

The New HST FITS Format, the FITS Kernel and OpenIRAF

P. Greenfield

Space Telescope Science Institute, Baltimore, MD 21218

Abstract. The pipeline calibration of STIS and NICMOS involves a number of changes from how previous HST instrument data were calibrated in the pipeline. These changes include a change in the data formats, what software language and libraries were used to write the programs, and the methods used to access the new data format. This paper will summarize the new STIS and NICMOS data formats, how to access these files using the new FITS kernel in IRAF, and the changes in how the software is written.

1. Introduction

The pipeline calibration of STIS and NICMOS data involves many changes from previous HST calibration pipelines for both the software and the data files. This paper will review what those changes are, particularly those that affect HST users most directly. The most obvious change is the data format used. Previous instrument data has been processed in GEIS (Generic Edited Information Set)—often referred to as ST format—and then converted to FITS format for the archive. STIS and NICMOS data files are now being processed as FITS files directly. The raw and processed results are stored directly in the archive with no conversions in the format. The new format and conventions will be described in detail.

Changes to the IRAF image kernel (the interface that handles image input and output) were necessary to recognize the FITS format directly. The new IRAF fits kernel is used by the calibration pipeline tasks to access the image files. The same fits kernel allows IRAF users to directly access the data products without converting them into other formats. While adding much power, some complications result that IRAF/STSDAS users should be aware of. This paper will describe some of these.

The new pipeline tasks have been written in the C programming language using aspects of the OpenIRAF effort. A brief description of what this entails and means for future STSDAS and TABLES development is included.

2. STIS and NICMOS Data Formats

Four basic reasons drove the decision to use multi-extension FITS files as the basic processing format. The first is that the FITS format is machine independent. The products can be moved from one computer to another without requiring the user to convert the file as is currently required for GEIS files. An added benefit is that, should the type of computer the pipeline is run on change, the software used to generate the raw data files will not have to be modified (the software producing the raw data files for WFPC2 and FOC would need to be modified).

The second reason is that no conversion is needed to store the data products in the HST archive, nor is one needed to convert retrievals of STIS or NICMOS data from the archive to a format useful for processing. The third reason is related; current GEIS files are converted to FITS for the archive, but the FITS files that result are not easily used by

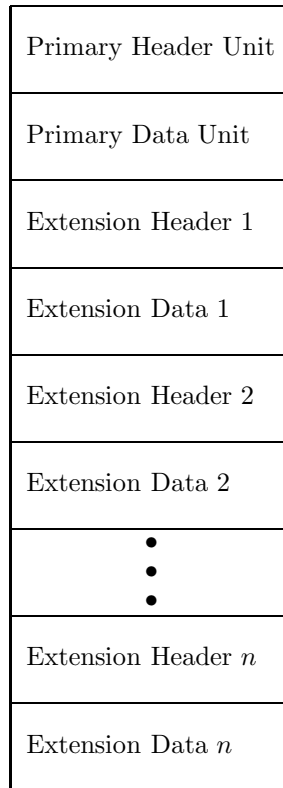


Figure 1. Organization of FITS files containing extensions.

other systems since the data have been encoded in a odd way so as not to lose information on conversion back to GEIS format. For example, the FITS file for WFPC2 contains a three dimensional array where the four images have been combined into a data cube. All the group parameters are stored in a table extension making their access very inconvenient. None of these problems exist with the use of the multi-extension FITS files. No rearranging of the data is required.

Lastly, unlike GEIS files, it is possible to group data of different type or size in the same file. This allows much more flexibility in the content of data files.

Figure 1 illustrates the general layout of multi-extension FITS files. It essentially consists of repeated Header/Data Units (HDU). The first is referred to the Primary HDU and all subsequent ones as Extension HDUs. It is allowable that some of the Data Units are absent (or ‘null’ in our terminology) as long as that is indicated in the corresponding header by setting the required keyword NAXIS equal to 0. There are currently three standard types for extensions: IMAGE, TABLE (otherwise referred to as ASCII tables), and BINTABLE (binary tables). The PDU (Primary Data Unit) is always assumed to be of type IMAGE.

STScI has adopted some local conventions regarding its use of multi-extension FITS files.

- Use of associated data (IMSETS).
- Extension naming (using EXTNAME and EXTVER).
- Null Primary Data array, all data in image extensions.
- Keyword inheritance.

		Primary Header (Null Data Array)
EXTNAME =	SCI	Science Image (float)
	ERR	Error Image (float)
	DQ	Data Quality Image (16-bit integer)
	SAMP	Samples Image (16-bit integer)
	TIME	Exposure Time Image (float)

Figure 2. Schematic outline of NICMOS data file containing one IMSET.

- Associated arrays with constant values represented by null arrays.

Extension naming and keyword inheritance are an integral part of the IRAF FITS kernel. Each of the above conventions will be described in turn.

STIS and NICMOS data consists of groups of arrays called ‘IMSETS.’ Each IMSET consists of a standard set of arrays, always present. STIS IMSETS consist of a science image, an error image, and a data quality image. NICMOS IMSETS consist of these three plus an additional two: a samples array and an integration time array. Figure 2 shows a schematic example of a NICMOS data file. STIS or NICMOS files may contain multiple IMSETS.

Extension naming means that the FITS file extensions can be referred to by name (the Primary HDU cannot be named under this scheme). We felt this necessary since requiring users to remember the numerical place of an extension in a file placed too great a burden on them, particularly when different types of extensions were combined in a data file. The scheme uses a name and number to identify unique extensions. In our scheme, the name is used to identify a component of an IMSET (e.g., SCI for science image), and the number to identify which IMSET (not to be confused with the extension number). The IMSET number does not necessarily—though usually does—reflect the order of the IMSET in the file. The FITS kernel will choose the first extension which matches the specified name. If the EXTVER (number component) is not specified, the kernel will match the first extension that matches the name component only. Extensions names are saved with the case as typed, but case is ignored when matching (e.g., Sci will match SCI).

So that all data elements can be referred to by name, STIS and NICMOS data files always leave the PDU null.

Keyword inheritance is a mechanism that allows the keywords in the Primary Header to be logically applied to all extension image headers. There are various rules about when keyword inheritance can be used that won’t be detailed here, but all STIS and NICMOS

Row	SPORDER	NELEM	WAVELENGTH	...	NET	FLUX	ERROR	DQ
1	66	1024	< array >	...	< array >	< array >	< array >	< array >
2	67	1024	< array >	...	< array >	< array >	< array >	< array >
3	68	1024	< array >	...	< array >	< array >	< array >	< array >
...

Figure 3. Structure of a STIS extracted spectra table. Two columns, GROSS and BACKGROUND, are not shown to reduce the printed size.

data make use of this feature. The FITS kernel, unless instructed otherwise, will combine the Primary Header with the Extension Header when accessing an extension. If the same keyword exists in both, the value from the extension header is used.

In a number of cases, particularly for NICMOS, some of the associated arrays in an IMSET may consist of a constant value for all the pixels. To save space, these extensions consist of a null data array. The necessary information to reconstruct the array is contained in the corresponding extension header. While some STSDAS tasks may recognize this convention, in general IRAF tasks will not.

For the time being, the data format for the old instruments (WFPC2 and FOC) is unchanged. That is to say, files are processed in GEIS format and converted to FITS. The possibility of converting the processing format to one using multi-extension FITS files is being considered, but no decisions have been made. Such a change requires significant system work so that existing WFPC2 tasks would continue to work.

Both STIS and NICMOS are using FITS BINTABLE format to store extracted 1-D spectra. They do so using the so called ‘3-D table’ format where each 1-D array is stored in a table cell rather than a table column. Thus, each row of the table contains a complete spectrum, as well as other associated arrays such as wavelength, error, and data quality. Figure 3 shows an example of a STIS extracted spectra table. Such FITS table extensions are accessed using the same naming convention as used for images. The TABLES package can now directly access FITS tables though many tasks do not yet deal with 3-D tables except for a few (they all will eventually). There are a number of tasks to translate 1-D data to and from 3-D tables so that such data may be used with existing IRAF tasks that do not handle 3-D tables.

3. IRAF/STSDAS/TABLES support for the New Data Formats

3.1. FITS image kernel

The IRAF FITS image kernel allows all existing IRAF tasks to directly access FITS image data in the same way it can access .imh or .hhh files—no conversions are necessary. However, with this convenience and power come some disadvantages.

The internals of the FITS kernel are quite complicated and hence, the initial versions are somewhat buggy. Since FITS files may now contain many image (or table) objects, users may become confused about the contents of the file. Because there are now multiple objects to worry about, there are unavoidable complications to the user interface. Finally, certain operations, such as inserting images into the middle of a FITS file with many large extensions, may trigger lengthy file rewrites. The potential problems will be discussed in more detail in the following discussions on how to use the FITS kernel.

First, certain IRAF environmental variables affect the behavior of the FITS kernel. For details on their use, refer to the IRAF 2.11 release notes:

<http://iraf.noao.edu/iraf/web/v211revs.html>

imtype —specify a default image type. The behavior is changed from older versions of IRAF. Previously, IRAF would only use the value of **imtype** when creating new files from scratch. When producing output files based on modified input files (such as **imcopy** or **imcalc**), it would instead use the input file type to set the output file type unless the output filename explicitly specified a different image file format. Now it will by default always use the value of **imtype**, regardless of the input file format unless the **inherit** option for **imtype** is specified. This **inherit** option has nothing to do with keyword inheritance.

imextn —This variable is used to identify the image format of a file corresponding to the file's extension name. This can be used to override IRAF's defaults for identifying which format it should assume for a given filename extension convention. For example, it would be possible to tell IRAF to treat all files ending in **.f** as fits files. Wildcards are permitted.

imlobber —If this is set to **yes**, then IRAF will overwrite output files if they already exist; the default setting is **no**, in which case an error results.

fkinit —This is used to set default behavior for the FITS kernel

FITS image extensions may be selected in a number of ways. The simplest is to specify the extension by extension number (0=Primary, 1=first extension, etc.)

```
infile.fits[3]
```

Selecting by extension name:

```
infile.fits[SCI,2]
```

In any case, selecting the specific extension is required by IRAF for multi-extension files. All HST FITS file are multi-extension, including those made from GEIS files.

Unfortunately, the task of filename specification has become more complicated for three reasons. First, an extension must be specified; second, for output files, guidance must be given about how or where to write the extension (start a new file, or append or overwrite an extension in an existing file?); and finally, how keyword inheritance should be handled. To handle these and other issues there are the following options that can be used when specifying filenames.

(no)inherit —combine Primary Header keywords with Extension Header keywords (or not) regardless of the default implied.

append —append output FITS image to an existing FITS file.

overwrite —overwrite an existing FITS image extension in an existing file.

ehulines —preallocate some number of blank lines in headers to allow room for future growth.

padlines —add extra blank lines to header on copy of header.

Plus some others that most should never need.

Note that keyword inheritance introduces some problems of its own. While it is useful for the original set of data, any operations on that set of data that propagate the extensions *en masse* to a new output image will effectively defeat the careful separation of the global keywords (those in the primary header) and the local ones (those in the extension header) since IRAF will combine them before writing the image out. Thus, the primary keywords will appear repeated in each of the output extensions. It is possible to keep them segregated by using only tasks that do this specifically (e.g., `calstis` or `calnic`) or by using the `noinherit` option on the input files and the `inherit` option on the output file. This last method can be quite tedious and error prone though. Most users should not bother trying to keep the global and local keywords separate.

Another potential problem with keyword inheritance is confusion about what is being changed. An example is editing a keyword that affects `calstis` or `calnic` processing in one of the science images not realizing that the keyword is a global one. A new one will appear in the extension header and then be ignored by the calibration pipeline task as it uses its old value which is still in the primary header.

IRAF keeps an internal cache of where the file extensions are located within a file. Use of non-IRAF or non-image file utilities (such as using the Unix `rm` command or IRAF's `del` command) may confuse the kernel and lead to errors. Expanding a header or overwriting an existing extension with a larger one (or even smaller one) will lead to the rewriting of the file and possibly long delays. The current kernel prevents updating multiple extensions simultaneously. It is necessary for tasks needing to do so to read in to memory all such arrays or create temporary output files in order to get around this limitation.

3.2. Some Examples

Use of the FITS kernel can be comparatively simple as the following couple of examples illustrate.

```
imcopy infile.fits[1] outfile.fits
```

Here the first extension is copied to a new file if one does not already exist or overwrites the old one if `imclobber=yes`.

```
imstatistics infile.fits[sci,1]
```

This prints image statistics for the image extension that has `EXTNAME=sci` (or `SCI`) and `EXTVER=1`.

But one can see quite complicated examples as well:

```
imcopy in.fits[SCI,5] [noinherit] [100:400,*]
      out.fits[6] [overwrite,inherit]
```

This copies a section of an image extension with `EXTNAME=SCI` and `EXTVER=1` without inheriting the keywords from the primary to the 6th extension of an existing FITS file, overwriting the existing extension, and setting it to inherit the primary keywords from the output file.

```
imsum in.fits[1],in.fits[2],in.fits[3] out.fits[sum123, append,inherit]}
```

This computes the sum of the first 3 extensions of `in.fits` and append the result in the existing `out.fits` file, and sets the extension to inherit the keywords from the primary header of the output file.

3.3. Tools for dealing with multi-extension FITS files.

stdas.fitsio.catfits —Use this to list the contents of a FITS file.

stdas.toolbox.imgtools.mstools —This package contains tasks that can handle imsets.

ecdel —Delete an entire class of FITS extensions from a FITS file (e.g., all NICMOS TIME extensions).

ecextract —Select all extensions of one selected ‘type’ from a FITS file (e.g., all SCI extensions).

extdel —Delete single extensions from a FITS file.

msarith —Image arithmetic with STIS and NICMOS files which uses and updates the data quality and error extensions.

mscombine —Combine NICMOS or STIS files using gcombine and appropriately using the data quality and error extensions.

msdel —Remove one entire IMSET from a FITS file.

mssort —Sort a FITS file to order extensions by EXTVER number.

msjoin —Join files containing single IMSETS into one file.

mssplit —Splits multi IMSET NICMOS or STIS files into separate files containing one IMSET each.

msstatistics —An extended gstatistics for .imh, .hhh, and .fits files which can take data quality and error arrays into account.

In addition, NOAO is developing a package of FITS utilities (`fitsutil`, currently an external package). However, they generally do not recognize our IMSET conventions.

fxcopy —Copy FITS files or FITS extensions to an output FITS file.

fxdelete —Delete FITS extensions in place.

fxinsert —Insert FITS files or extensions into another MEF file.

fxheader —List one line of header description per FITS unit.

fxsplit —Split a multiple extension FITS file into single FITS files.

fxdummyh —Create a dataless single FITS file.

The external package is in: `ftp://iraf.noao.edu/pub/fitsutil.tar.Z`

3.4. Sources of information about the FITS kernel

STScI User’s Guide to the IRAF FITS kernel

`http://ra.stsci.edu/stdas/fits_userguide.html`

NOAO’s Fits Kernel User’s Guide

`ftp://iraf.noao.edu/iraf/docs/fits_userguide.txt`

IRAF 2.11 Release Notes

`http://iraf.noao.edu/iraf/web/v211revs.html`

STSDAS manuals are being updated; the new versions will be available in 1997.

3.5. Using FITS BINTABLEs

A row selector mechanism is available for some tasks (`tscopy`, `sgraph`, `igi`) to allow the selection of specific rows so that 1-D data in 3-D tables can be used with tasks that accept 1-D data. The row selector mechanism will be incorporated into the TABLES interface in the next release so that all tasks can incorporate the row selector mechanism though not all tasks may use 3-D tables by then.

Tasks that provide a means to convert 3-D tables to and from other forms or formats include:

tiimage —Insert images into rows of a 3-D table.

titable —Insert 2-D tables into rows of a 3-D table.

tximage —Extract images from rows of 3-D tables.

txtable —Extract 2-D tables from rows of 3-D tables.

tscopy —Copy row/column subsets of tables using selectors.

stsdas.hst.calib.ctools.tomultispec —Convert spectra from rows of a fits 3-D binary table to an IRAF multispec image.

Further information on row and column selectors can be obtained by typing **help selectors**.

4. Elements of OpenIRAF

OpenIRAF represents an effort to make it easier to import outside software into IRAF as well as making it easier to use IRAF tasks outside the usual IRAF environment. One of the first steps in this direction has been the development of a C language interface to the IRAF libraries. Such C bindings have been developed for the major IRAF libraries and are distributed with STSDAS v2.0.

All the STIS and NICMOS calibration pipeline tasks are written in ANSI C with no dependence on vendor extensions. In general, these C tasks can be compiled as traditional native IRAF tasks (runnable from the CL or host level) or as standalone tasks that are linked with the host LIBC. Currently, the pipeline tasks are compiled as standalone tasks.

All the image file I/O is handled by a higher level library—HSTIO—which is layered on the IRAF imio library. This serves to minimize the dependence on the software environment. All image data are mapped into C data structures which correspond to all important data objects from data arrays (of various types), image extensions, IMSETS, and files. This library provides convenient routines to read and write to such data structures—high level and low level—as well as accessing sections of arrays. A more generalized version is being developed to handle accessing other data formats (e.g., GEIS) and incomplete IMSETS so that programs written using the library can be used for general purpose analysis.

Documentation on the C interface and the HSTIO library may be found in the STSDAS v2.0 subdirectories `lib/cvos/doc` and `lib/hstio/doc`.

Acknowledgments. The work described here involved the efforts of many people including all of the Science Software Group (Phil Hodge, Howard Bushouse, Allen Farris, Ivo, Busko, J. C. Hsu, Michele de la Pena, Warren Hack, Bernie Simon, Ray Williamson, Eric Wyckoff, and Rick White) as well as considerable contributions from the STScI instrument groups, NICMOS and STIS IDT's, OPUS, Hubble Data Archive, and the NOAO IRAF group.