one of the archive hosts and you can also request a copy of the *HST Archive Primer* to get you started. Requests can be made via e-mail to: `archive@stsci.edu`

Users at STScI will also find that the most current reference files and tables are available on-line. Reference tables are also available in the Calibration Data Base (CDBS) relations and can be accessed using the **cdbsutil** tools in the STSDAS **stlocal** package. Query tools in **cdbsutil** create STSDAS tables which can then be manipulated using tasks in the **ttools** package.

Calibration Data Base

Because the Calibration Data Base is stored on a dedicated data base server at STScI, the **cdbsutil** package and other CDBS-related tasks are not exported with the rest of the STSDAS software. Off-site users can use StarView to determine the best reference files to use in calibrating data. Within StarView are calibration reference screens for each instrument that produce listings of the references files used to calibrate a dataset and the current best reference files for these data. StarView can also be used to retrieve these reference files from the Archive. See “Determining the “Best” Reference Files” on page 70 in the *HST Data Handbook*.

The **cdbsutil** package provides tools to access the Calibration Data Base System (CDBS). CDBS contains a historical record of reference files used in the calibration of HST observation. In addition to tracking these calibration reference files, CDBS also tracks other files used by the synthetic photometry package. Tasks in this package can be divided into three broad groups. The first group reads information already contained in CDBS. The second group writes new information into CDBS (as part of the process of delivering new reference files). The third group of tasks does not access CDBS directly; these tasks either reformat data so that it can be submitted to CDBS or they reformat the output of other tasks so it can be used by other software.

The one task most likely to be useful to most users is **getreffile**, which retrieves the names of the best set of reference files for calibrating a specific observation. The reference files named in the image header are not always the best because newer and better reference files may have been delivered after the observation was taken.

The task **getsource** is rather specialized. It returns the number of reference files derived from each source for a specified data base table. The information produced by **getsource** is used to track the delivery of reference files for each instrument. There are three sources for CDBS data:

- In-flight data
- Ground data
- Dummy data

General tasks for searching the data base include **inquiry** and **query**. **inquiry** lets a user interactively select rows from a data base table using a form to describe the search criteria. **query** also searches the data base, but instead of using a form, an expression is used to specify the column names and data base tables that are to be searched. **query** requires that a user know the data base table names and their column names. Two tasks can be used to find this information. **shopar** returns the name of all data base tables associated with a specified instrument with a short description of their contents. **dbformat** returns the columns contained in a specified data base table and their types.

The process of delivering new files to PODPS begins with **stopar** which reads the contents of an STSDAS table into the CDBS data base. The **lodpar** task performs the inverse of this process, reading the most recent data inserted into a data base table back into an STSDAS table. The **verify** task sets the verification level of data in the CDBS data base. Data must be verified before being moved to PODPS. Reference tables must undergo two additional steps before being delivered to PODPS. **lodtab** creates an STSDAS table that has one row for each combination of data base columns that define an observation mode. It retrieves the most recent row from the database table for each observation mode. **stotab** stores the name of this table in the CDBS data base so it can be transferred to PODPS. The last step in the process of moving files to PODPS is **release**. This task creates a catalog of all files that are ready to be transferred to PODPS. This catalog is used to populate the PODPS data base. Some data base tables, whose names begin with the letters "cr", are used to track files used by the synthetic photometry software. These files are installed in the data base by **refdat**. The task **refcheck** can be used to check the format of these tables before installation.

Several tasks reformat information, either so it can be stored in the data base or used by other tasks. **rdschf** and **rdsiaf** take files from the project data base (PDB) and convert them to STSDAS tables that can be installed in the data base using **stopar**. **treformat** takes an STSDAS table whose column names are close, but not identical to the names of columns in a data base table, and renames them so they match the data base table. This is necessary for installation of the table using **stopar**. **mkretlist** takes the output produced by **getreffile** and writes a file that can be used to retrieve the reference files from the archive using StarView. **upreffile** updates the image header keywords containing the names of the calibration reference files using the output of **getreffile**.

Synthetic Photometry

Synthetic photometry is performed in the STSDAS **synphot** package. This package provides a uniform method of simulating and analyzing HST photometric and spectrophotometric observations. A detailed discussion of the techniques of synthetic photometry and the tasks available in the **synphot** package can be found in the *Synphot User's Guide* which is available from the STSDAS group.

The STSDAS **synphot** tasks allow you to:

- Plot HST sensitivity curves and calibration target spectra
- Predict count rates for observations in any available HST instrument mode
- Examine photometric transformation relationships among the various HST observing modes as well as conventional photometric systems, such as Johnson-Cousins *UBVRI*.

Table 6.2 lists the tasks available in the **synphot** package and a brief description of their functions. Several tasks simulate HST instrument passbands and spectra, calculate photometric observations from simulated or observed spectra, display and compare synthetic and observed data, and fit simulated data to observations. Synphot accesses the HST Calibration Data Base System (CDBS) reference files which contain sensitivity or throughput curves for all optical components in the HST and its instruments. The CDBS reference files also contain flux distributions of HST calibration targets which can be used in **synphot** to simulate observations of real targets.

The instrument mode sensitivity curves are generated at runtime by a passband generator by combining the throughput curves for the individual optical components that make up a given instrument observing mode. This is necessary because the number of possible instrument modes is so large that it would be impractical to independently calibrate and store the final sensitivity curves for every mode. To specify an instrument mode in the **synphot** tasks, you provide a list of component keywords such as `WFC,3,f555w`. The passband generator follows a path through the instrument *graph table*—i.e., the table that defines which optical components can be connected together. If the keywords define a unique path through the instrument graph, then a passband is built up as the product of the individual passbands of the components contained in the path.

The tasks **graflist**, **grafplot**, **grafcheck**, and **grafpath** are useful for getting information about a given instrument graph table. **graflist** makes a list of the components downstream from a given component and **grafplot** similarly provides a graphical representation of component paths. The **grafpath** task makes a list of all the component throughput tables encountered along the path of a given observation mode, while **grafcheck** is used to check an instrument graph table for bad rows.

Using the **synphot** tasks **calcband** or **plband**, you can calculate and display the sensitivity curve (passband) for any HST observing mode. These tasks also allow you to synthesize arbitrary passbands from analytic forms such as Gaussians, rectangular boxes, and series of Legendre polynomials, where you specify relevant parameters such as the central wavelengths, widths, and polynomial coefficients.

Similarly, the tasks **calcspec** and **plspec** are used to manipulate and display spectral data. The spectral data may be read from existing data files such as the HST calibration target spectral library or your own observations of targets of interest. Alternatively, spectra may be synthesized at runtime from a choice of available forms including black body, powerlaw, and HI emission spectra, where the user specifies desired parameters such as temperature, spectral slope, column density, etc. The spectrum calculator also allows you to renormalize spectra to any desired flux level, add or remove extinction effects, add and subtract spectra, and multiply by a chosen observation mode passband. Spectral data may also be calculated or converted to any of eleven types of units, including f_ν, f_λ, counts, observed magnitudes, Jy, and mjy.

The **calcphot** and **countrate** tasks can calculate photometric quantities for given combinations of spectra and observing modes. The **calcphot** task can be used to compute the total flux in a given passband, while the **countrate** task will in addition create an output spectrum containing the number of counts as a function of wavelength within the specified passband and is therefore especially well-suited for simulating observations for spectroscopy instruments, such as the Faint Object Spectrograph (FOS) and the Goddard High-Resolution Spectrograph (GHRS). The input parameters for the **countrate** task were specifically designed to mimic the list of entries that an observer will encounter on an HST exposure log sheet. On page 139 we provide an example of how the **countrate** task can be used with GHRS data.

The task **plratio** displays the ratio of observed to predicted photometry and spectrophotometry and produces statistics of the observations. The **pltrans** task displays photometric transformation plots, such as color-magnitude or color-color diagrams.

All **synphot** tasks are designed to read input data (for either passband throughput curves or spectral data) from tables. These input tables may be in either ASCII text format or in STSDAS table format. Output data generated by the **synphot** tasks are stored in STSDAS tables. The task **imspec** can be used to convert spectral data from IRAF or STSDAS images into STSDAS tables (or vice versa) for use in **synphot**.

There are several spectral atlases of observed and model data that are available in STSDAS table format for use with synphot tasks. These atlases are available from STEIS under the `cdbs` directory. Detailed descriptions and locations of these atlases can be found in Appendix B (page 93) of the *Synphot User's Guide*.

Task Name	*Function*
calcband	Calculate a model passband
calcphot	Calculate photometric quantities for spectra and passbands
calcspec	Calculate a model spectrum
countrate	Evaluate the photometric count rate of the HST
evalband	Compute the instrument passband for an observation mode
fitband	Fit a passband model to data using Amoeba simplex method
fitgrid	Fit a spectrum model to a grid of spectral data
fitspec	Fit a spectrum model to data using Amoeba simplex method
genwave	Interactively generate a wavelength set
grafcheck	Check an instrument graph table for bad rows
graflist	List the components downstream from a given component
grafpath	Print keywords and component names for a path through the graph table
grafplot	Plot the components downstream from a given component
imspec	Convert an image to a synphot table or vice versa
plband	Plot a set of passbands
plratio	Plot the ratio of observed to synthetic spectral
plspec	Plot spectral and photometric data
pltrans	Plot photometric transformation diagrams, such as color-magnitude or color-color

Table 6.2: Tasks in the synphot Package

Example of Using countrate

To give you a feel for how **synphot** tasks work, we provide below a simple example of using the **countrate** task. Many other examples of using **synphot** tasks are provided in Chapter 3 of the *Synphot User's Guide*. The **countrate** parameters are shown in Figure 6.2. Before we jump into the example, let's first look at the task first to get an idea of how **synphot** tasks and their parameters work.

```
output =              Output table name
instrument =          Science instrument
(detector = " ")      Detector used
(spec_elem = " ")     Spectral elements used
(aperture = " ")      Aperture / field of view
(cenwave = INDEF)     Central wavelength (HRS only)
(userspec = " ")      User supplied input spectrum
(synspec = " ")       Synthetic spectrum
(synmag = " ")        Magnitude of synthetic spectrum
(refwave = INDEF)     Reference wavelength
(reddening = 0.)      Interstellar reddening E(B-V)
(exptime = 1.)        Exposure time in seconds
(verbose = yes)       Print results to STDOUT ?
(count_tot = INDEF)   Estimated total counts (output)
(count_ref = INDEF)   Estimated counts at reference wavelength
(refdata = "")        Reference data
```

Figure 6.2: Countrate Task Parameters

About countrate

Unlike other **synphot** tasks in which the observing mode is specified via the single parameter `obsmode`, the **countrate** task uses separate `instrument`, `detector`, `spec_elem`, and `aperture` parameters to specify the observing mode. The `instrument` parameter specifies one of the HST instruments: FGS, FOC, FOS, HRS, HSP, WF/PC, or WF/PC-2. If you set the instrument parameter to HRS, you should also specify a value for `cenwave`, the central wavelength of the spectrum (see below). The `detector` parameter specifies the name of the desired detector, if there is more than one available for the chosen instrument. The `spec_elem` parameter specifies the name of spectral elements, such as filters or gratings. Finally, the `aperture` parameter specifies the name of the instrument aperture, if there is more than one available for the chosen instrument. The `instrument` parameter must always be specified, but one or more of the `detector`, `spec_elem`, and `aperture` parameters may be left blank, depending on their applicability to a given instrument. The task automatically chooses a wavelength set tailored to each HST instrument and observing mode.

The `cenwave` parameter specifies the central wavelength for the simulated observation, in Angstroms; the output spectrum will be centered at this wavelength. If set to “INDEF”, the output spectrum will contain the entire wavelength range covered by the chosen observation mode. This parameter only affects HRS instrument modes, because it is the only instrument where the detector cannot cover the entire wavelength range of an observation. This parameter is ignored for all non-HRS instrument modes.

The target spectrum may be specified in one of two ways:

- As a table name for the `userspec` parameter
- As a spectrum to be synthesized by the task via the `synspec` and `synmag` parameters

The synthetic spectrum will only be used if the `userspec` parameter is blank. The `crcalspec$` and `crgrid$` directories contain tables of HST flux calibration star spectra and various spectral atlases (both models and observational data) that can be used as source spectra. The `synspec` parameter recognizes the functional forms listed in Table 6.3.

Value	*Functional Form*
`bb temp`	A blackbody at temperature `temp`.
`hi temp colden`	Emission spectrum of an LTE slab of hydrogen at temperature `temp` and column density `colden`.
`pl refval exp`	A power law spectrum of the form $(\lambda \,/\, \mathrm{refval})^{\mathrm{exp}}$

Table 6.3: Functional Forms

The `synmag` parameter is used in conjunction with `synspec` to normalize the synthetic spectrum to a chosen absolute flux level. This is done by specifying the desired integrated broadband magnitude, in units of `stmag` , for the spectrum in one of the standard *UBVRI* passbands. The syntax used to specify `synmag` is a two-word string consisting of the desired magnitude value, followed by the name of the desired passband, e.g., "`synmag = 17.3 V`."

In addition to computing the total counts integrated over the chosen instrument passband, it is possible to compute the number of counts at a particular reference wavelength through the use of the `refwave` parameter. If the chosen instrument is a spectrograph (FOS or HRS), the task will compute the counts in the pixel containing the reference wavelength. If the chosen instrument is not a spectrograph, the task will compute the counts per Angstrom at the chosen reference wavelength.

The `reddening` parameter may be used to add or remove the effects of interstellar reddening on the input spectrum and is specified in units of $E(B-V)$. It provides the same functionality as the `ebmv` function in the spectrum interpreter of other **synphot** tasks. Either the user spectrum (`userspec`) or the synthetic spectrum (`synspec`) may be modified by this parameter.

The exptime parameter is used to specify the desired exposure time, in seconds, to be applied in the calculation of the total counts.

If the verbose parameter is set to “yes”, then the results of the total counts and counts at the reference wavelength calculations will be written to the terminal. In addition, these two quantities are also written to the two output parameters count_tot and count_ref.

Sample countrate Run

This example computes the result of a 100 second observation of the star Feige 110, using the HRS with the large science aperture (lsa) and the g160m grating. We’ll also have it compute the number of counts in one pixel at 1504 Å. The spectral data for Feige 110 are read from the table feige110_002.tab in the directory crcalspec$. The calculated spectrum, in units of counts per pixel, is written to the table hrsobs.tab.

```
sy> countrate hrsobs hrs spec=g160m ap=lsa cen=1495 exp=100 \
>>> refw=1504 userspec=crcalspec$feige110_002.tab
```

With verbose=yes, the output on your screen will be the following:

```
133605.84 # Total counts
   222.53 # Counts at reference pixel
```

Figure 6.3 shows the resulting spectrum as plotted using the STSDAS task **stplot.sgraph** (e.g., sgraph "hrsobs wavelength flux").

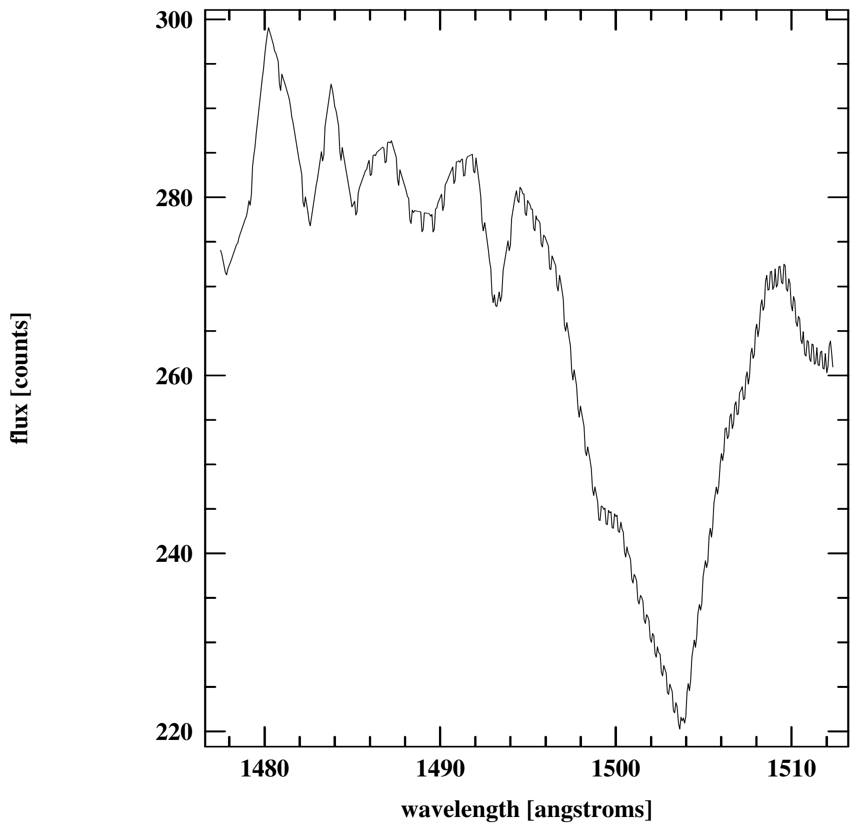

Figure 6.3: Plot of Countrate Example Spectrum

Faint Object Camera

The software for the Faint Object Camera is rather extensive, and is divided into five subpackages under the **foc** package:

- **focgeom** - FOC geometric corrections.
- **focphot** - Tasks having to do with FOC photometry.
- **focutility** - General FOC utilities, including **calfoc**.
- **focprism** - Tasks for FOC prism calibration.
- **foccs** - Calibration tasks.

The **focutility** package contains a number of tasks used to convert images to the particular reference data file format (background correction, geometric correction, detector efficiency correction, etc.) needed in the pipeline calibration. (Tasks that read and write data in the internal data formats used for the ground-system calibration of the FOC can be found in the **stlocal.testfoc** package. These tasks will rarely be needed, except by users who are developing new calibration reference files for the FOC.)

For historical reasons, most tasks in the FOC packages have the letter “x” at the end of the task name. In HST jargon, “x” is the instrument code letter for the FOC. Because of the package organization, however, this is not necessary, and task names may be simplified in a future version of STSDAS.

FOC Geometric Processing

Geometry tasks in the **focgeom** package will calibrate, model, and display FOC distortion, generate calibration files for the pipeline processor, and rectify images.

Geometric distortion occurs both in the optics (the Optical Telescope Assembly, COSTAR, and the FOC relay) and in the Photon Detector Assembly (PDA).

The FOC relay produces an image on the faceplate of the PDA. The amount of optical distortion introduced is about 10 or 20 pixels maximum. This was calculated by ray tracing through the optics.

The PDA distortion is larger—about 50 pixels towards the edges of the format. Because the PDA is an electron-optic device, the distortion field is complex, variable, and cannot be modelled to an accuracy of 0.1 pixels by simple functions in two dimensions. Instead, the distortion in a real image is measured by locating—in a fairly well-lit (i.e., a long enough exposure time to ensure statistical accuracy), uniformly illuminated image—-marks specially deposited on the faceplate of the PDA intensifier and lying in a regular grid (called a *reseau*). These marks are referred to as *reseau marks*. The marks on the photocathode form a regular 17 x 17 grid. Each reseau mark is approximately 3 pixels (75 microns) square.

Optical Distortion

The optical distortion has been modelled by a polynomial fit, and for each relay a file containing the fit may be found in the **focgeom** package directory.

The basic steps required to create a reference image for geometric correction are as follows.

1 • Find the reseau marks in a flat-field image.

2 • Create a reseau table containing a reference grid corresponding to the measured reseau positions.

3 • Evaluate the polynomial fit for optical distortion over the reference grid, thus creating a reseau table with the reference positions projected onto the sky.

4 • Fit a polynomial that maps the sky positions to the reseau positions measured in the image.

5 • Convert that polynomial fit into a geometric correction reference image.

Figure 6.4 shows the entire process, starting with a flat-field image (`x1j30104t.d0h`) and ending with a geometric correction reference image (`zlrg.hhh`). Parameters not listed take the default values. Further details are given later in this section. The discussion below is general, but this example includes specific numbers that are appropriate for the full-format COSTAR-corrected f/96 relay.

```
fo> dezoomx x1j30104t.d0h x1j30104t.hhh
fo> resfind x1j30104t.hhh respos.tab respos
fo> imdelete x1j30104t.hhh
fo> print ("26 986 -27 933", > "temp.lis")
fo> rgenx temp.tab "" rotref temp.lis 17 17 squa
fo> rtransformx temp.tab * rotref.tab rs -5 506 453 \
>>>  shift1=0 shift2=-10
fo> delete "temp.lis,temp.tab"
fo> revalfitx focgeom$opt151.poly optref rotref.tab \
>>>  rotref sky.tab
fo> rfitx respos.tab respos sky.tab optref zlrg.poly \
>>>  1 1090 -60 1000 xxorder=9 xyorder=9 xxterms=yes \
>>>  yxorder=9 yyorder=9 yxterms=yes
fo> p2geo zlrg.poly respos zlrg.hhh 1024 1024 pxformt="zoom" \
>>>  sampoff=0 lineoff=0 samppln=512 linepfm=1024
```

Figure 6.4: Creating a Reference Image for Geometric Correction

The **focgeom** package directory contains five files with names such as `opt151.poly`. The number (e.g., 151) is the focal ratio of the relay to which the file applies. These are IRAF *text database* files which contain the coefficients of two-dimensional polynomial (actually Chebyshev) fits. The fit describes the inverse optical distortion, that is, the mapping from full-format image pixel coordinates to the corresponding positions on the sky, also in pixels. The coordinates of points on the photocathode range from one to 1024 in each axis. For each of these files, the point (512,512) is mapped into approximately (512,512).

The positions of reseau marks in a flat field may be found using the **resfind** task in the **focgeom** package. This task combines an extrapolation technique with subroutines from **rfindx** to locate reseau marks. It is not necessary for all the reseau positions to be found. In fact, if some reseau marks are ill defined or obscured, it is usually better to leave their positions undefined.

Now we must create a reference grid of reseau positions. This grid is where the observed reseau marks would end up after geometric correction if we only corrected for PDA distortion and not optical distortion. A reference grid may be created with the **rgenx** task using the parameter setting `gridtype = "squa"` and a `coordfile` that contains one row with four numbers, the first and last X and Y positions of the reference reseau marks. The values should be chosen to give a spacing of 60 pixels between the individual reseau marks. The example above shows an appropriate choice for these numbers for a full-format image. This will create a grid parallel to the image axes, and you would then use

rtransformx to rotate the grid to approximately line up with the observed reseau grid.

The next step is to run **revalfitx** to evaluate the optical distortion polynomial fit at the positions in the rotated reference grid (identified by entry name `rotref`). The result is the set of positions on the sky (but in pixel coordinates) that would be imaged by the optical system onto the rotated reference grid. Note that the `.poly` files in **focgeom** were designed to be run with **revalfitx** parameter setting `direction = "forward"`. These positions projected onto the sky (entry name `optref`) together with the observed reseau positions (entry name `respos`) then fully describe the distortion, to the extent that we can know it based only on the reseau marks. The dimensions of the `optref` grid and the `respos` grid are the same, e.g., 17 columns by 17 rows. For each position in `optref` there is a corresponding position in `respos`, although some of the latter may be undefined (i.e., set to INDEF).

Next run **rfitx** to fit a polynomial between the reference positions projected onto the sky and the reseau positions measured in the image. The input reseau table `input1` is the set of positions obtained by **resfind**, while the reference reseau table `input2` is the rotated reference grid projected onto the sky.

Finally, run **p2geo** to convert the polynomial fit output from **rfitx** into a geometric correction reference image.

Spectrograph Mode

For spectrograph mode observations, the model describes the distorted appearance on the PDA faceplate of slit images of a line source. Ray tracing on a 5 x 5 grid performed by Matra/LAS showed the distortion to be linear to within about one pixel. It is assumed that the physical slit entrance aperture is perfectly straight. No account is taken of how the slit maps onto the sky. There is not yet any task that applies the spectrographic optical distortion to reseau positions.

A distortion model must be generated so that a spectrograph image can be rectified, oriented so that the slit images from a line source are parallel to `AXIS1` and at the same time the wavelength dispersion is uniform in the `AXIS2` direction.

Generating a spectrograph geometric reference file is a four step operation.

1 • Correct an image of an emission-line source for PDA distortion.

2 • Measure the angle of the long slit lines to `AXIS1` using the **slitangle** task. Rotate the square reference file by the negative of the previously-measured angle using the **rtransformx** task. Geometrically correcting the spectrograph image with this file will produce an image having spectral lines parallel to `AXIS1`, however, the dispersion may not be linear.

3 • Identify lines in the central section of the spectrum using the task **noao.twodspec.longslit.identify**. Extend the line identification along the slit using **longslit.reidentify**.

4 • Fit a polynomial function to the dispersion relation using **longslit.fitcoords**, and evaluate the fit over a uniform grid using **revalfitx**.

Producing FOC Photometric Files

For reasons of convenience, the calibration for normal imaging mode is done in two stages: relative DE and an absolute sensitivity. The latter does not change the data values themselves—it only computes a factor to be added to the image header.

Relative DE: The relative detector efficiency (DE) files (also known as the "UNI" files) are images normalized to unity that describes the variation in the detector response over the imaging area when illuminated uniformly with almost monochromatic light. In principle, the non-uniformity of the source should be corrected so that we have the response to a uniform illumination. We start with a flat-field image and perform the following steps:

1 • Subtract dark count. The **imarith** task can be used to multiply the dark count image by the exposure time and subtract it from the observed flat field.

2 • Run **flall**.

Absolute Sensitivity: During pipeline calibration, a factor is computed that may be used to convert from image counts to flux density as seen by the telescope. This factor is written into the header of the calibrated image using the keyword `PHOTFLAM`. The **synphot** package contains tasks (e.g., **evalband** task) for computing this factor based on the instrument configuration, and **calfoc** calls the **synphot** subroutine to obtain this factor. The factor has been checked using in-flight data. This was done by comparing the counts in actual stellar images with the counts predicted by `PHOTFLAM`.

Three factors contribute to the absolute sensitivity:

- HST optics throughput.
- Filter transmissions. These are assumed to be known from ground-based measurements on the filters.
- Detector (photocathode) sensitivity. This has been updated based on in-flight observations.

Wide Field/Planetary Cameras

The **wfpc** package contains the Routine Science Data Processing (RSDP) calibration code for WF/PC and WF/PC-2 observations, **calwfp** and **calwp2**, as well as tasks to generate parameters and perform other supporting functions. **calwfp** performs a standard sequence of processing steps on individual WF/PC camera observations, removing most of the known instrument signature; **calwp2** performs similar functions for observations taken with the WF/PC-2 instrument. These corrections include the A/D fixup, removing the global bias level, subtracting bias, preflash, dark, and superpurge frames (after appropriate scaling), and applying the flat field frame. The input data quality file (DQF) is updated with pixels observed to be saturated, and with the DQF of known bad pixels (static mask), and with the DQFs of any images used in the processing sequence. Histograms of the input data, the data following A/D correction, and the output data are generated. The steps performed by this task are specified by setting keywords in the input image header.

Several other tasks are available in the **wfpc** package to aid in the analysis of calibrated data. Most of these tasks are used for both WF/PC and WF/PC-2 data. Most of these tasks can exclude any or all classes of bad pixels (as identified in the DQF associated with each image) from the calculations, and all tasks either perform operations on all groups by default, or provide a means to switch between groups without exiting the task, as appropriate. These tasks include:

- **checkwfpc**: Check the correctness of the image header of a WF/PC reference file and its associated data quality mask.
- **combine**: Combine images using various algorithms and pixel rejection schemes. This task is similar to **images.imcombine**, but will ignore DQF-flagged pixels and loop through all image groups.

- **crrej**: Reduce the cosmic rays registered in images of multiple exposures of the same field and combine the images by rejecting very high counts in each pixel stack. This task has more sophisticated options for propagating cosmic ray events to neighboring pixels and doing iterative rejections with several threshold levels.
- **dqfpar**: A pset for specifying which DQF error flags to use in masking bad pixels. The user may exclude any or all pathologies when using the **combine**, **noisemodel**, and **wstatistics** tasks.
- **engextr**: Extract information from WF/PC engineering data (i.e., the `.x0h` and `.x0d`) files and write it to an STSDAS table.
- **metric**: Calculate the RA and Dec of points in a WF/PC image while correcting geometric distortions and offset and rotations between the chips.
- **noisemodel**: Determine noise model parameters from CCD frames. The gain, read noise, and scale noise are used in other tasks as a basis for modelling the noise characteristics of WF/PC data.
- **noisepar**: Set parameters describing noise model (pset).
- **pixcoord**: Compute pixel coordinates of stars in WF/PC frames based upon an input list of stellar RA and Dec, and world coordinate system (WCS) information in the image header. This task also fine-tunes the WCS information with an interactive fit of the mapped stellar positions.
- **qwmosaic**: Produce a quick (i.e., no geometric correction) mosaic of the four groups that comprise the full WF/PC field of view. A more sophisticated mosaic task is available (**wmosaic**) to correct for the geometric distortion and inter-chip rotations, offsets, and scale differences.
- **seam**: After the four groups of a WF/PC image are mosaicked (using the task **wmosaic**) into one image, brightness discontinuities may still show up near the boundaries between the four frames creating a *seam* effect. This task will essentially airbrush the seams from the image, making it look smoother.
- **wfixup**: This task will take an input image and its data quality file, check for pixels flagged as bad in the data quality file, and do a simple linear one-dimensional interpolation over the bad pixels. The result looks better when displayed, but may not be improved from a scientific perspective.

- **wmosaic**: Mosaic the four individual WFPC frames into one image. This task does geometric correction and corrects for rotation, offsets, and scale differences among the chips. The **qwmosaic** is quicker than **wmosaic**, but does no corrections.
- **wstatistics**: Compute and print WF/PC image pixel statistics. This task is similar to **images.imstatistics**, but it will loop through all image groups and optionally exclude DQF-flagged pixels from the calculation. This task is somewhat outdated and will be replaced by the **imgtools.gstatistics** task.

There are two subpackages in the **wfpc** package: **u_calib** and **w_calib**. These subpackages contain tasks that are used for only WF/PC data or for only WF/PC-2. The WF/PC-specific tasks are in **w_calib**, the WF/PC-2 specific tasks are in **u_calib**.

Faint Object Spectrograph

The **fos** package contains the Routine Science Data Processing (RSDP) calibration code (**calfos**), routines to generate the calibration reference files and tables required by **calfos**, routines to generate the parameters needed for observation planning and instrument configuration, and routines to display and analyze FOS data. The RSDP pipeline, operated at STScI, provides consistent calibration of FOS observations. Tasks related to spectropolarimetry can be found in the **specpolar** subpackage.

Calibration Reference Files and Tables

Table 6.4 summarizes the FOS calibration parameters needed by RSDP. For each parameter, the table lists the STSDAS tasks that generates the parameter, and the name of the CDBS relation (if any) in which the parameter is stored.

Parameter	*Used by Task*	*Data Base Relation*
Dead time constants	**deadtimey**	CYCCG2R
Aperture parameters	**aperlocy, apscale**	CYCCS0R
Aperture positions	**ybase**	CYCCS1R
Emission line regions		CYCCS2R
Sky and background filter widths		CYCCS3R
Wollaston prism and waveplate parameters		CYCCS4R
Sky shift parameters		CYCCS5R
Wavelength coefficients	**linefind, dispfity, waveoffset, fitoffsety**	CYCCS6R
GIMP correction scale factors		CYCCS7R
Predicted background count rates		CYCCS8R
Background reference files	**parthity**	CYBACR
Flat field reference files	**flatfield**	CYFLTR
Inverse sensitivity reference file	**abbsseny, absfity**	CYINVR
Retardation reference files		CYRETR
Disabled diodes files		CYDDTR
Quality initialization files	**parthity**	CYQINR

Table 6.4: FOS Calibration Parameters Required by RSDP

RSDP Calibration Code—calfos

The calibration process consists of the following steps:

1 • Compute statistical errors

2 • Convert to count rates

3 • Apply GIMP correction

4 • Correct for diode non-linearities

5 • Subtract background

6 • Subtract scattered light

7 • Apply flat fielding

8 • Subtract sky

9 • Assign wavelengths

10 • Convert to absolute flux units

11 • Special mode processing

The propagated error is calculated at each point in the spectrum. This is done by first calculating the statistical errors of the raw data (as the square root of the raw counts per pixel). The error spectrum is then processed in lock-step with the science spectrum throughout the remaining steps of the calibration process.

Raw counts are converted to count rates by dividing each data point by the exposure time and correcting for disabled diodes. If the science data header keyword `DEFDDTBL` is true, then the disabled diodes are taken from the Unique Data Log (UDL), otherwise, disabled diodes are found in the file specified in the keyword `DDTHFILE`. This file is selected from the reference relation `CYDDTR`.

Count rates are corrected for the geomagnetically-induced image motion problem (GIMP). GIMP-correction scale factors (table `CCS7`) are used to shift the spectrum in the FOS X-direction to correct for the effects of the Earth's magnetic field on the FOS detector. Offsets are based on model calculations of the Earth's magnetic field at the position of the telescope at the time of the observation. This correction is not necessary for observations that were obtained using the on-board GIMP correction scheme (which became available in April 1993).

Raw count rates are corrected for saturation in the detector electronics. Dead time constants (table `CCG2`) are used to correct for the nonlinear response of each diode. These constants are not expected to change through the lifetime of the instrument. Analysis of prelaunch data indicate that the same constants can be used for both FOS detectors.

The background can be subtracted from the sky and object spectra. If no background was taken, a default reference background, indicated by the keyword `BACHFILE`, is used. The observed background is smoothed with a median, followed by a mean filter, before subtraction. Filter widths are stored in table `CCS3`. No smoothing is done to the reference file background, if used. The reference background may, however, be scaled (using table CCS8) according to the geomagnetic position of the telescope at the time of the observation. The default background file is selected from the reference relation CYBACR.

Diode-to-diode sensitivity variations and fine structure are removed by multiplying by the flat field response. This process requires the flat field response file given by the keyword FL1HFILE. A second flat field file, FL2HFILE, is required for paired-aperture or spectropolarimetry observations. These files are selected from the reference relation CYFLTR.

The sky spectrum, if observed, can be subtracted from the object spectra. The emission line relation (CCS2) gives regions where a sky spectrum should not be smoothed before subtraction from the object spectrum. The filter parameters table (CCS3) contains the filter widths (mean and median) for smoothing the sky (or background) spectra. The sky shift parameter (CCS5) gives an integer shift—in pixels—that is to be applied to the sky spectrum before subtraction. The aperture parameters (CCS0) contain aperture areas required to scale a sky spectrum taken in one aperture of a paired aperture before subtraction from the object spectrum taken in the other aperture. Values for this relation are required only for the paired apertures. The aperture position parameters (CCS1) are used to determine which aperture (UPPER or LOWER) of an aperture pair, was used for observing an object or sky spectrum.

The wavelength parameters (CCS6) give the dispersion coefficients used to generate wavelength scales. Separate coefficients are required for each detector, disperser, aperture, and polarizer mode.

Object spectra are converted to absolute flux units by multiplying by the inverse sensitivity vector. This step requires the inverse sensitivity file found in header keyword IV1HFILE. A second inverse sensitivity file, IV2HFILE, is required for paired-aperture or spectropolarimetry observations. These files are selected from the reference relation CYIVSR.

The final step performs ground software mode dependent reductions for time-resolved, spectropolarimetry, and rapid-readout observations. The spectropolarimetry reductions require the Wollaston and Waveplate parameter table, CCS4, and the retardation reference file, RETHFILE. The retardation file is selected from the reference relation CYRETR.

The calibration process is controlled by the values of certain header keywords. These keywords may be examined and modified using the task **chcalpar**.

The later versions of **calfos** (version 1.2.3 and later) are not backwards compatible with all FOS header formats. In this case, the task **newfoshdrs** can be used to transform the old headers to an acceptable format for running **calfos**. Please note, that some additional header changes may be needed to get **calfos** to run to completion.

Observation Planning and Instrument Configuration

In addition to the calibration parameters needed by RSDP, there are other FOS calibration parameters that are needed to configure the instruments, and compute target acquisition parameters. These additional parameters are described in Table 6.5.

Parameter Description	***Used by Tasks***
Y-bases	**ybase**
Discriminator values	**phay**
Target acquisition	**aperlocy, apscale**

Table 6.5: FOS Calibration Parameters (Non-RSDP)

Y-bases are the Y positions required to center a spectrum on the diode array for any detector, disperser, aperture, and polarizer mode. These must be closely monitored as they affect the quality of the scientific data. Improper Y-bases will decrease the amount of flux recorded by the diode array.

The detector discriminator values are computed by the **phay** task, which fits a Gaussian function with a polynomial baseline to the differential pulse height distribution.

The **aperlocy** and **apscale** tasks are used to analyze FOS target acquisition images.

Data Reduction and Display

The tasks listed in Table 6.6 are used to display or prepare the data for further analysis.

STSDAS Task	*Description*
fwplot	Plot flux vs. wavelength
mkmultispec	Add wavelength solution information to an image header
rapidlook	Display a multigroup image as a two-dimensional image
vac2air	Convert vacuum wavelength to air
resample	Resample data to a linear wavelength scale

Table 6.6: FOS Display and Analysis Tasks

Goddard High Resolution Spectrograph

The **hrs** package contains the Routine Science Data Processing (RSDP) calibration code (i.e., the **calhrs** task), routines that generate the calibration reference files and tables used by **calhrs**, routines to generate parameters needed for observation planning and instrument configuration, and routines to display and reduce GHRS data. The RSDP pipeline calibrates all observations in a consistent manner. The STSDAS calibration software provides the calibration parameters used in the pipeline, and performs the actual calibration of all observations.

Calibration Reference Files and Tables

Table 6.7 describes the GHRS calibration parameters used in the pipeline. For each parameter, the table lists the STSDAS tasks that generate the parameter, and the name of the CDBS relation (if any) in which the data are stored.

Parameter Description	*Tasks Using Parameter*	*CDBS Relation*
Deadtime constants	**deadtimez**	`C0CCG2R`
Aperture parameters	**tamode**	`CZCCR0R`
Line mapping parameters	**lmap**	`CZCCR1R`
Sample mapping parameters	**smap**	`CZCCR2R`
Detector parameters		`CZCCR3R`
Wavelength ranges		`CZCCR4R`
Spectral orders	**yfind, yfit, carrousel**	`CZCCR5R`
Dispersion constants	**wavecal**	`CZCCR6R`
Thermal constants		`CZCCR7R`
Incidence angle	**woffsets, incidence**	`CZCCR8R`
Echelle interpolation constants	**centerflux, abssenz, ripple**	`CZCCR9R`
Echelle non-interpolation constants		`CZCCRAR`
Global wavelength coefficients		`CZCCRCB`
Photocathode blemish table		`CZCCRDR`
Scattered light coefficients		`CZCCRBR`
Diode response	**diodecal**	`CZDIORR`
Photocathode response	**cathcal**	`CZPHCRR`
Vignetting		`CZVIGRR`
Absolute sensitivity	**centerflux, mergecar**	`CZABSR, CZNETR`
Data quality	**parthity, quality**	`CZQINR`

Table 6.7: Pipeline Calibration Parameters for GHRS

The calhrs Calibration Code

The following steps comprise the HRS calibration process:

1 • Removal of diode non-linearities and non-uniformities

2 • Removal of photocathode granularities

3 • Remove vignetting and low frequency variations across photocathode

4 • Wavelength assignments

5 • Background subtraction

6 • Echelle ripple correction

7 • Conversion to absolute flux units

The calibration uses tables and files whose names are contained in keywords in the science data header. The keywords are similar (but not identical) to the CDBS relation names.

Diode non-linearities are removed by applying a paired-pulse or deadtime correction using coefficients stored in the table `CCG2`. The non-uniformities are then removed by dividing by the diode response for each diode (or an average response if comb-addition is used), stored in reference file `DIOHFILE`. `DIOHFILE` contains 512 responses (one per diode) and is selected by detector.

Photocathode granularities are removed by first mapping each data point from detector coordinates (diode and deflections) to photocathode coordinates (line and sample) using coefficients in tables `CCR1` and `CCR2`. The photocathode response is then obtained from the reference file `PHCHFILE`, which contains a response grid organized by line and sample. The reference file `PHCHFILE` is selected by grating mode.

Vignetting correction corrects light loss caused by both vignetting and low-frequency sensitivity variations across the photocathode. The vignetting file, `VIGHFILE`, is selected by grating mode, and has values tabulated as a function of carrousel position, line position, and sample position.

Wavelengths are determined for each point by solving the dispersion equation relating photocathode sample position to wavelength and spectral order. For the first-order gratings, the spectral order is defined as 1. For the Echelle spectra, the spectral order is computed from the Y-deflection and carrousel position, using parameters in table `CCR5`. The dispersion coefficients are found in table `CCR6` and are corrected for thermal effects using coefficients in table `CCR7`. If the object is in the large science aperture, an incidence angle offset is added to the wavelengths. The offset is computed using coefficients stored in table CCR8. A velocity correction is done to convert to heliocentric wavelengths and finally, a vacuum-to-air correction is performed for wavelengths above 2000 Å.

The background is smoothed by a median and a mean filter (with widths specified in table CCR3, or a low-order polynomial (with order specified in table CCR3). If the background is a sky spectrum (as specified by its substep bin ID), it is normalized to the object spectrum aperture using aperture sizes stored in table CCR0 before being subtracted from the entire spectrum. Contributions from scattered light are calculated using coefficients found in table CCRB and are subtracted with the background.

A ripple correction is performed for data taken with an Echelle mode using coefficients in tables CCR9 and CCRA.

The net flux is then converted to absolute flux units using the sensitivity tables stored in reference files ABSHFILE and NETHFILE.

The calibration process is controlled by the values of certain header keywords. These keywords may be examined and modified using the task **chcalpar**.

The later versions of **calhrs** (version 1.2.3 and later) are not backwards compatible with all GHRS header formats. In this case, the task **newhrshdrs** can be used to transform the old headers to an acceptable format for running **calhrs**. Please note, that some additional header changes may be needed to get **calhrs** to run to completion.

Observation Planning and Instrument Configuration

In addition to the calibration parameters needed by RSDP, other GHRS calibration parameters are needed to optimize the performance of the detectors and configure the instrument.

Parameter Description	***STSDAS Tasks***
Detector focus	**focusz**
Thresholds	**pha_summary**
Target acquisition	**tamode, tascale, spiralmap, showspiral**

Table 6.8: Other GHRS Calibration Parameters

The GHRS detector focus is measured by the **focusz** task as the distance (in deflection units) from the 10% to the 90% level of the photocathode edge profile.

The detector thresholds are computed by **phaz** by fitting a Gaussian function with a linear baseline to the differential pulse height distribution. The position of the Gaussian is used to set an optimal threshold for each diode.

The **tamode** and **tascale** tasks are used to analyze GHRS field maps and convert from detector to V2,V3 coordinates. **spiralmap** and **showspiral** create and display a mosaic of a spiral search target acquisition observation.

Data Reduction and Display

The tasks listed in Table 6.9 are used to display or prepare the data for further analysis.

Task Description	*STSDAS Tasks*
Plot flux vs. wavelength	**fwplot**
Display multigroup images as two-dimensional images	**rapidlook**
Convert vacuum wavelengths to air	**vac2air**
Resample data to a linear wavelength scale	**resample**
Apply new absolute sensitivity corrections to calibrated flux data	**reflux**
Combine individual spectra from an FP-SPLIT observation into a single spectrum	**specalign**
Determine pixel offsets between spectra by cross-correlation	**poffsets**
Convert wavelength solutions to header parameters	**mkmultispec**

Table 6.9: Data Reduction and Analysis Tasks

High Speed Photometer

Software for the High Speed Photometer is located in the **hsp** package, including the calibration task **calhsp**. The **hsp** package includes tasks that can be used to calibrate the High Speed Photometer, monitor the instrument's health, and reduce its data. The tasks are grouped below, based on their relation to the Routine Science Data Processing (RSDP), which is automatically performed at STScI on all science data received from the spacecraft. RSDP-related tasks include algorithms to generate the calibration coefficients for high voltage and gain factors, detector efficiency, dark signal, dead time, etc., as well as the calibration algorithm (**calhsp**), which applies all these correction factors to the observational data. Tasks not related to RSDP include those used to monitor the instrument's health (warm-up time determination, detector linearity verification, etc.), calibrate the polarimeter, locate the apertures, and other general tasks (spacecraft velocity and position determination, polynomial fitting and evaluation, etc.).

RSDP-Related Calibration Software

The Routine Science Data Processing is performed at STScI to calibrate all HSP observations in a consistent manner. The main purpose of the STSDAS calibration software is to provide the calibration parameters needed for RSDP; Table 6.10 summarizes these parameters. For each parameter, the table lists the primary STSDAS task that generates the parameter, and the name of the calibration data base table (if any) that is used to store the parameter. A complete definition of the calibration data base tables can be found in the *Calibration Data Base Data Design* document.[1]

1. The *Calibration Data Base Data Design* document is currently in preparation. Users who need a copy should contact the Science Programs Division at STScI.

Parameter	*Used by Task*	*Data Base Relation*
Aperture size		`CCP0`
High-voltage factor	**voltfac**	`CCP1`
Analog gain factor	**gainfac**	`CCP2`
Pre-amplifier noise	**twodpolyfit**	`CCP3`
Detector efficiency (relative sensitivity)	**relsen**	`CCP4`
Detector dark signal	**darkv**	`CCP5`
Current-to-voltage converter (CVC) offset	**twodpolyfit**	`CCP7`
Paired-pulse correction (dead time)	**deadtimev**	`CCP8`
Dark aperture translation table		`CCP9`

Table 6.10: HSP Calibration Parameters Required by RSDP

Aperture sizes (`CCP0`) and dark aperture translations (`CCP9`) are not quantities that are calibrated with observational data. The rest of the above quantities, except paired-pulse correction (`CCP8`), have the identical formula in RSDP processing. They are expressed as a third-order, bi-dimensional polynomial function of temperature and time:

$$
\begin{aligned}
X = X0\{ & \left[1.0 + a_{01}(t-t0) + a_{02}(t-t0)^2 + a_{03}(t-t0)^3\right] + \\
& \left[a_{10} + a_{11}(t-t0) + a_{12}(t-t0)^2 + a_{13}(t-t0)^3\right](T-T0) + \\
& \left[a_{20} + a_{21}(t-t0) + a_{22}(t-t0)^2 + a_{23}(t-t0)^3\right](T-T0)^2 + \\
& \left[a_{30} + a_{31}(t-t0) + a_{32}(t-t0)^2 + a_{33}(t-t0)^3\right](T-T0)^3\}
\end{aligned}
$$

Where:

X is a calibration quantity,

X0 is the base value of *X*,

t is the observation epoch,

t0 is the reference epoch,

T is the temperature, and

T0 is the reference temperature.

Thus, sixteen coefficients are needed for each set-up combination (for example, in the case of analog gain factor, each detector ID and gain setting combination has 16 polynomial coefficients) in each relation. Paired-pulse correction is calibrated only as a linear function of temperature for each set-up combination (detector ID, voltage, and threshold setting).

Non-RSDP-Related Calibration Software

In addition to those calibration parameters needed by RSDP (described above), several HSP calibration parameters are needed to make full use of the instrument's other functions, and to monitor its performance. These additional parameters are described in Table 6.11.

Parameter	*Used by Task*	*Data Base Relation*
Aperture location (phase I)	**apercen**	CVAPER1
Aperture plate scale	**imgscale**	CVAPER2
Large aperture center coordinate	**lgaperloc**	CVAP3L
Small aperture center coordinate	**smaperloc**	CVAP3S
Optimum discriminator threshold setting (pulse height analysis)	**phav**	CVDISCR
Polarization transmission coefficients	**poleffv**	CVPOLEF
Polarization position angle offset	**polpav**	CVPOLPA
Polarization verification	**polverify**	CVPOLVF

Table 6.11: Additional Calibration Parameters (Non-RSDP) for HSP

Non-RSDP tasks in this package can be put into four groups. The first group performs polarimetry calibrations, the second locates the aperture center, the third monitors instrument health, and the fourth performs miscellaneous functions.

- Polarimetry Calibrations:
  - **poleffv**: Determines the polarization transmission coefficients of each of the Polaroid analyzers in the polarization detector, i.e., the amount of observed polarization of an ideal, 100% linearly-polarized target. This task uses data obtained by observing a target with different telescope roll angles. By comparing the result polarization

with the target's *known* polarization value, the polarization transmission coefficients are acquired.

- **polpav**: Determines the polarization position angle offset from the standard equatorial coordinate system. This task's input data are obtained by observing a target with all four Polaroid analyzers. By comparing the resulting position angle with the target's known position angle, the position angle offsets can be calculated.
- **polverify**: Verifies the entire polarization data calibration procedure. A target is observed with one Polaroid analyzer at different roll angles and then the two previously-described corrections are performed. The result should agree with the known values.

• Aperture Location Calibration:

- **apercen**: The first step ("Phase I") of aperture location calibration is to determine the aperture's center in the coordinate system used for the image dissector tube (IDT) deflections. This is done by observing an extended target that fills the aperture and scanning the aperture. The center coordinate is then obtained by fitting a circle to the edge of this two-dimensional image.
- **imgscale**: The second step ("Phase II") of aperture location calibration, is to establish the transformation coefficients between the IDT deflection coordinate system and the V2,V3 coordinate system. This is done by moving the pointing of the V1 axis to a matrix of sky positions to locate a point source target. The deflection coordinate of the target is obtained by doing an area scan at each pointing.
- **lgaperloc** and **smaperloc**: ("Phase III") The third stage of aperture location is to obtain the aperture's center coordinate to an accuracy of 0.02 arcseconds in the V2,V3 coordinate system. For small apertures (task **smaperloc**), this is done by moving the telescope pointing to a matrix of sky positions to locate a point source target. A circle is then fitted to the edge beyond which no flux can be obtained from the target. For large apertures the scheme is slightly different: an open cross pattern is used in place of a square matrix for pointings, and the center coordinate is similarly calculated from the edge coordinates.

• Instrument Health Monitoring:

- **abssenv**: Compares the digital count rates and known fluxes of standard targets. Such comparison verifies the linearity of the detectors and the correctness of the entire calibration procedure. It also ties the instrument count rates to the absolute unit of flux or magnitude.

- **aloglin**: Verifies the linearity of analog data by comparing the analog measurements and known fluxes of standard targets. This task works like **abssenv**.
- **flex**: Determines the HSP optical bench flexure caused by temperature differences within HSP. This is also known as the "banana mode correction." A one-dimensional polynomial is used to fit the deflection coordinate deviation as a function of temperature differences.
- **focusv**: Determines the best HSP electronic focus setting by observing the same target with different focus settings. The setting having the highest count rate presumably has the least beam spreading, and therefore, is the optimum setting.
- **phav**: Determines the optimum threshold setting for the pulse amplifier and discriminator (PAD). A two-component (Gaussian and exponential) model is assumed to represent the noise generated from the detectors, and a least-squares fitting is used to obtain the model coefficients.
- **scatterv**: Determines the amount of scattered light from a bright source near, but outside, an aperture's view. This task produces a table of count rates from scattered light as a function of angular distances from the aperture center.
- **taflat**: Calibrates the flat field of HSP target acquisition apertures. A presumably uniform extended target is used for this calibration.
- **warmup**: Determines the time a detector takes to stabilize after its high voltage power supply is turned on. This quantity can be used in planning the instrument's schedule.

• General Tools:

- **monitor**: This task reduces the data obtained when HSP is operated as a space particle monitor. The result will primarily be used to delineate the South Atlantic Anomaly (SAA).
- **parthitv**: Removes particle events (data of abnormally high counts compared to their neighboring pixels) and calculates the mean count rate of a data file.
- **polyeval**: Evaluates polynomials representing the quantities needed by RSDP. This is a general tool used in many scripts to do partial calibrations.

- **polyfit**: Performs one-dimensional polynomial fitting. The task can fit any order of polynomial (up to nine), using a set of orthogonal polynomials.
- **posvel**: Calculates the position and velocity of the spacecraft, based on a fitted model from the flight ephemeris.
- **twodpolyfit**: Performs two-dimensional polynomial fitting. This will generate the final coefficients of the RSDP relations described above. This is a general tool that can fit any order of polynomials (up to 9 x 9), and it uses the singular value decomposition (SVD) algorithm.

CHAPTER 7:

Programming

In This Chapter...

There will always be things users want to do with their data that we have not anticipated, or that we have not had time to implement. Programming facilities are provided in IRAF and STSDAS to allow users to design and build their own software. These facilities range from the simple use of the IRAF CL to write data processing procedures, to Fortran language programming using one of two different Fortran interfaces, to programming in the IRAF Subset Pre-processor (SPP) language. The complexity of the problem you are faced with will govern the choice of a programming environment. If you happen to write a generally useful set of routines that might be useful to wide community, and would be willing to share them through the **contrib** package, please send e-mail to the HotSeat.

CL as a Programming Language

The IRAF CL contains a number of control constructs that enable you to build data processing scripts directly in the CL, allowing you to automate your processing or to build up your own programs. There are two ways to do this:

- Write your own scripts

- Use the **mkscript** task to automatically create a script

Writing Scripts

Even at the script-writing level there is a large variation in the complexity of operations that can be done. The use of wildcards in input file names is probably the simplest kind of procedure (`doit input*`), but if you look at the STSDAS software, you will find that some STSDAS tasks are actually CL procedures which call other more basic tasks elsewhere in the system.

The introduction to the CL in this manual can get you started in using the CL to construct data processing procedures, but you should refer to the documents *A User's Introduction to the IRAF Command Language* and *An Introductory Guide to IRAF Scripts* for help. Both documents are part of the standard IRAF documentation set. The STSDAS group can provide a *CL Programmer's Quick Reference Card* on request.

Using mkscript

The **mkscript** task in the **system** package creates a CL script so that you can run several commands together. While it's relatively easy to build simple scripts manually, **mkscript** does much of the tedious work to make sure that all of the parameters are present in the task calls. This becomes particularly useful for longer scripts involving tasks with many parameters.

Note that **mkscript** does not include any of the programming capabilities available in the CL such as loops or conditional execution. Nor does it produce a script task with parameters. However, any script produced by **mkscript** can be customized and enhanced by editing and adding to the script file. (See "An Introductory User's Guide to IRAF Scripts" by Ed Anderson for a much more complete description.)

In operation, **mkscript** prompts you for a task to add to the script. When you type the task name, **mkscript** runs **eparam** so that you can specify the task parameters. After exiting from the parameter editor, **mkscript** lets you add more tasks. At each point, **mkscript** prompts you to continue. When no more tasks are added, **mkscript** lets you submit the script as a background job.

The sequence of steps are:

1 • See if the script already exists. If the script exists and `append = no` the script file is deleted. (When `verify = yes` you will be prompted to confirm the deletion.)

2 • Provide a task name. If a task is not specified on the command line, you will be asked to specify one. The task must be in a loaded package. If the task can be loaded a message will be displayed.

3 • Set the task parameters using **eparam**.

4 • Confirm whether you want to add the task to the script. If `verify = yes` the command is displayed and you will be asked if the command is correct.

5 • Add additional commands. You will be asked if another command is to be added to the script. If you say “`yes`”, steps 2 through 5 will be repeated.

6 • Save the script or start over. If `verify = yes` the script is displayed, page by page, and you will be asked if the script is correct. If it is not, the script will be deleted and steps 2 through 6 will be repeated.

7 • If the `submit` parameter is not specified on the command line, you will be asked if the script should be submitted as a background job.

The parameter `hidden` is important because hidden parameters are not explicitly included in the script commands, therefore the values of the hidden parameters will be taken from the parameter file at run time. Thus, changes in the hidden parameters with **eparam** or explicit assignments may produce unexpected results if you run the task again later. On the other hand, if the hidden parameters are never changed, then the commands are more readable when the hidden parameters are not included.

---

mkscript is implemented with a script which makes it seem somewhat slow. The most important limitation is that the command formatter is based on the output of **lparam**. If a task parameter name exceeds 12 characters it is truncated by **lparam** and is then also truncated by the command formatter. The script will then fail when executed! Also, the task name may not be abbreviated.

Psets and mkscript

Note that **mkscript** does not automatically expand pset parameter tasks. If you wish to use pset parameters different from the current default values, you may include psets as task calls to be added to the script. You will be able to modify the parameters with **eparam** as for any task. However, when a pset task is executed by name, it invokes **eparam** for modifying the parameters. This means that any script containing a pset call will invoke the **eparam** parameter editor.

Alternately, you can edit the script created by **mkscript** and include the pset parameters in the argument list for the parent task, or set each parameter via an explicit assignment. For example, the following is the result of including the **pltpar** pset task and the **sgraph** task from the **stplot** package in a script using **mkscript**. (This would be the appropriate order since **sgraph** depends on values **pltpar**.)

```
pltpar (stack=no,
axis=1, pointmode=yes, marker=”box”, szmarker=0.005,
erraxis=0, errtype=”bartck”, pattern=”solid”,
crvstyle=”straight”, rejectlog=yes, barpat=”hollow”,
crvcolor=4, color=3, cycolor=no, version=”20July1993”)

sgraph (“dev$pix[*,100]”,
errcolumn=”E”, dvpar=””, pltpar=””, axispar=””,
version=”20Jul93”)
```

If you don’t want to have **eparam** executed when you run the script, you could edit the script and include some or all of the **pltpar** parameters in the **sgraph** calling sequence:

```
sgraph (“dev$pix[*,100]”,
errcolumn=”E”, dvpar=””, pltpar=””,axispar=””, stack=no,
axis=1, pointmode=yes, marker=”box”, szmarker=0.005,
erraxis=0, errtype=”bartck”, pattern=”solid”,
crvstyle=”straight”, rejectlog=yes, barpat=”hollow”,
crvcolor=4, color=3, cycolor=no)
```

mkscript Task Parameters

- `script` - The output script file name. Commands will be successively added to this file.
- `task` - The task name of the command to be added to the script. If given on the command line, then only commands for this task may be added to the script. If not given on the command line, then **mkscript** will query for a task name for each new command. The task name must *not* be abbreviated.

- `submit` - Submit the completed script as a background job as the last act of the task? If not given on the command line **mkscript** will query before submitting the script.
- `append` - Append new commands to an existing script file? If no the file will be deleted first. If `verify = yes`, the user will be asked to confirm the deletion. The default is "`no`".
- `hidden` - Include hidden parameters in each command? The default is "`yes`".
- `verify` - Verify each command, any file deletions, and the final script? The default is "`yes`".
- `logfile` - The script log file name. When **mkscript** submits the script as a background job, any command and error output is directed to this file. The default is `script.log`.

Example

A common use of **mkscript** in data analysis is to create repeated commands with different input data or parameters. The following is an example of creating a script with **mkscript** to do this sort of operation:.

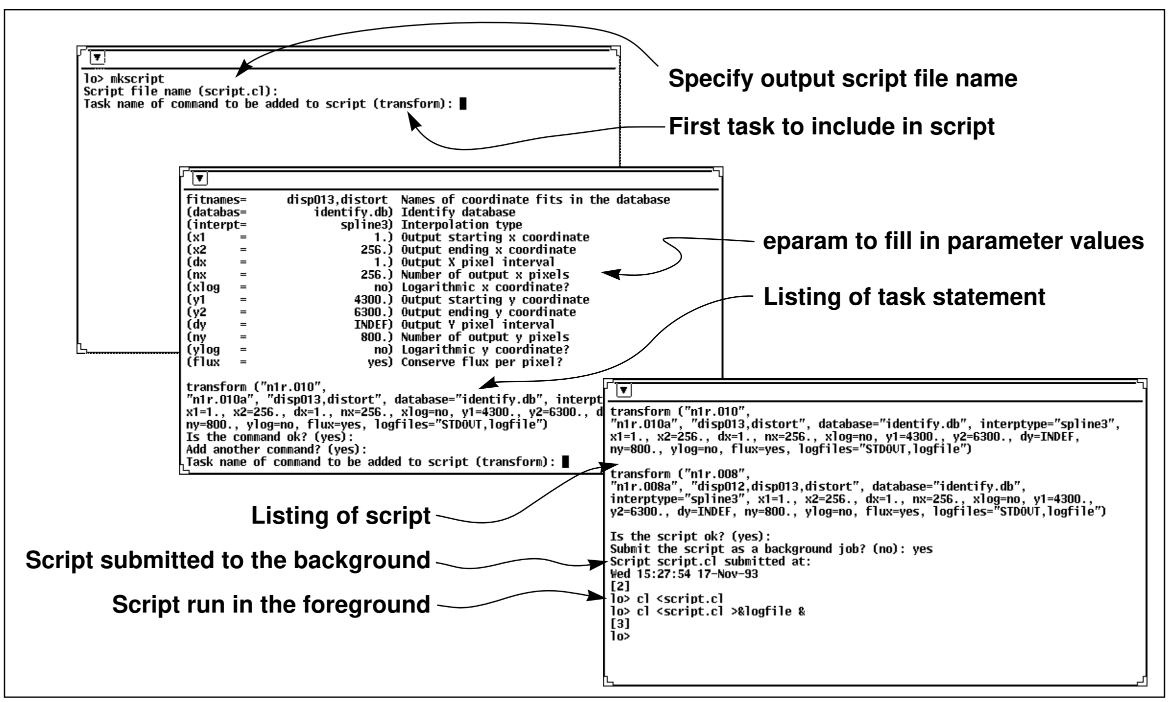

Figure 7.1: Creating a Script with mkscript

The IMFORT Interface

IRAF contains a Fortran interface called *IMFORT* which allows users to write their own Fortran programs that read and write data in the IRAF (OIF) image format. The subroutines provided in IMFORT can be called from any Fortran program, and you can run your program independently from IRAF. There is a limited capability for integrating an IMFORT program with IRAF so that the program can be used in conjunction with other IRAF or STSDAS packages without leaving the IRAF CL.

IMFORT supports image I/O to IRAF (OIF) format files *only*. You cannot read or write GEIS format images or STSDAS tables using this interface.

The F77VOS Interface

The STSDAS group developed a higher-level Fortran interface for IRAF called *F77VOS* (Fortran 77/IRAF Virtual Operating System). The F77VOS interface has a set of image subroutines similar to those in IMFORT, but they can access data in both the IRAF OIF and STSDAS GEIS formats. Also, F77VOS provides subroutines libraries that allow Fortran programs to be fully integrated with the IRAF environment. You can read and write parameters in the IRAF CL, send text messages to the user, do standard graphics, and read and write data in STSDAS tables. Because F77VOS programs are fully layered on the IRAF VOS, they must be run from the IRAF CL.

The F77VOS interface is described in the document *A Fortran 77 Interface for IRAF*, which is available from the STSDAS group at STScI.

SPP

Most IRAF and STSDAS applications are written in *SPP*, or *Subset PreProcessor*, which is the native programming language of the IRAF system. SPP is based on RATFOR—rational Fortran—and combines the familiarity of Fortran with some features of the C programming language. Users who wish to write applications that access the full range of IRAF capabilities should really learn and use SPP rather than the IMFORT or F77VOS interfaces. The greatest advantage of SPP is its portability. Applications written in SPP that use the data I/O and other interfaces correctly, can most likely be compiled and installed, without modification, on any machine and operating system for which an IRAF port exists.

The *SPP Programmer's Reference*, which is available from the STSDAS group, is a reasonably complete guide to writing and implementing SPP applications. This guide covers the basics of the language syntax, the system libraries and interfaces, and error handling. The guide also provides example procedures, instructions on compiling and linking programs as IRAF tasks, and instructions for debugging and maintaining tasks once they are written.

CHAPTER 8:

Troubleshooting

In This Chapter...

The STSDAS Group at STScI supports users who encounter problems installing or using the software, however, many of the problems that are reported as bugs or problems turn out to be simple oversights, or minor problems in the local installation or configuration. If you run into problems that are not solved in this guide or are not explained in our other documentation, contact us for help.

Basic Problems

Figure 8.1 lists a few ideas for starting to solve problems in IRAF or STSDAS.

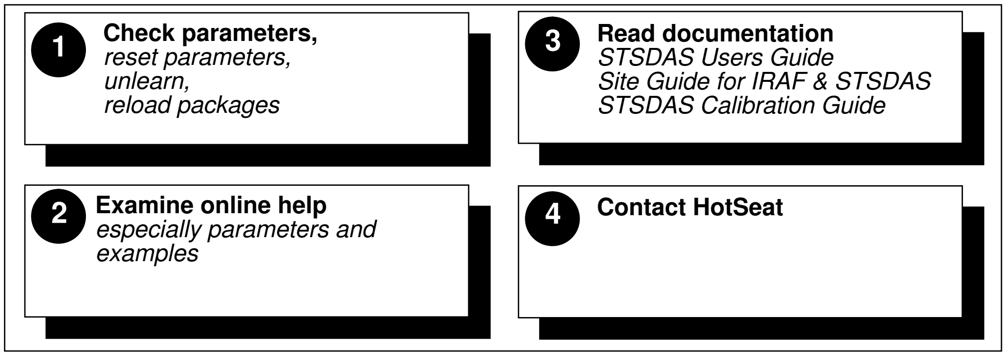

Figure 8.1: Troubleshooting Process

If you run into any problems using STSDAS, you can isolate and solve the problem by:

1 • Checking parameters.

Most problems are caused by parameter conflicts. Typical symptoms of this problem are:

- Unexpected prompts
- "Parameter not found" error messages

How to resolve parameter conflicts:

- Verify that parameter values are correct and reasonable.
- Clear unexpected values using **unlearn** on both the task *and* package. Also **unlearn** *previously loaded* packages.
- Make sure you start IRAF from within your *IRAF home* directory, otherwise your defaults will not be set correctly.
- Exit and reload the package.
- Exit IRAF and restart the CL. (Try this only if the previous steps failed.)

2 • Reading the online help.

Look at:

- Parameters (What values are expected? Are you giving the task appropriate values?)
- Examples

3 • Reading the documentation.

Yes, we know that some people don't want to read, but *most* questions answered by the HotSeat *are* in the documentation. You should have:

- *STSDAS Users Guide*
- *STSDAS Calibration Guide*[1]
- *Site Guide for IRAF and STSDAS*
- *Synphot Users Guide*
- *IGI Reference Manual*
- Quick Reference Cards
- IRAF documentation

If you don't have these documents, request them from the HotSeat.

Site managers should also have:

- *STSDAS Site Managers Installation Guide and Reference*

4 • Finding out if there's a task that does what you want.

To find out what task to use to do a particular job:

- Use the **apropos** command
- Look at the package organization chart (Figure 2.7 on page 25)
- Contact the STSDAS HotSeat

5 • Contacting HotSeat.

If your problem persists, and you suspect that there is an error in the software, contact the STSDAS HotSeat staff by sending e-mail to: `hotseat@stsci.edu`, or phone (410) 516-5100.

Tape Drives

Almost half the problems coming into the STSDAS HotSeat staff concern tape drives. To use the tape drive, you need to:

1 • Find out what drives are available on your host, where they are physically located, and what IRAF aliases they use.

2 • Allocate the drive.

3 • Mount your tape.

4 • Read and write files.

If you have problems using the tape drives, Figure 8.2 will help pinpoint common sources of tape errors.

1. This will be replaced in early 1994 by the *HST Data Handbook*.

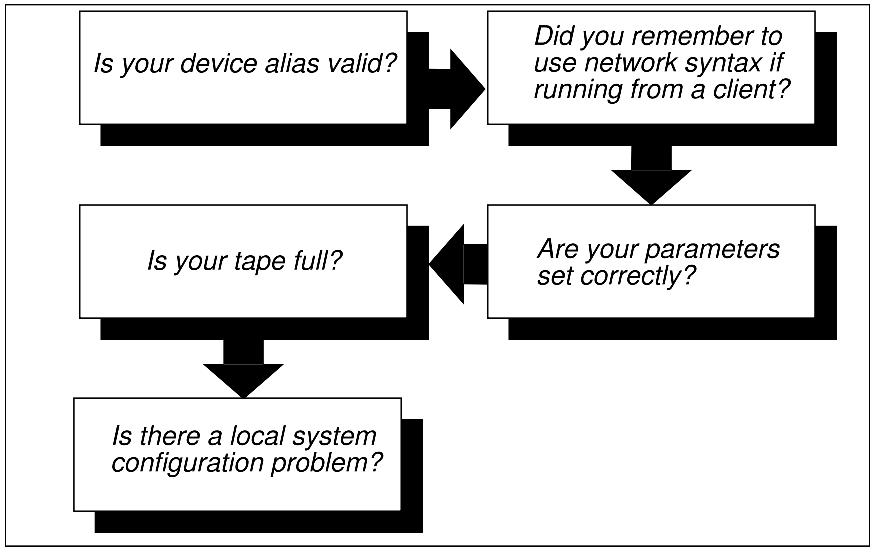

Figure 8.2: Common Tape I/O Problems

In the following sections, we will discuss some common tape problems, such as:

- How to find out what's on a tape.
- How to find a device alias using the dev$tapecap file (page 177).
- Using networking syntax with tape drives mounted on a server (page 177).
- Incorrect parameter settings (page 178).
- Full tapes (page 179).
- System configuration problems (page 179).

What's On Your Tape

Perhaps you're not having trouble reading the tape but don't know what the file names mean. There are three ways to find out what's on the tape:

- Look at the description of files that was sent with your tape.
- Use the **catfits** task in the **fitsio** package.

- If you're dealing with HST data, you can read the instrument *Data Products Guides*,[2] which describes the purpose of various files and the types of calibration reference files and tables you need to recalibrate the data set.

Device Aliases—tapecap File

You can always check to see if a particular tape drive on a particular host is supported by scanning through the file `dev$tapecap` file (use the **page** command in IRAF or your favorite editor). Figure 8.3 shows what a `tapecap` file looks like.

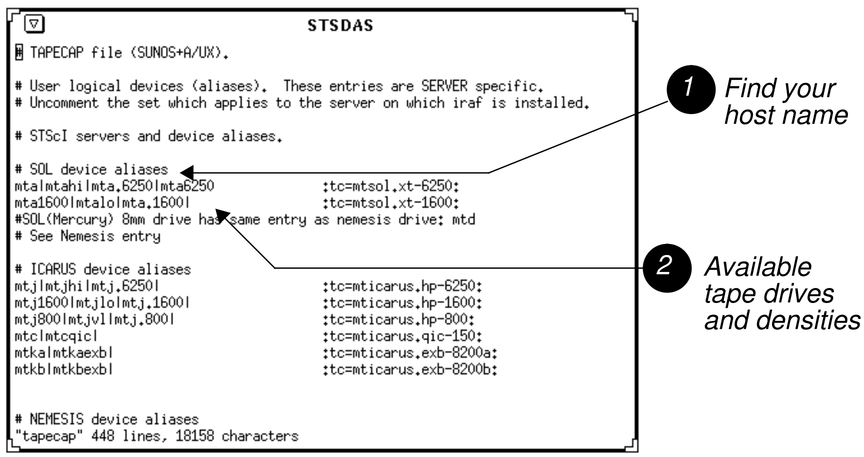

Figure 8.3: Sample dev$tapecap File

If one particular drive won't work—or one type of drive—and the `tapecap` file is correct, it is possible that an incompatible executable was somehow left lingering. Contact your local STSDAS manager.

Clients and Hosts—IRAF Networking

Quite a few users seem to occasionally run into problems when they are logged in to a client of a host while trying to use the host's tape drive. There are usually three problems that come up in this situation:

2. The *Data Product Guides* will be replaced by the *HST Data Handbook* in early 1994.

- Forgot to use network syntax
- Multiple users of `iraftape` account conflicting
- Network problems

Keep in mind that when you are using a tape drive mounted on a server but you're logged in to a client, you need to use *networking syntax*. For example, if you are logged in to the `neptune` client on the `sol` cluster and want to use `sol`'s tape drive `mta`, you can't just type commands like `allocate mta`, you would need to type `allocate sol!mta`. This is probably the problem if you get an error message like:

```
cannot rewind mta
```

If two people try using the `iraftape` account at the same time, one of them will get incorrect information from **devstat** because IRAF will think both users are the same person.

Occasionally, you will run into problems using the networking capability to read and write tapes on remote hosts. If so, try logging on to the remote machine and then using the tape drive. For example, if you are logged in to a `nemesis` client and try accessing a tape drive on the `ra` cluster, you should log in to `ra` before allocating the drive, otherwise you might see an error like:

```
segmentation violation
```

Pinpointing the source of a problem is not always easy. Sometimes what seems to be a tape I/O error is actually a networking problem. For example, look at the following error message:

```
Warning: Cannot close file (nemesis!nrmt8)
```

This happened during a tape read operation, but the fault is not with the tape reader or the drive, but rather with a faulty network connection. The clue to this is the exclamation point (bang symbol) indicating that this was a network operation.

Parameters

Another common source of problems is the *parameters*. Here are some common errors:

- `force_minmax`: When reading and writing FITS files, a floating operand error is a sign that the `force_minmax` parameter is wrong—try setting it to `yes`.

Also check the keywords in the offending file—particularly `DATAMIN` and `DATAMAX`.

- `xdimtogf`: If your file is empty or truncated, you probably set `xdimtogf` incorrectly. Read the parameter description in the help file and try it again.

If you use **wtar** on the VAX, the size of the file is not known ahead of time and will be padded with blanks giving you a no end-of-file error. You can read the file using the `-n` option with **rtar**.

Full Tapes

Tapes fill up when you put too much data on them. When this happens, you might get a message like:

```
kernel server died
```

System Problems

Not every problem you run into while reading tapes is going to be your fault. Some of the more common system-related tape I/O errors are:

- Tape drives moved or redefined.

  Sometimes the system managers move or redefine tape drives without letting IRAF know; other times they may modify or move files without checking to be sure the file is in the right place and has the right file protection privileges.

- Privileges set wrong.

  VAX users who get an error message like the following should suspect that their system managers have incorrectly set the `BYPASS` and `SYSPRV` privileges:

  ```
  %MOUNT-F-NOPRIV
  ERROR: cannot allocate device mtg
  ```

- Buffer too small.

  IRAF has an initially fixed-size buffer to copy FITS header parameters to the output image. This problem can occur in either **dataio.rfits** or in **fitsio.strfits**; it is identified by the message:

  ```
  card images discarded
  ```

To retrieve the missing parameters, you need to make the user area larger and rerun the FITS reader. This can be done with the following command:

```
fi> reset min_lenuserarea=60000
```

- FITS writer runs out of memory.

  On some systems, it is possible that you will run out of memory when running **stwfits** multiple times. This will give you the error message:

  ```
  out of memory
  ```

  If this happens, type `flpr` to flush the process cache and then rerun the task. It is also possible that a new device does not have a correct entry in the `tapecap` file or that you'll need some modifications to the FITS reader and writer. Contact the HotSeat in this case.

- Other system I/O errors.

  Occasionally, very odd I/O problems will show up that are not attributable to any specific user or system error. For example, in IRAF V2.9 there was a bug that would cause records to be skipped when using the FITS reader or would abnormally terminate the program. In this case, a patch was made available to fix the problem. (Contact the HotSeat for information about recent patches or bug fixes.) The following is a sample of the error messages that a user who encountered this bug might see:

  ```
  Panic in 20201450_C_2 Memory has been corrupted
  ERROR: Abnormal termination of child process STRFITS()
  Subprocess died
  % SYSTEM-F-ABORT
  ```

CD-ROM

In earlier versions of STSDAS, it was necessary to use the **cdrom** task to read files from a CD-ROM. CD-ROMs now work like any other disk in the file system. To access FITS files on a CD-ROM, use a task like **strfits**.

IRAF Networking

Very often, when IRAF is unable to establish a connection to a remote machine, the message is obscure or incorrect. For example, if the `.irafhosts` file is missing or is incorrectly set up, IRAF networking may generate the message "`no files found`". It is generally a good idea to first use the **dir** command to be sure that IRAF can see the files in question, before trying to access them over the network.

When moving files between the Suns and the VAXes, you need to make sure you have the following lines in your VAX `LOGIN.COM` file:

This directory is site-dependent—see your local administrator

```
if f$mode() .eqs. "OTHER" then exit
$ OTHER:
$ @DISK$STSDAS:[IRAF.VMS.HLIB]IRAFUSER.COM
$ EXIT
```

There are a few points to remember about moving files across the network:

- Tasks like **sun2vax** can move the files as they convert the binary format between the systems.
- When moving STSDAS images using **ftp**, you must set the transfer mode to `binary` or else problems will occur such as headers getting concatenated and parts of the image getting truncated.
- If you want to view files from remote clients, or view files stored on the VAX using your Sun workstation's screen, see page 9 of the *STScI Site Guide for IRAF and STSDAS*.

Graphics Output: Images, Displays, Printing and Plotting

Image display is fairly straightforward in the IRAF environment—you use imtool or SAOimage. Still, some problems come up with users not understanding where files are going or in what format they are being written. Similarly, plotting is straightforward but users still become confused with how graphics are mapped to the hardware or when plots get printed. Figure 8.4 shows some common problems when working with images and plots on various printers and displays.

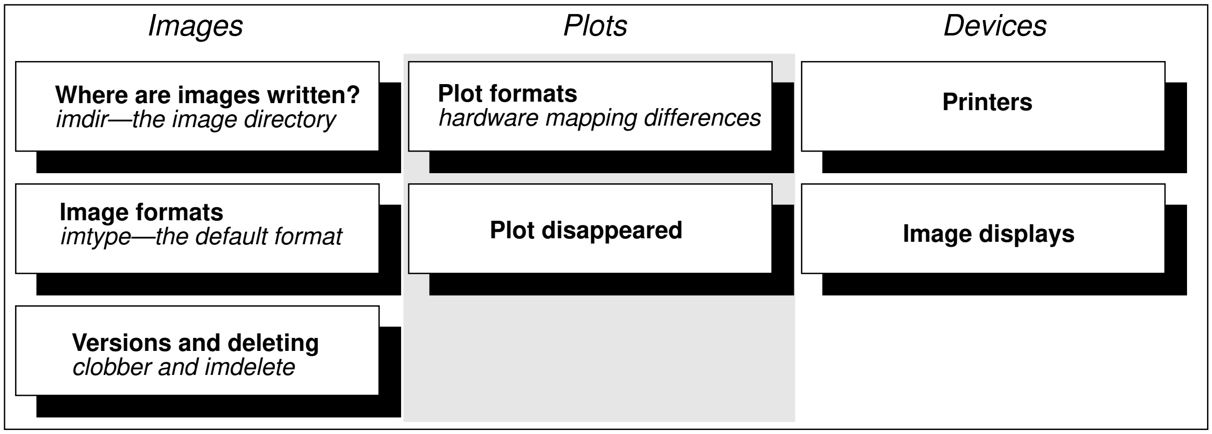

Figure 8.4: Imaging and Plotting Problem Areas

In this section, we will describe:

- Image problems
- Plot problems (page 184)
- Device problems (page 186)

Images

There are two environment variables that control how images are stored:

- `imdir` - Location of IRAF pixel files
- `imtype` - Default image format

imdir—the Image Directory

Images—both IRAF (OIF) and STSDAS (GEIS) format—comprise two disk files, an image header and a pixel file. In the case of IRAF images, the pixel files do not necessarily appear in the same directory as the header. However, you can override the default and specify where to put the pixel files. There are a few options and each has its advantages and disadvantages.

When you ran **mkiraf** to initialize IRAF, the `login.cl` that was created had a statement that looked like the following:

```
set imdir = "/usr/stsci/temp/user/"
```

The directory path specified by the value of the `imdir` variable is where the pixel files will be written whenever you create a new OIF image.

You can change this to be any directory you wish. A special value you can use for `imdir` is

```
set imdir = "HDR$"
```

The `HDR$` tells IRAF to put the pixel files in the same directory as the header files, regardless of where those headers are located. An extension to this is an additional path after the `HDR$` that would specify a subdirectory to that of the header files. For example, if you use **imcopy** to create a new copy of an image:

```
im> imcopy dev$pix home$images/mypix
```

And you set `imdir` as:

```
set imdir = "HDR$pixels/"
```

Your header file (`mypix.imh`) would be in `home$images` and the pixel file (`mypix.pix`) would be in `home$image/pixels`.

The disadvantage of specifying an absolute directory path in `imdir` has to do with IRAF networking. In order to locate the pixel file, there is a pointer in the image header to the pixel file. If you specify an absolute path for `imdir`, as is the default, then IRAF prepends the host name onto the pointer to the pixel file in the image file header, for example:

```
nemesis!/nem/data1/user/iraf/heq_1024.pix
```

The disadvantage of this is that if you then log onto a different host, even if it uses the same file system (say a Sun cluster sharing a file system through a server), then each time you access the image, IRAF will prompt you for your password. This is because it sees the host name followed by the "!" in the pointer to the pixel file, which indicates that it must use networking to access the pixel file.

imtype—Image Formats

New data files, such as those created by the **rfits** task, will be written in the format specified by the environment variable `imtype` (unless overridden). Any task that reads an input image and produces an output image will write the output image in the same format as the input image unless the output file extension is specified, for example:

```
cl> imcopy input.hhh output
```

would produce files `output.hhh` and `output.hhd` in the STSDAS (GEIS) format, and

```
cl> imcopy input.hhh output.imh
```

would convert the STSDAS format file input to IRAF format. The value of `imtype` can be defined using the **reset** command:

```
cl> reset imtype=imh     # IRAF format
cl> reset imtype=hhh     # STSDAS format
```

When image files are created or updated, they use the file `dev$pix` as a template. If you ever encounter a problem where an image header created on one system is different from that created on another system, compare the two `dev$pix` files. They should be identical, but have (on rare occasions) been known to get corrupted. Image formats are discussed in detail in Chapter 3.

Plots

When you print a plot, remember:

- The plot is not actually produced by the printer until you issue a **gflush** command.
- If the plot comes out wrong, you may need to use **unlearn**.

  For example, incorrect axis labels are a sign that you've got parameter conflicts. Remember to try unlearning other packages and tasks that may be loaded.
- If you don't know how to draw a particular type of plot, try using the **apropos** command to find an appropriate task.

  - Square plots are drawn by getting the aspect ratio from the `graphcap` file and using the `vx` and `vy` parameters in **pltpar**.

  - X,Y point plots are done using the `:/type` command from the graphics cursor.

  - Histograms? See the `crvstyle` parameter in **sgraph**, also look for the **images.imhist** and **stplot.histogram** tasks.

  If you still can't figure out how to do a plot, contact the HotSeat.

Disappearing Plots

When plots disappear, it is usually a sign that you need to clear the CL's graphics buffer. This is done using the **gflush** command after writing a hardcopy plot.

If **gflush** doesn't produce your plot, and the printer is working, then the batch job that handles output may have failed. On a VAX, you can check for a file called `sgiqueue.log` which contains a log of the job. This file is located in your login root directory (`SYS$LOGIN`), not the IRAF home directory. There are two reasons a plot job might fail:

- Too many pages submitted at once. Break the job into manageable pieces of no more than 12 pages.

- Your `LOGIN.COM` file failed. Most likely, you had it change your process name. If you reset the process name, you must be careful to only do so for interactive logins. For example, you should use the following commands:

```
$ if f$environment ("INTERACTIVE") .eqs. "FALSE"
then exit
$ set process/name="<witty name>"
```

Printed Plots Differ From Screen

There are two reasons that your printed plot may be different from that on the screen:

- Terminals and printers are handled differently
- IRAF graphics buffer is too small

IRAF hides device-dependent details, particularly when handling graphics. However, devices are inherently different and a terminal screen has a different size and shape than a page from a printer. Therefore, plots do not necessarily appear identical on the screen and on the printed page. This is most noticeable if you compare printed plots made in two ways. First, you might run a plotting task to display a graph on a terminal and use `=gcur` and `:.snap` to print the plot. Later, you might run the same task and set `device=stdplot` to send the graph directly to the printer. You will notice that these two plots look different, particularly if the task draws features with some particular scale or aspect ratio. For example, a circle will be circular on the screen but not on the printed plot. In the second case, circles should be circular on the printed plot.

Lines will not appear in a printed plot exactly as they appear on your screen because different terminals support different types of lines and IRAF graphics use whatever line types are supported for a particular type of terminal. These line types are defined in the file `dev$graphcap`. Lines drawn on a printed plot are drawn explicitly, they are not predefined. If there is a problem with the line types on a particular terminal type, contact the HotSeat.

The other reason for plots appearing differently is that the IRAF graphics buffer is too small. This buffer is a piece of memory maintained by the CL; it contains a copy of the instructions for drawing the plot. The CL uses the graphics buffer to interact with the plot, for example, in cursor mode to zoom, scroll, make hardcopies, etc. If you have a plot with many lines, (such as a contour map), then the graphics buffer can get filled. This will not affect the appearance of the original plot on the screen, However, if you replot on the screen or make a hardcopy, part of the plot will be missing. You can enlarge the buffer by setting the IRAF environment variable `cmbuflen` either on the command line or in your `login.cl` file. The default value is 64000, so for a very large plot, you may wish to increase this, for example:

```
cl> set cmbuflen = 512000
```

Your printer may not support plots of this size (512000).

Other Plot Problems

There are a few problems that come up occasionally in graphics tasks. These include:

- **wcslab** crashes with segmentation violation when overlaying a coordinate grid.

  The task is reading the image information from the display and not finding an image. You need to load the image using SAOimage or imtool before using **wcslab**. It's also possible that you read your files using the IRAF **rfits** task instead of the STSDAS **strfits** task.

- The `:skip` command doesn't work in **gkimosaic**.

  Some of the **gkimosaic** commands are different from generic `=gcur` commands, for example, the `:skip` command can only be used in interactive mode. You also need to use the space key to plot the frame specified by the `skip` parameter.

- Infrequent bus error when doing a plot.

  Type `flprcache` before running the script again.

Devices

In this section we describe problems related:

- Particular displays
- SAOimage on some workstations
- Printers

Displays

If you're using DecWindows and interacting between text, graphics, and images, you need to understand where the task is expecting to receive its input. This is called *input focus*. You set the input focus by clicking in the appropriate window—preferably in the menu bar to prevent the click from being interpreted as a keyboard interrupt. The input focus concept is especially important when using **apphot** tasks.

Below are a few additional questions that have come up related to image displays:

- Image rotation. If you use **images.rotate** and set `ncols=64` and `nlines=96` to rotate a 64 x 96 image, the image gets truncated to 64 x 64.

  Set `ncols=nlines=0` to let **rotate** calculate the correct dimensions.

- How can you set imtool to an 800 x 800 display?

  Use the `fitframe` option. If you want to see the whole image, you need to use a window at least as large as the image, however, the largest Imtool window you can have is 1144 x 880. If you load an image that's larger than the display buffer (up to 4096 x 4096), you can scroll a smaller window. The `fitframe` option forces the window size to match the display buffer size (up to the full screen size).

SAOimage Problems

IRAF can communicate with only one execution of SAOimage on a given workstation at any one time. It is possible to run multiple copies but IRAF will hang or produce unpredictable results if there's more than one.

Note that there may even be copies of SAOimage for which there is no visible display. You can determine how many copies are running on a given host by using the Unix command

```
ps -aux | grep saoimage
```

If there is only one copy, the output should look something like the following (the second line echoes the "grep" command used with ps):

```
levay 3772 0.0 0.0 1444 0 p1 IW 16:07 0:00 saoimage
levay 3778 0.0 0.7 32 220 p1 S  16:08 0:00 grep saoimage
```

If there is more than one copy, you can use the Unix kill command with the process number listed in output from the ps command:

```
kill -9 3778
```

Printers

If you have problems when printing:

- Make sure your environment variables are set correctly
- Check the printer to see if it is online and operating

- Check the `dev$graphcap` file
- If everything seems to be correct and you still can't print, contact the HotSeat.

If you're trying to make a hardcopy of the image on your display and you think you sent the image to the printer but nothing came out, you might try checking the environment variables in your `login` file. It could be that you set variables such as `SHINKO` to send the file to a Shinko printer. If you run into problems with SAOimage when displaying images from hosts to clients or across the network, check to make sure that you're only running one version of SAOimage on one machine.

If you get an error printing a file to a certain printer, but not to other printers, there could be a problem with a missing driver or font file. This might give you an error message like:

```
ERROR: cannot close file
/nem/p1/stsdasx/bin.f68881/x_cdroms.e: Command not found
```

If you get these errors, notify your local system administrator.

Files and Directories

This section describes problems with image files being stored in the wrong directories, missing half the file, or being written in the wrong format. Basically, STSDAS works with three types of files: *images*, *tables*, and *ASCII list files*. These formats are described in the Data Formats chapter on page 37.

Calibration Files

Calibration routines in STSDAS rely on a series of calibration reference files and tables. These are stored in the directory named by the variable `crrefer`. Other variables used to represent reference file directory names, such as `zref` and `mtab`, are subdirectories of `crrefer`. They are defined when STSDAS is loaded. You may need to check to see which directories are being pointed to by your variables (use the **show** command to see the current value of a variable; use **set** to reset it).

Incorrect `crrefer` values are indicated by error messages like the following:

```
ERROR Reference TABLE not found for keyword CCRB
```

Once you check to see which directories your environment variables point to you might want to check to see that the directory exists and that it's not located on a scratch disk and that you have read permission on the disk and directory.

File Format Conversion

Images can be written to IRAF format (`.imh`) rather than the default STSDAS format (`.hhh`) by using the **strfits** task and setting the parameter `iraf_file` to something with an `.imh` extension, for example, `dummy.imh`. If you don't know how to create an image using an ASCII list, look at the help for the **dataio.rtextimage** task. If you want to convert an ASCII file with flux vs. wavelength to an image, look at the tasks **tcreate**, **trebin**, **tpring**, and **rtext**. Remember that the **convert** task in daophot will only convert a dao text data base file to a table.

Tables

Table display tasks in the **ttools** package (e.g., **tread**, **tedit**, and **tprint**) can not handle tables where a single column is wider than your screen size. This should not generally happen, but some tasks (notably **calcphot**) create columns as wide as 159 characters. If your table has overly wide columns, you will see an error message when you try to display or edit the table. This error message might be:

```
Screen size error: t=1 b=30 l=4 r=3
ERROR: Abnormal program termination
```

Some of the table problems referred to the HotSeat are from users who simply don't understand how to use the tables. Read the online documentation and use the **?** command to list the brief descriptions of the tasks.

- **tchcol** changes the display format of the table.
- Select information from a table using the syntax `row()<...`

Other File Questions

Most users won't have trouble with files, but here's a couple questions that have come up:

- How do you create a multigroup file from image tasks?
  - Specify the total number of groups when writing the first group, and only the specific group number for each additional group, for example, `[1/5]` to create room for five groups; subsequent write operations would have groups specified as simply `[2]`, etc., i.e., without the slash. Group syntax is described on page 5-13 and more examples can be found there.
- How do you select header files based on a keyword?
  - Use **hselect**. An appropriate example is shown in the help file.
- How do you read image headers and group parameter blocks?
  - Use **images.imhead**

Installation and Site Management

This section describes both problems that system administrators might have installing IRAF and STSDAS as well as those problems that a user might discover that point to an installation problem.

A comprehensive discussion of the installation process is provided in the *STSDAS Site Manager's Installation Guide and Reference*.

Local IRAF system administrators should have:

- *STSDAS Site Managers Quick Reference Card*
- *STSDAS Site Manager's Installation Guide and Reference*
- *Site Guide for IRAF and STSDAS*

If you don't have these documents, request them by sending e-mail to `hotseat@stsci.edu`

Installation problems should be reported to the STSDAS administrator via e-mail to: `sdas@stsci.edu`

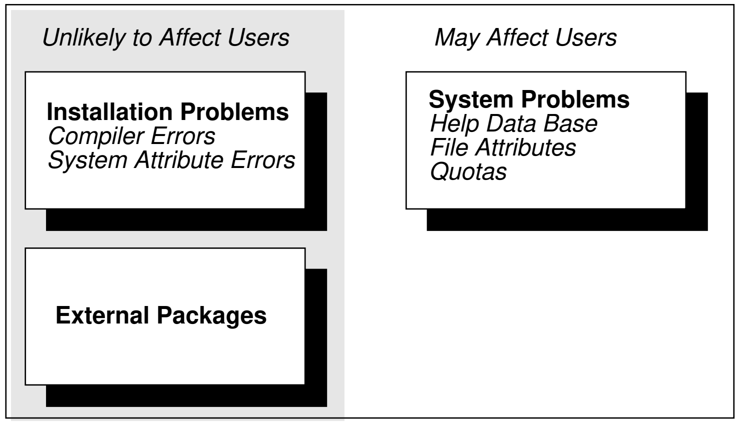

Figure 8.5: Types of Problems for Site Managers

In this section, we will describe:

- Common installation errors
- External packages (page 192)
- System problems (page 193)
- Setting up the help data base (page 193)
- File attributes (page 194)
- Quotas in VMS (page 195)

Installation Problems

System administrators will occasionally run into compiler errors or error messages produced by `mkpkg`.

- Can't compile at all.

  Check the version of xc. Older versions may not be able to execute f77.

- Can't link on DecStation.

  If you're having problems with the packages not linking on a DecStation because of unresolved references, then you need to modify xc to include an explicit library search path because the libraries are in a non-standard location.

- Compilation Errors.

  If mkpkg cannot find files or directories, you will get an error message such as:

  ```
  Warning: mkpkg line 23:
  dependency file '<tbset.h>' not found
  ```

  --or--

  ```
  ld: -llibraryname: No such file or directory
  ```

  To solve these problems, include the package name in the mkpkg statement, for example:

  ```
  mkpkg -p stsdas update
  ```

  If the Fortran compiler is not where IRAF expects to find it, you will get the following message:

  ```
  ld -lF77: No such file or directory
  ```

  To solve the problem, contact the IRAF group at NOAO to obtain a fix for xc. Send e-mail to: iraf@noao.edu

- Other Installation Errors

  If you have trouble installing and compiling STSDAS and can't solve the problem yourself, contact the STSDAS administrator, send e-mail to: sdas@stsci.edu

External Packages

Sites can install IRAF packages that are user-defined or that are obtained from other sites outside the normal IRAF distribution. Doing this often requires setting up definitions or file attributes, such as setting file ownership to iraf—this needs to be done by the IRAF administrator. If you need help doing this, contact the STSDAS administrator by sending e-mail to: sdas@stsci.edu

If you try installing a package with the wrong file attributes, you may get strange error messages, such as:

```
PANIC in path/file memory has been corrupted
PANIC in path/file Salloc
```

System Problems

Some errors can have multiple causes. Users will occasionally run into a message such as:

```
ERROR: Cannot open connected subprocess(pkg$x_pkg.e)
```

This message indicates one of five errors:

- STSDAS (or package) never compiled.
- **Executable not installed in** `stsdasbin`

  Verify that a file `xx_package.exe` exists in the directory. If so, move it to the correct directory with the command:
  ```
  mkpkg -p stsdas install
  ```
- Compilation aborted.

  Check the output log.
- Needs soft link to STSDAS directory.

  Some architectures, such as DecStations, require a soft link in the top STSDAS directory, pointing to the correct `bin` directory (for example, `bin -> bin.sparc` for a Sun Sparc). If the link is correct, do the following:
  ```
  % rm bin
  % ln -s bin.sparc bin
  ```
- File Attribute Errors.

  See the “File Attribute” section on the next page.

The Help Data Base

Occasionally, it may be necessary to rebuild the help data base, for example when new help documentation is added to the system. You will need to do this if you see an error message like the following:

```
Error: Help can't find stsdas or pipeline tasks
Cannot open help database file stsdas$hlib/helpdb.mip
```

This *does not always* mean you need rebuild the help data base. Often you simply need to type `unlearn help`. This is the case if the message above appears in conjunction with:

```
ERROR: parameter 'curpack' not found
```

If you do need to rebuild the help data base, you can do so as shown below:

```
cl> softools
so> mkhelpdb helpdir=stsdas$lib/root.hd \
>>>  helpdb=stsdas$hlib/helpdb.mip
so> mkhelpdb helpdir=pipeline$lib/root.hd \
>>>  helpdb=pipeline$lib/helpdb.mip
```

File Attributes

System administrators need to make sure that file protections are set correctly on all files used by IRAF and STSDAS. For example, if users do not have execute permission for the task executables, they will see a message like:

```
ERROR: Cannot open connected subprocess
  (stsdasbin$x_ttools.e)
```

System managers also have to make sure that read-write-execute permissions are set correctly on disks and directories used by IRAF. For example, some tasks write temporary files to the disk and directory pointed to by the `tmp$` variable. If that disk or directory has its permissions set wrong, you might see an error message like the following:

```
ERROR on line 51: cannot open ‘tmp$celco4944c.com’ for writ-
ing
celco ()
celco ()
```

or

```
ERROR (741, "Cannot open file (tempdisk:[iraftmp]sgk9554a")
```

Permission problems can sometimes manifest themselves in strange ways, for example, IRAF networking can fail when system-wide default login files have incorrect permissions. Users who encounter problems with incorrect permissions on temporary or scratch disks can work around the problem by setting up a `temp` subdirectory in their own home directory and redefining their environment variables, for example: `reset tmp=home$temp/`

If your tasks cannot open files, it is also possible that the files have been inadvertently moved or deleted. Try moving to the directory and listing the files to see if they really exist. If you find that you’re getting `par` files left in strange directories where you didn’t expect them to be, then you’re probably not starting IRAF from the correct directory. Always start the CL from within your IRAF home directory. (Some of the extraneous files you might get will be things like `uparmimsimhism.par` or perhaps `uparmsymlprint.par`).

Quotas

VMS users can sometimes use up their subprocess or page file quotas. If this happens, you will see an error message like:

```
Subprocess died: %SYSTEM-F-EXQUOTA, exceeded quota
Warning: User area too small 58 card images discarded
```

If this happens, you need to see your system manager and get your quotas increased. VMS IRAF users should have quotas of at least user class 3.

If your user area is too small, and your quotas are correct (or if you're running under Unix, not VMS) then you may just need to reset the size of your user area. The default users are is 20,000 bytes and each FITS header record is 160 bytes. You can reset the user area size with the following command (which will allow room for 375 records):

```
fi> reset min_lenuserarea=60000
```

System administrators installing STSDAS may have problems if their `/tmp` disk is too small. You can change the location of the default temporary directory by editing the file `hlib$mkpkg.inc` to add compiler and linker flags, or you can create a soft link to a directory with more space.

APPENDIX A:
CL Crib Sheet

In This Appendix...

Environment Variables

There are three commands for working with IRAF environment variables:

- **set** - Define the value of a variable. Typing set without an argument lists the values of defined variables.
- **show** - Displays the values of all environment variables.
- **reset** - Sets the default value of a variable.

When you **set** an environment variable, the new value is written to a stack, so it is only in effect for packages loaded after setting the value. Using the **reset** command instead of **set** will cause the original value to be overwritten so the new value will be in effect even if you exit from a package (using the **bye** command)—this will only work for overwriting pre-existing variables, new variables will be discarded.

Environment Variable and Default Value	***Purpose***
`ttynlines=24`	Number of text lines on screen
`ttyncols=80`	Number of text columns on screen
`terminal=gterm`	Type of terminal
`stdplot=lw`	Default plotting device (printer)
`stdimcur=stdimage`	Standard image cursor
`stdimage=imt512`	Image display device
`stdgraph=gterm`	Graphics display device
`printer=lw`	Output device
`multversions=no`	Keep multiple versions? (VMS only)
`imdir=tmp$`	Directory for storing `.pix` files (IRAF images)[a]
`editor=vi`	Editor invoked by typing “edit”
`clobber=no`	Overwrite files?
`cmbuflen=256000`	Graphics buffer size
`min_lenuserarea=20000`	Image header storage size
`nomwcs=yes`	Ignore WCS transformation?[b]

a. The value of `imdir` is site-dependent.
b. The value of `nomwcs` is often undefined by default.

Table A.1: IRAF Environment Variables and Default Values

CL Parameters

CL parameters control the behavior of the CL itself and are managed with the following commands:

- To set a CL parameter, type: *`parameter`*`=`*`value`*
- To display the current value of a parameter, type: `=`*`parameter`*
- To display the values of all CL parameters, type: `lparam cl`

CL Parameter	*Attribute*
`gcur=""`	Graphics cursor
`imcur=""`	Image cursor
`ukey=""`	Gloabl user terminal keyboard keylist
`(abbreviate=yes)`	Allow abbreviations in operand names?
`(echo=no)`	Echo CL command input on error?
`(ehinit="nostandout eol noverify"`	Ehistory options
`(epinit="standout showall"`	Eparam options
`(keeplog=no)`	Record interactive commands in log file?
`(logfile="home$logfile.cl")`	Name of log file, if used
`(logmode="commands nobackground noerrors notrace")`	Logging control options
`(lexmodes=yes)`	Enable conversation mode?
`(menus=yes)`	Display menu when changing package?
`(showtype=yes)`	Add task-type suffix in menus?
`(notify=yes)`	Message after background job completes?
`(szprcache=4)`	Size of process cache[a]
`(version="IRAF V2.10 EXPORT April 1992 revision 1")`	Current IRAF version
`(mode="ql")`	CL mode; (query or query and learn)
`(auto="a")`	Read only
`(query="q")`	Read only
`(hidden="h")`	Read only
`(learn="l)`	Reaad only
`(menu="m")`	Read only
`(b1)`	User variable, boolean
`(b2)`	User variable, boolean

CL Parameter	*Attribute*
`(b3)`	User variable, boolean
`(i)`	User variable, integer
`(j)`	User variable, integer
`(k)`	User variable, integer
`(x)`	User variable, floating point
`(y)`	User variable, floating point
`(z)`	User variable, floating point
`(s1)`	User variable, string
`(s2)`	User variable, string
`(s3)`	User variable, string
`(list="")`	List
`(line="..."`	Line

a. The process cache is often set to 3 on a VAX.

Table A.2: CL Parameters and Default Values

I/O Redirection

Redirection Command	*Function*
`>`	Send STDOUT to a file
`>>`	Append STDOUT to file
`|`	Pipe STDOUT of one task to STDIN of another
`>&`	Send output (STDOUT and STDERR) to a file

Table A.3: I/O Redirection Commands

Directory Specification

Character	*Represents...*
`$`	Logical directory (e.g., `uparm$logfile` may correspond to the VMS directory `[user.iraf.uparm]logfile`
`/`	Logical subdirectory seperator (e.g., `root/subdir` corresponds to `[root.subdir]`)

Table A.4: Characters Found in Directory Paths

History Editing

Command	*Function*
`history`	Display last 15 commands
`history` *n*	Display last *n* commands (*n* is now default)
`^`	Execute previous command
`^`*n*	Execute previous command *n*
`^ab`	Execute previous command beginning with "ab"
`ehist` *n*	Edit command *n* and execute after pressing Return
`ehist ?ab`	Edit previous command containing string "ab" and execute after pressing Return
taskname `^^`	Execute task *taskname* using first argument of previous command
taskname `^*`	Run task *taskname* using all arguments of previous command

Table A.5: History Editing Commands

Background Jobs

Command	*Purpose*
`&`	Submit a background job (e.g., `contour &`)
`jobs`	See what jobs are running in the background
`service` *n*	Respond to prompt from job *n*
`wait` *n*	Wait for background job *n* to complete
`kill` *n*	Terminate job *n*

Table A.6: Background Job Commands

Arithmetic and Logical Expressions

<table>
<tr><th>Operator</th><th>Function</th></tr>
<tr><td>+, −, *, /</td><td>Basic arithmetic functions</td></tr>
<tr><td>**</td><td>Exponentiation</td></tr>
<tr><td>//</td><td>String concatenation (e.g., ’ab’//’cde’ = ’abcde’</td></tr>
<tr><td>&&</td><td>Logical AND</td></tr>
<tr><td>||</td><td>Logical OR</td></tr>
<tr><td><</td><td>Less than</td></tr>
<tr><td>></td><td>Greater than</td></tr>
<tr><td><=</td><td>Less than or equal</td></tr>
<tr><td>>=</td><td>Greater than or equal</td></tr>
<tr><td>!=</td><td>Not equal</td></tr>
<tr><td>==</td><td>Equal</td></tr>
<tr><td>+=</td><td>Incremental sum (i +=1 ==> i=i+1)</td></tr>
<tr><td>−=</td><td>Incremental difference (i-=1 ==> i=i-1)</td></tr>
<tr><td>*=</td><td>Incremental product (i*=2 ==> i=i*2)</td></tr>
<tr><td>/=</td><td>Incremental division (i/=2 ==> i=i/2)</td></tr>
</table>

Table A.7: Arithmetic and Logical Operators

Mathematical Functions

Token	*Purpose*	*Mathematical Use*
`abs`	Absolute value	=abs(x)
`atan2`	Arc tangent	=atan2(x,y)
`cos`	Cosine	=cos(x)
`exp`	Exponentiation	=exp(x)
`frac`	Fractional part	=frac(x)
`int`	Integer part	=int(x)
`log`	Natural logarithm	log(x)
`log10`	Common logarithm	=log10(x)
`max`	Maximum value	=max(x,y,z...)
`min`	Minimum value	=min(x,y,z...)
`mod`	Modulus	=mod(x,base)
`radix`	Radix to any base	=radix(x,base)
`real`	Convert to real	=real(i)
`sin`	Sine	=sin(x)
`sqrt`	Square root	=sqrt(x)
`tan`	Tangent	=tan(x)

Table A.8: Mathematical Functions in CL

CL Language Components and Control Structures

Statement	*Function*
`begin`	Begins executable portion of a procedure script
`break`	Break out of loop
`case`	Possible switch setting
`default`	Default switch statement
`end`	Terminate script
`else`	Else clause for "if" statement
`for`	C-style "for" loop
`if`	If statement (condition)
`goto`	Unconditional jump
`next`	Start next iteration
`procedure`	Start a procedure script
`return`	Break out of a script and return a value
`switch`	Choose from a set of actions
`while`	Repeat while condition is true

Table A.9: Logical Control Statements in CL

String Functions

Operator	*Function*	*Usage*
`str`	Convert to string	s1=str(num)
`stridx`	Index of character in string	i=stridx(s1,'abc')
`substr`	Select substring from string	s1=substr(s2,3,7)

Table A.10: Built-in String Functions in CL

Data Types for Declarations

Symbol	*Data Type*	*Context*
`int`	Integer	Scalar and array
`real`	Real	Scalar and array
`char`	Character string	Scalar and array
`bool`	Boolean (yes or no)	Scalar and array
`file`	File name	
`struct`	Structured variable	
`gcur`	Graphics cursor	Internal to tasks
`imcur`	Image cursor	Internal to tasks

Table A.11: Data Types in Declarations

Optional Fields for Data Declarations

Option	*Purpose or Constraint*
`mode`	Auto, query, or hidden
`min`	Minimum legal value for range checking
`max`	Maximum legal value for range checking
`enum`	Enumerated list of allowable values
`prompt`	Prompt string

Table A.12: Range Checking Limits for Data Declarations

APPENDIX B:

Bibliography

STSDAS documentation is available from STScI's anonymous ftp host: `stsci.edu` in the directory `/documents/stsdas-docs/`. You can alse request printed versions by sending e-mail to the STSDAS group at: `hotseat@stsci.edu`

IRAF documentation is available from the IRAF group at NOAO via anonymous ftp to `tucana.noao.edu` in the directory `/iraf/docs/`.

FITS documentation is available from the NSSDC at Goddard Space Flight Center via anonymous ftp to `nssdca.gsfc.nasa.gov` in the directory `/FITS/`.

IRAF Documentation

Barnes, Jeannette, *A Beginner's Guide to Using IRAF*, August 1993.

The IRAF Data Reduction and Analysis System, (in IRAF Doc Set Vol. 1A).

A User's Introduction to the IRAF Command Language, (in IRAF Doc Set Vol. 1A).

Anderson, Ed, *An Introductory User's Guide to IRAF Scripts*, (in IRAF Doc Set Vol. 2B).

IRAF Doc Set Vol. 2B, includes cookbooks for many of the IRAF analysis packages.

STSDAS Documentation

STSDAS Users Guide. General information about STSDAS, tasks, packages, data formats, etc.

IGI Reference Manual. Detailed information and examples explaining how to use the Interactive Graphics Interpreter.

Synphot Users Guide. Detailed information and examples explaining how to do synthetic photometry.

STSDAS Calibration Guide. Detailed descriptions of how to calibrate data from the HST. This will be replaced in early 1994 by the *HST Data Handbook*.

STScI Site Guide for IRAF and STSDAS. Contains site-specific information on available devices, IRAF installation, etc.

STSDAS Site Manager's Installation Guide and Reference. Instructions for retrieving and installing STSDAS.

Interface Control Document 19—PODPS to SDAS. Defines all STSDAS data structures except for the Fine Guidance Sensors.

Interface Control Document 46—AEDP to PODPS. Defines the structure of the astrometry data files from the Fine Guidance Sensors.

Interface Control Document 47—PODPS to CDBS. Defines the structure of all CDBS reference data files.

Calibration Data Base Data Design. Describes the structure of all CDBS data base relations.

FITS Format

"Implementation of the Flexible Image Transport System (FITS)," 1991, NASA/OSSA Office of Standards and Technology, Code 993, NASA Goddard Space Flight Center, Greenbelt, Maryland 20771.

Wells, D.C., Greisen, E.W., and Harten, R.H., 1981, *Astron. Astrophys. Suppl*., 44, 363, "FITS: A Flexible Image Transport System."

Greisen, E.W., and Harten, R.H., 1981, *Astron. Astrophys. Suppl.*, 44, 371, "An Extension of FITS for Groups of Small Arrays of Data."

Grosbol, P., Harten, R.H., Greisen, E.W., and Wells, D.C., 1988, *Astron. Astrophys. Suppl.*. 73, 359, "Generalized Extensions and Blocking Factors for FITS."

Harten, R.H., Grosbol, P., Greisen, E.W., and Wells, D.C., 1988, *Astron. Astrophys. Suppl.*, 73, 365, "The FITS Table Extension."

Image Restoration References

Kondo, K., Ichioka, Y., Suzuki, T., 1977, "Image Restoration by Wiener Filtering in the Presence of Signal-Dependent Noise," in *Applied Optics*, 16, page 2554.

Busko, I.C., 1991, "Weiner Restoration of HST Images: Signal Models and Photometric Behavior," in *Proc. 1st Annual Conf. on Astr. Data Analysis Software and Systems*, Tucson, Arizona.

Bates, R.H.T., and McDonnell, M.J., 1986, *Image Restoration and Reconstruction*, Clarendon Press, Oxford.

Andrews, H.C., and Hunt, B.R., 1977, *Digital Image Restoration*, Prentice-Hall, Englewood Cliffs, New Jersey.

Whal, F.M., 1987, *Digital Image Signal Processing*, Artech House, London.

Pratt, W.K., 1978, *Digital Image Processing*, John Wiley and Sons, New York.

Lagendijk, R.L., Biemond, J., and Boekee, D.E., 1986, "Iterative Image Restoration with Ringing Reduction," in *Signal Processing III: Theories and Applications*, I.T. Young et al. (ed.), Elsevier Science Publishers B.V., North-Holland.

White, R.L. and R.J. Allen (eds.), *Restoration of HST Images and Spectra*, Proceedings of a Workshop help at the Space Telescope Science Institute, August 1990.

Hanisch, R.J. (ed.), "Restoration: Newsletter of STScI's Image Restoration Project," Summer 1993, Space Telescope Science Institute.

Jain, A.K., 1989, *Fundamentals of Digital Image Processin*g, Prentice-Hall.

Stark, H., 1987, *Image Recovery: Theory and Applications*, Academic Press.

References for Statistical Analysis Methods

Avni, Y., Soltan, A., Tananbaum, H. and Zamorani, G., 1980, *Astrophys. J.*, 235, 694.

Avni, Y. and Tananbaum, H., 1986, *Astrophys. J.*, 305, 57.

Feigelson, E.D. and Nelson, P.I., 1985, *Astrophys. J.*, 293, 192.

Isobe, T., Feigelson, E.D., and Nelson, P.I., 1986, *Astrophys. J.*, 306, 490.

Lawless, J.F., 1982, *Statistical Models and Methods for Lifetime Data* (Wiley: New York).

Schmitt, J.H.M.M., 1985, *Astrophys. J.*, 293, 178.

Schneider, H., 1986, *Truncated and Censored Samples Normal Populations*, (Dekker: New York).

Wagner, A.E. and Meeker, W.Q. Jr., 1985, in *1985 Proceedings of the Statistical Computing Section*, (Amer. Stat. Assn.: Washington, DC), p. 441.

Synthetic Photometry

Horne, K., 1988, “Application of Synthetic Photometry Techniques to Space Telescope Calibration”, in *New Directions in Spectroscopy*, p. 145, L. Davis Press.

Koorneef, J., et al., 1986, “Synthetic Photometry and the Calibration of the Hubble Space Telescope”, in *Highlights of Astronomy*, p. 833, Reidel, Dordrecht Holland.

Synphot Users Guide. 1993. Detailed information and examples explaining how to do synthetic photometry in STSDAS.

APPENDIX C: Glossary

The following terms and acronyms are used in this manual and throughout the IRAF and STSDAS systems.

A/D: Analog to digital conversion.

background: Non-interactive execution of tasks. Background tasks can be used to run long-running tasks, freeing the terminal for other work. Background can also mean non-feature data in an image.

bug: An error in a program that causes incorrect results or prevents execution.

cache: Small block of quick-access memory in which the CL stores a task to allow faster initiation.

CCD: Charge-coupled device.

CDBS: Calibration Data Base System.

CL: The IRAF command language.

command: Any statement in the CL, including invoking a task or loading a package.

COSTAR: Corrective Optics Space Telescope Axial Replacement.

CTIO: Cerro Tololo Inter-American Observatory.

DE: Detector efficiency.

directory: A logical grouping of files within the hierarchical file system.

DQF: Data quality file. Mask image of bad or suspect pixel locations.

executable: Binary program that can be loaded and executed.

extension: The last portion of a file name, usually delineated with a period.

f77vos: Fortran-77 virtual operating system. IRAF interface for Fortran, developed at STScI.

FGS: Fine Guidance Sensors.

file system: The collection of files available to the system, the organization of these files, their attributes, and the physical devices on which they reside.

flush: Remove tasks from the cache. Clear the memory.

FOC: Faint Object Camera. Instrument aboard HST.

FOS: Faint Object Spectrograph. Instrument aboard HST.

foreign task: Any program that does not use the IRAF interface, such as a host system task, Fortran or C program.

GEIS: Generic Edited Information Set. The image format used for HST images. Also referred to as "multigroup STSDAS images."

GIMP: Geomagnetically-Induced Motion Problem. Shift in FOS images.

GIO: Graphics input/output.

GPB: Group parameter block. Part of STSDAS group-format file (see "Data Formats" on page 37 for details).

graphics: Visual display, usually referring to line drawings on a standard terminal or workstation window.

Guide Star Catalog: Sometimes referred to as *GSC*. A sky survey containing positions of over 20 million objects.

header: The part of an image data structure that describes the attributes of an image.

help: CL facility used to provide information about tasks or packages. Similar to the Unix man page facility.

hidden: Attribute of a parameter causing the CL to use default values rather than prompting the user for input. Sometimes referred to as "keyword task parameter."

HotSeat: STSDAS user support service.

HRS: High Resolution Spectrograph. Instrument aboard HST. Also known as "GHRS."

HSI: Host System Interface. The lowest level of the interface between IRAF and the host operating system. The system-dependent part of IRAF.

HSP: High Speed Photometer. Instrument aboard HST.

HST: Hubble Space Telescope.

IDT: Image dissector tube (in HSP).

image: Data structure containing information usable by IRAF tasks.

image section: A portion of an image, specified by additional syntax in the file name.

interactive: Permitting communication between a user and an active program.

IPC: Inter-Process Communication. The connection between tasks or processes that allows output from one task to be used as input to another.

IRAF: Image Reduction and Analysis Facility.

keyword task parameter: see "hidden."

learn: CL feature that retains parameter values from run to run.

MEM: Maximum Entropy Method. Commonly-used algorithm for restoring blurred images.

mode: Attribute of parameters, tasks, and packages that determines how prompting will be handled.

NOAO: National Optical Astronomy Observatories.

NSSDC: National Space Science Data Center; at NASA/Goddard Space Flight Center.

OIF: Old IRAF format. The native image data format for IRAF (see also "GEIS image").

OS: Operating system.

OTA: Optical Telescope Assembly.

package: A collection of tasks and subpackages within the IRAF system.

PAD: Pulse amplifier and discriminator. (On HSP).

parameter: A value passed to a command or task. An argument.

parse: To distinguish between different components of a string or command line.

path: The list of directories through which the system must navigate to find a file.

PDA: Photon Detector Assembly.

pipe: The CL facility that allows output from one task to be used as input to another. Similar features exist in some operating systems, such as Unix.

pixel: Picture element. A value in image data.

PMT: Photomultiplier tube.

PODPS: Post Observation Data Processing System. The standard HST pipeline process.

procedure: A CL script, installed as a task.

process: A program executing on a host computer.

prompt: A string displayed by the CL indicating that the system is waiting for user input.

PROS: Post Reduction Offline Software. IRAF external package for reducing X-ray data.

pset: Parameter set. A group of parameters treated as a single task or parameter.

PSF: Point Spread Function. Used as input image for image restoration tasks.

psikern: PostScript kernel.

QPOE: Quick Position Oriented Event. IRAF data format used in the PROS package for storing X-ray data.

query: Parameter mode that forces prompting.

redirection: Facility or feature that allows output from a task to be saved as a file.

RSDP: Routine Science Data Processing. The calibration pipeline.

SAA: South Atlantic Anomaly.

script: Executable set of CL commands, definable as a task.

SDA: Science Data Analysts. STScI staff members available to assist observers with data analysis.

S/N: Signal-to-noise ratio.

SOGS: Science Operations Ground System.

SPP: Subset Preprocessor. The IRAF programming language.

SSE: Star selector encoder.

STDERR: Standard error. The channel through which error messages are sent to the user's terminal. The CL text error stream.

STDIN: Standard input. The CL text input stream. The channel through which the user passes input to IRAF, usually defined as the keyboard.

STDOUT: Standard output. The CL text stream through which output is returned from the program to the user. Usually directed to the terminal.

STEIS: Space Telescope Electronic Information Service. System from which information, documentation, reference files and other resources pertaining to HST can be obtained using anonymous FTP, gopher, or WWW clients.

STF: Space Telescope format. This is the GEIS image data format used for HST data.

STSDAS: Space Telescope Data Analysis System. An IRAF external package tailored to analyzing data from the Hubble Space Telescope.

SVD: Singular value decomposition. Polynomial fitting algorithm.

table: Data structure organized by rows and columns.

task: A command or program executable from within the IRAF CL.

TIM: Telescope Image Modelling. Software used to generate PSF images for HST data.

UDL: Unique data log. File in HST dataset.

unlearn: Restore a task parameter to its default value.

uparm: Directory containing user parameters. Also an environment variable pointing to the user parameter directory.

VFN: Virtual File Name. The IRAF name for a file, possibly different from the host operating system file name.

VMS: Operating system used by VAX computers.

VOS: Virtual Operating System. The libraries of procedures that defines the interface between an application and IRAF.

WF/PC: Wide Field/Planetary Camera. Instrument aboard HST.

WWW: World Wide Web. Hypertext-oriented method of storing and retrieving information over the Internet using client software such as NCSA Mosaic.

Index

U

V

W

X