A Descritpion of the HSTIO Interface to the STIS and NICMOS Pipelines

Allen Farris
Space Telescope Science Institute
E-mail: farris@stsci.edu

August 21, 1996

1 The HSTIO Header File

The hstio.h File

```
# if !defined(HSTIO_)
# define HSTIO_

#if defined(__cplusplus)
extern "C" {
# endif

/*
** Data Structures and I/O Function Declarations for
** STScI Calibration Pipeline Software for STIS and NICMOS
**
** Version 1.20
**
** Nov. 29, 1995
**
** Table of Contents
**      Basic structures to represent two-dimensional arrays.
**      Definitions of Science, Data Quality and Error data.
**      Macros to implement 2-d array indexing.
**      Definition of data sections.
**      Definition of Header "card" image arrays.
**      Definition of I/O descriptor.
**      Definitions of Single and Multi-Group data structures.
**      Declarations of error handling functions.
**      Declarations of high-level I/O functions.
**      Declarations of header manipulation functions.
**      Declarations of low-level I/O functions.
**      Declarations of functions to initialize, allocate and free
**              data storage contained within data structures.
**
** History
** Version 1.00 (06/21/95).
** Revisions to version 1.00 to produce version 1.01 (08/03/95).
**      Corrected bug in definition of IHdr.
```

```
**      Changed declaration of putMultiGroup function.
**      Changed the ordering within structures to Sci, Err, Dq.
** Revisions to version 1.01 to produce version 1.10 (09/28/95).
**      Corrected bug in definition of OUT_OPTION.
**      Added ifdef extern C to accomodate C++.
**      Added support for NICMOS samples and integration time data.
**      Made IODesc structure private by changing to void pointer
**          and changed the definitions of the IO_OPTIONS and added
**          functions to retrieve data from the IODesc structure.
**      Added open and close FITS file functions.
**      Major revision to functions to manipulate header arrays.
**      Added additional error handling functions
**      Revised definition of single and multi-group structures
**          to achieve greater compatibility between the two
** Revisions to version 1.10 to produce version 1.11 (09/29/95).
**      Added an option parameter to the putGroupHdr functions.
** Revisions to version 1.11 to produce version 1.20 (11/29/95).
**      Major revisions to the low-level I/O function and to the
**          low-level data structures.  References to Sci, Err, and
**          DQ were removed from the low-level structure and function
**          names in favor of generic names such as float and short.
**
**
*/

# include <stdlib.h>

typedef struct {
        short *buffer;          /* the pointer to the beg. of the buffer */
        int buffer_size;        /* the size of the full 2-d array in the */
        int tot_nx;             /* buffer.                               */
        int tot_ny;             /*     buffer_size = tot_nx*tot_ny       */
        int nx;                 /* The size of the current "view" of the */
        int ny;                 /* full 2-d array.                       */
        short *data;            /* The pointer to the beginning of the   */
                                /* subsection of the full 2-d array.     */
} ShortTwoDArray;

typedef struct {
        float *buffer;
        int buffer_size;
        int tot_nx;
        int tot_ny;
        int nx;
        int ny;
        float *data;
} FloatTwoDArray;


/*
** The Following macros are used to represent 2-d indexing.
** Two dimensional arrays are stored in FITS order.
**
**        ny
**        ^
**      N | a05   a15   a25   a35
**      A | a04   a14   a24   a34
**      X | a03   a13   a23   a33
**      I | a02   a12   a22   a32
**      S | a01   a11   a21   a31
**      2 | a00   a10   a20   a30
**         ---------------------------> nx
**            NAXIS1
**
**      NAXIS1 is 4 and NAXIS2 is 6
**      PIX(a,1,4) accesses a14
*/

```

```
# define Pix(a,i,j)      (a).data[(j)*(a).tot_nx + (i)]
# define DQPix(a,i,j)    (a).data[(j)*(a).tot_nx + (i)]
# define DQSetPix(a,i,j,value) (a).data[(j)*(a).tot_nx + (i)] = (value)

# define PPix(a,i,j)      (a)->data[(j)*(a)->tot_nx + (i)]
# define PDQPix(a,i,j)    (a)->data[(j)*(a)->tot_nx + (i)]
# define PDQSetPix(a,i,j,value) (a)->data[(j)*(a)->tot_nx + (i)] = (value)

/*
**              Examples of using macros.
**
** Suppose we have the following declarations and that these data
** structures have been properly initialized and allocated.
**
**      SciData picture;
**      DQData dataqual_in1, dataqual_in2, dataqual_out;
**      short val;
**
** Now set all the diagonal pixels in the picture array to 1.
**
**      for (i = 0; i < min(picture.nx,picture.ny); i++)
**              Pix(picture,i,i) = 1.0F;
**
** Now combine the two data quality arrays.
**
**      for ( j = 0; j < dataqual_out.ny; j++)
**          for (i = 0; i < dataqual_out.nx; i++) {
**              val = DQPix(dataqual_in1,i,j) | DQPix(dataqual_in2,i,j);
**              DQSetPix(dataqual_out,i,j,val);
**          }
**
*/

/*
** The data section structure can be used to read sections of an image
** from disk into memory.  It is intended to be used only for high-level
** I/O operations.  At present, the entire array is read into memory.
*/
typedef struct {
        int x_beg;              /* The beginning coordinates of the     */
        int y_beg;              /* section.                             */
        int sx;                 /* The sizes of the X and Y dimensions  */
        int sy;                 /* of the section.                      */
} DataSection;

# define HDRSize 81
typedef char HdrArray[HDRSize]; /* Headers are simply an array of fixed */
                                /* length, null terminated strings that */
                                /* represent FITS card images.          */

typedef struct {
        int nlines;             /* The number of lines actually used.   */
        int nalloc;             /* Number of lines currently allocated. */
        HdrArray *array;        /* The buffer of card images.           */
} Hdr;

/*
** I/O Options and the I/O Descriptor Pointer
**
** The I/O descriptor pointer points to an I/O structure that is
** used internally within the I/O functions.
*/
# if !defined(IO_OPTIONS_)
# define IO_OPTIONS_
enum IO_OPTION_ {
        Sequential = 0x1,
        Random = 0x2,
        ReadOnly = 0x4,
```

```
        WriteOnly = 0x8,
        ReadWrite = 0x10,
        New = 0x20,
        Overwrite = 0x40,
        Force = 0x80,
        Inherit = 0x100,
        Dupname = 0x200,
        NoDupname = 0x400,
        RemainOpen = 0x800,
        Geis = 0x1000,
        Tape9 = 0x2000,
        Std = 0x4000,
        Disk = 0x8000
};
typedef enum IO_OPTION_ OP_OPTION;
# endif
typedef void *IODescPtr;

/*
** The following data structures define combinations of both
** headers and data.
*/
typedef struct {
        IODescPtr iodesc;
        DataSection section;
        Hdr hdr;
        FloatTwoDArray data;
} FloatHdrData;

typedef struct {
        IODescPtr iodesc;
        DataSection section;
        Hdr hdr;
        ShortTwoDArray data;
} ShortHdrData;

typedef FloatHdrData SciHdrData;
typedef FloatHdrData ErrHdrData;
typedef ShortHdrData DQHdrData;
typedef ShortHdrData SmplHdrData;
typedef FloatHdrData IntgHdrData;

/*
** The SingleGroup data structure is used for single images or single
** groups from a multi-group file.  The disk file name together with
** the group number uniquely identifies the source of the data array
** in the file.  The group number is the FITS EXTVER keyword value.
*/
typedef struct {
        char *filename;
        int group_num;
        Hdr *globalhdr;
        SciHdrData sci;
        ErrHdrData err;
        DQHdrData dq;
} SingleGroup;

typedef struct {
        char *filename;
        int group_num;
        Hdr *globalhdr;
        SciHdrData sci;
        ErrHdrData err;
        DQHdrData dq;
        SmplHdrData smpl;
        IntgHdrData intg;
} SingleNicmosGroup;

```

```
/*
** The MultiGroup data structure is used to read members of a multi-group
** file.
*/
typedef struct {
        int ngroups;            /* number of groups contained in the struct  */
        SingleGroup *group;
} MultiGroup;

typedef struct {
        int ngroups;            /* number of groups contained in the struct  */
        SingleNicmosGroup *group;
} MultiNicmosGroup;

/*
** Error Handling Function Declarations
*/
enum HSTIOERROR_ {
        OK, NOMEM, BADOPEN, BADCLOSE, BADREAD, BADWRITE, BADEXTNAME,
        BADHSIZE, NOGET, NOPUT, BADDIMS, BADTYPE, NOSCI, BADSCIDIMS,
        BADGROUP, BADGET, BADFITSEQ, BADFITSQUOTE, BADFITSNUMERIC,
        BADFITSTYPE, BADPUT, BADNAME, BADBITPIX, BADNDIM, BADEXIST
};
typedef enum HSTIOERROR_ HSTIOError;
HSTIOError hstio_err();
char *hstio_errmsg();
typedef void (*HSTIOErrHandler)();
int push_hstioerr(HSTIOErrHandler);
int pop_hstioerr();

/*
** High-level I/O Function Declarations
*/
int openFitsFile(char *filename, unsigned long option);
int closeFitsFile(char *filename);
int getSci(char *filename, int extver, SciHdrData *);
int putSci(char *filename, int extver, SciHdrData *, int option);
int getErr(char *filename, int extver, ErrHdrData *);
int putErr(char *filename, int extver, ErrHdrData *, int option);
int getDQ(char *filename, int extver, DQHdrData *);
int putDQ(char *filename, int extver, DQHdrData *, int option);
int getSmpl(char *filename, int extver, SmplHdrData *);
int putSmpl(char *filename, int extver, SmplHdrData *, int option);
int getIntg(char *filename, int extver, IntgHdrData *);
int putIntg(char *filename, int extver, IntgHdrData *, int option);
int getFloatHD(char *filename, char *extname, int extver, FloatHdrData *);
int putFloatHD(char *filename, char *extname, int extver, FloatHdrData *, int option);
int getShortHD(char *filename, char *extname, int extver, ShortHdrData *);
int putShortHD(char *filename, char *extname, int extver, ShortHdrData *, int option);
int getSingleGroup(char *filename, int extver, SingleGroup *);
int putSingleGroupHdr(char *filename, SingleGroup *, int option);
int putSingleGroup(char *filename, int extver, SingleGroup *, int option);
int getSingleNicmosGroup(char *filename, int extver, SingleNicmosGroup *);
int putSingleNicmosGroupHdr(char *filename, SingleNicmosGroup *, int option);
int putSingleNicmosGroup(char *filename, int extver, SingleNicmosGroup *,
        int option);
int getMultiGroupHdr(char *filename, MultiGroup *);
int getMultiGroup(MultiGroup *, int ngroup, int extver);
int putMultiGroupHdr(char *filename, MultiGroup *, int option);
int putMultiGroup(char *filename, int extver, MultiGroup *, int ngroup,
        int option);
int getMultiNicmosGroupHdr(char *filename, MultiNicmosGroup *);
int getMultiNicmosGroup(MultiNicmosGroup *, int ngroup, int extver);
int putMultiNicmosGroupHdr(char *filename, MultiNicmosGroup *, int option);
int putMultiNicmosGroup(char *filename, int extver, MultiNicmosGroup *,
        int ngroup, int option);

/*
```

```
** Functions for Accessing and Manipulating Keywords in a FITS Keyword List
*/

# if !defined(BOOL_)
# define BOOL_
enum Bool_ { False = 0, True = 1 };
typedef enum Bool_ Bool;
# endif
# define NotFound NULL
typedef void *FitsKw;
enum FitsDataType_ { FITSNOVALUE = 0, FITSLOGICAL, FITSBIT, FITSCHAR,
        FITSBYTE, FITSSHORT, FITSLONG, FITSFLOAT, FITSDOUBLE, FITSCOMPLEX,
        FITSICOMPLEX, FITSDCOMPLEX, FITSVADESC, FITSSTRING
};
typedef enum FitsDataType_ FitsDataType;

/*
** Making a FITS Header
**
** These return 0 is successful and -1 if an error occurred.
*/
int makePrimaryArrayHdr(Hdr *, FitsDataType, int dims, int *ndim);
int makeImageExtHdr(Hdr *, FitsDataType, int dims, int *ndim,
        char *extname, int extver);

/*
** Finding keywords in a FITS keyword list.
**
** The "find" function starts at the beginning of the list and finds the
** first occurence of the desired keyword.  The "findnext" function
** begins at the current position and returns the next occurence of the
** keyword.  NotFound is returned if no such keyword is found.  The "first"
** function sets the current position to the beginning of the list.
** The "next" function sets the position to the next keyword in the list.
** "Next" returns NotFound after the last keyword in the list.  "GetKw"
** returns the n-th keyword.  The special function "insertfirst" is used to
** position the cursor prior to the first item in the list so that a new
** keyword may be inserted at the beginning of the list.  (Insert functions
** always insert keywords at a position after the cursor.)
*/
FitsKw findKw(Hdr *, char *name);
FitsKw findnextKw(Hdr *, char *name);
FitsKw first(Hdr *);
FitsKw next(FitsKw);
FitsKw getKw(Hdr *, int n);
FitsKw insertfirst(Hdr *);

/*
** Accessing and changing name, value, or comment in a FITS keyword.
**
** These functions must be preceded by a "find" or "findnext" operation.
** Data conversions in the "get" routines:
**              FITS Type       Possible Requested Type
**              ---------       -----------------------
**              Logical         Bool, Int
**              Int             Int, Double
**              Float           Float, Double
**              Double          Double
**              String          String
** If the requested type conversion is not allowed, an error code is set.
*/
char *getKwName(FitsKw);
char *getKwComm(FitsKw);
FitsDataType getKwType(FitsKw);
Bool getBoolKw(FitsKw);
int  getIntKw(FitsKw);
float getFloatKw(FitsKw);
double getDoubleKw(FitsKw);
```

```
int getStringKw(FitsKw, char *str, int maxch);
int putKwName(FitsKw, char *name);
void putKwComm(FitsKw, char *comment);
int putBoolKw(FitsKw, Bool value);
int putIntKw(FitsKw, int value);
int putFloatKw(FitsKw, float value);
int putDoubleKw(FitsKw, double value);
int putStringKw(FitsKw, char *value);

/*
** Adding keywords to a FITS Header.
**
** The new keywords are added to the end of the keyword list.  There
** is no "END" keyword maintained in the list.  The software writing
** the FITS header automatically adds an "END" keyword before writing
** the header.
**
*/
int addBoolKw(Hdr *, char *name, Bool value, char *comment);
int addIntKw(Hdr *, char *name, int value, char *comment);
int addFloatKw(Hdr *, char *name, float value, char *comment);
int addDoubleKw(Hdr *, char *name, double value, char *comment);
int addStringKw(Hdr *, char *name, char * value, char *comment);
int addSpacesKw(Hdr *, char *comment);
int addCommentKw(Hdr *, char *comment);
int addHistoryKw(Hdr *, char *comment);

/*
** Inserting keywords in a FITS Header.
**
** The keywords are inserted after the current position.  The position
** is set with the "find", "first" and "next" functions.
**
*/
FitsKw insertBoolKw(FitsKw, char *name, Bool value, char *comment);
FitsKw insertIntKw(FitsKw, char *name, int value, char *comment);
FitsKw insertFloatKw(FitsKw, char *name, float value, char *comment);
FitsKw insertDoubleKw(FitsKw, char *name, double value, char *comment);
FitsKw insertStringKw(FitsKw, char *name, char * value, char *comment);
FitsKw insertSpacesKw(FitsKw, char *comment);
FitsKw insertCommentKw(FitsKw, char *comment);
FitsKw insertHistoryKw(FitsKw, char *comment);

/*
** Adding and inserting already formatted FITS cards to the list
**
** These functions take an already formatted FITS "card" and either adds
** the string to the list or inserts it after the current position.  If
** the string is less than 80 bytes it is padded with spaces to 80 bytes.
** If the string is more than 80 bytes it is truncated.
*/
int addFitsCard(Hdr *, char *card);
FitsKw insertFitsCard(FitsKw, char *card);

/*
** Deleting keywords in a FITS Header.
**
** The "del" function must be preceeded by a "find" or "findnext" operation.
** The "delAllKw" function deletes all keywords.
*/
void delKw(FitsKw);
void delAllKw(Hdr *);

/*
** Low-level I/O Function Declarations
*/
IODescPtr openInputImage(char *filename, char *extname, int extver);
IODescPtr openOutputImage(char *filename, char *extname, int extver, Hdr *hdr,
```

```
        int dim1, int dim2, FitsDataType type);
IODescPtr openUpdateImage(char *filename, char *extname, int extver, Hdr *hdr);
void closeImage(IODescPtr );

char *getFilename(IODescPtr);
char *getExtname(IODescPtr);
int getExtver(IODescPtr);
int getNaxis1(IODescPtr);
int getNaxis2(IODescPtr);
int getType(IODescPtr);

int getHeader(IODescPtr, Hdr *);
int putHeader(IODescPtr);

int getFloatData(IODescPtr, FloatTwoDArray *);
int putFloatData(IODescPtr, FloatTwoDArray *);
int getShortData(IODescPtr, ShortTwoDArray *);
int putShortData(IODescPtr, ShortTwoDArray *);

int getFloatLine(IODescPtr, int line, float *);
int putFloatLine(IODescPtr, int line, float *);
int getShortLine(IODescPtr, int line, short *);
int putShortLine(IODescPtr, int line, short *);

int getFloatSect(IODescPtr, int x_beg, int y_beg, int sx, int sy, FloatTwoDArray *);
int putFloatSect(IODescPtr, int x_beg, int y_beg, int sx, int sy, FloatTwoDArray *);
int getShortSect(IODescPtr, int x_beg, int y_beg, int sx, int sy, ShortTwoDArray *);
int putShortSect(IODescPtr, int x_beg, int y_beg, int sx, int sy, ShortTwoDArray *);

/*
** Initialization, Allocation, and Freeing Storage Function Declarations
*/
# define IFloatData { NULL, 0, 0, 0, 0, 0, NULL }
void initFloatData(FloatTwoDArray *);
int allocFloatData(FloatTwoDArray *, int, int);
void freeFloatData(FloatTwoDArray *);
# define IShortData { NULL, 0, 0, 0, 0, 0, NULL }
void initShortData(ShortTwoDArray *);
int allocShortData(ShortTwoDArray *, int, int);
void freeShortData(ShortTwoDArray *);
# define IHdr { 0, 0, NULL }
void initHdr(Hdr *);
int allocHdr(Hdr *, int);
int reallocHdr(Hdr *, int);
void freeHdr(Hdr *);
int copyHdr(Hdr *to, Hdr *from);

# define IFloatHdrData { NULL, { 0, 0, 0, 0 }, IHdr, IFloatData }
void initFloatHdrData(FloatHdrData *);
int allocFloatHdrData(FloatHdrData *, int, int);
void freeFloatHdrData(FloatHdrData *);
# define IShortHdrData { NULL, { 0, 0, 0, 0 }, IHdr, IShortData }
void initShortHdrData(ShortHdrData *);
int allocShortHdrData(ShortHdrData *, int, int);
void freeShortHdrData(ShortHdrData *);

# define ISingleGroup { NULL, 0, NULL, IFloatHdrData, IFloatHdrData, \
IShortHdrData }
void initSingleGroup(SingleGroup *);
int allocSingleGroup(SingleGroup *, int, int);
void freeSingleGroup(SingleGroup *);
# define IMultiGroup { 0, NULL }
void initMultiGroup(MultiGroup *);
int allocMultiGroup(MultiGroup *, int);
void freeMultiGroup(MultiGroup *);

# define ISingleNicmosGroup { NULL, 0, NULL, IFloatHdrData, IFloatHdrData, \
IShortHdrData, IShortHdrData, IFloatHdrData }
```

```
void initSingleNicmosGroup(SingleNicmosGroup *);
int allocSingleNicmosGroup(SingleNicmosGroup *, int, int);
void freeSingleNicmosGroup(SingleNicmosGroup *);
# define IMultiNicmosGroup { 0, NULL }
void initMultiNicmosGroup(MultiNicmosGroup *);
int allocMultiNicmosGroup(MultiNicmosGroup *, int);
void freeMultiNicmosGroup(MultiNicmosGroup *);

#if defined(__cplusplus)
}
# endif

# endif
```

2 Manipulating FITS Header Keywords

2.1 Making a FITS Header

```
int makePrimaryArrayHdr(Hdr *, FitsDataType, int dims, int *ndim);
int makeImageExtHdr(Hdr *, FitsDataType, int dims, int *ndim,
        char *extname, int extver);
```

These routines make header arrays with the minimal set of FITS keywords. They return 0 is successful and -1 if an error occurred.

2.2 Finding keywords in a FITS keyword list.

```
FitsKw findKw(Hdr *, char *name);
FitsKw findnextKw(Hdr *, char *name);
FitsKw first(Hdr *);
FitsKw next(FitsKw);
FitsKw getKw(Hdr *, int n);
FitsKw insertfirst(Hdr *);
```

The "find" function starts at the beginning of the list and finds the first occurence of the desired keyword. The "findnext" function begins at the current position and returns the next occurence of the keyword. NotFound is returned if no such keyword is found. The "first" function sets the current position to the beginning of the list. The "next" function sets the position to the next keyword in the list. "Next" returns NotFound after the last keyword in the list. "GetKw" returns the n-th keyword. The special function "insertfirst" is used to position the cursor prior to the first item in the list so that a new keyword may be inserted at the beginning of the list. (Insert functions always insert keywords at a position after the cursor.)

2.3 Accessing and changing name, value, or comment in a FITS keyword.

```
char *getKwName(FitsKw);
```

```
char *getKwComm(FitsKw);
FitsDataType getKwType(FitsKw);
Bool getBoolKw(FitsKw);
int  getIntKw(FitsKw);
float getFloatKw(FitsKw);
double getDoubleKw(FitsKw);
int getStringKw(FitsKw, char *str, int maxch);
int putKwName(FitsKw, char *name);
void putKwComm(FitsKw, char *comment);
int putBoolKw(FitsKw, Bool value);
int putIntKw(FitsKw, int value);
int putFloatKw(FitsKw, float value);
int putDoubleKw(FitsKw, double value);
int putStringKw(FitsKw, char *value);
```

These functions must be preceded by a “find” or “findnext” operation. Data conversions in the “get” routines:

FITS Type	Possible Requested Type
Logical	Bool, Int
Int	Int, Double
Float	Float, Double
Double	Double
String	String

If the requested type conversion is not allowed, an error code is set.

2.4 Adding keywords to a FITS Header.

```
int addBoolKw(Hdr *, char *name, Bool value, char *comment);
int addIntKw(Hdr *, char *name, int value, char *comment);
int addFloatKw(Hdr *, char *name, float value, char *comment);
int addDoubleKw(Hdr *, char *name, double value, char *comment);
int addStringKw(Hdr *, char *name, char * value, char *comment);
int addSpacesKw(Hdr *, char *comment);
int addCommentKw(Hdr *, char *comment);
int addHistoryKw(Hdr *, char *comment);
```

The new keywords are added to the keyword list just prior to the “END” keyword.

2.5 Inserting keywords in a FITS Header.

```
FitsKw insertBoolKw(FitsKw, char *name, Bool value, char *comment);
FitsKw insertIntKw(FitsKw, char *name, int value, char *comment);
FitsKw insertFloatKw(FitsKw, char *name, float value, char *comment);
FitsKw insertDoubleKw(FitsKw, char *name, double value, char *comment);
FitsKw insertStringKw(FitsKw, char *name, char * value, char *comment);
```

```
FitsKw insertSpacesKw(FitsKw, char *comment);
FitsKw insertCommentKw(FitsKw, char *comment);
FitsKw insertHistoryKw(FitsKw, char *comment);
```

The keywords are inserted after the current position. The position is set with the “find”, “first” and “next” functions.

2.6 Adding/Inserting already formatted keywords

```
int addFitsCard(Hdr *, char *card);
FitsKw insertFitsCard(FitsKw, char *card);
```

These functions take an already formatted FITS "card" and either adds the string to the list or inserts it after the current position. If the string is less than 80 bytes it is padded with spaces to 80 bytes. If the string is more than 80 bytes it is truncated.

2.7 Deleting keywords in a FITS Header.

```
void delKw(FitsKw);
void delAllKw(Hdr *);
```

The “del” function must be preceeded by a “find” or “findnext” operation. The “delAllKw” function deletes all keywords.

3 Error handling in hstio.c

I have built in some degree of flexibility into the error handling. The routines that return status information return 0 if successful and -1 for an error. You can access further error information via the functions:

```
HSTIOError hstio_err();
char *hstio_errmsg();
```

The first merely returns an enumerated code. The second returns a text string with the error message itself. These functions can be examined after any operation. “hstio_err()” will return 0 if there was no error.

In addition, you can install a global error handler. Suppose you have a function of the form:

```
void detect_error() {
        fprintf(stderr,"There was an ERROR!\n%s\n",
                hstio_errmsg());
        ... or whatever ...
        ... decide what to do ...
}
```

Then, you can install this function as a global error handler with the function:

```
push_hstioerr(detect_error);
```

There is a stack of function pointers (limited to a nesting depth of 32, which is arbitrary). If there is a non-zero pointer on the stack, that function is automatically called when an error occurs. You are free to do anything in the error handling function: print a message, abort, or whatever. Execution continues after executing the error handling function.

Of course, there is also a pop function:

```
pop_hstioerr();
```

So, you can manipulate the error handling stack as you see fit, e. g. by installing a new error handling function at the beginning of some complex procedure and popping it at the end. At any point, if you want to override the action of the global handler and give special handling after executing some function, you can merely push 0 onto the stack before executing the operation. That prevents the global error handler from being called. Of course, the pop function then restores the global error handler.

4 IRAF initialization

You don't have to call the c_irafinit routine anymore. The openXXImage routines do that. You still can call c_irafinit if you need to for other reasons.

5 OpenOutputImage

If the openOutputImage() routine is used to write a primary header, then the file designated by the filename must NOT already exist. If it does already exist, an error is returned. If the openOutputImage() routine is used to write an extension image, that HDU is appended to the end of the file.

6 OpenUpdateImage

If the openUpdateImage() routine is used the file designated by the filename must already exist. The HDU number associated with extname and extver is read and then re-written.

The openUpdateImage() routine automatically reads the header associated with the image. You do not have to do a separate getHeader().

7 Append versus Overwrite in high-level functions

In general, for all high-level I/O functions, if the option parameter is 0, whatever HDUs are involved are either written to a newly created file or appended to the end of an existing file. In other words, no existing data is destroyed. However, if the option parameter is Overwrite, whatever HDUs are involved overwrite HDU number extver.

```
For putSingleGroup:
    if option is 0, extver is ignored
        if the file does NOT exist
            the global header is written and
            the single group is written
        else if the file DOES exist
            the single group is appended
        endif
    else if option is Overwrite
        the file MUST exist
        the single group overwrites HDU number extver
    endif
```

```
For putSingleGroupHdr:
    if option is 0
        the file must NOT exist
        the primary HDU is written
    else if option is Overwrite
        the file MUST exist
        the primary HDU is overwritten
    endif
```

8 SingleGroup and MultiGroup Structures

The SingleGroup and MultiGroup structures have been changed. They are now:

```
typedef struct {
        char *filename;
        int group_num;
        Hdr *globalhdr;
        SciHdrData sci;
        ErrHdrData err;
        DQHdrData dq;
} SingleGroup;

typedef struct {
        char *filename;
        int group_num;
        Hdr *globalhdr;
        SciHdrData sci;
        ErrHdrData err;
        DQHdrData dq;
        SmplHdrData smpl;
        IntgHdrData intg;
} SingleNicmosGroup;
```

This is much more convenient than the previous definition because it makes a multi-group merely an array of single groups. However, this complicates the handling of the global header. Fortunately, this complication is completely buried in the I/O routines, at least if you use the normal init-alloc-free routines. Here's what those routines do. The SingleGroup case is fairly easy to handle. InitSingleGroup merely sets the pointer globalhdr to NULL, allocSingleGroup allocates it, and freeSingleGroup frees it. For a multigroup, group[0] contains the space for the actually allocated filename and the global header. The other groups in the multi-group merely point to the same allocated space. This means that there is only one global header and one filename, so it doesn't matter which group you use to change things in the global header. The freeMultiGroup routine is smart enough to free only group[0]. Neither input nor output have to be changed; everything should work just as before.

In the test examples, output has been copied from input. Handling output from scratch, i. e. generating it from completely new data, might take some explanation. Here's a code fragment as an example. Notice that the individual groups within a MultiGroup do not have to have the same dimensionality.

```
SingleGroup sg;
MultiGroup mg;

initSingleGroup(&sg);
initMultiGroup(&mg);

allocSingleGroup(&sg,128,128);
... fill the global header ...
... fill the data arrays ...
putSingleGroup("filename",1,&sg,0);
freeSingleGroup(&sg);

allocMultiGroup(&mg,3);
allocSingleGroup(&mg.group[0],128,128);
allocSingleGroup(&mg.group[1],256,256);
allocSingleGroup(&mg.group[2],512,512);

... fill the global header ...
... fill data group[0] ...
... fill data group[1] ...
```

```
... fill data group[2] ...

for (k = 0; k < 3; ++k)
    putMultiGroup("filename",(k + 1),&mg,k,0);
freeMultiGroup(&mg);
```

9 Example Programs

9.1 Reading an Image or Image groups

```
# include <hstio.h>
# include <stdio.h>

void showHdr(Hdr *h, char *title) {
    int i;
    printf("%s: number of lines: %d\n",title, h->nlines);
    if (h->array != NULL)
        for (i = 0; i < h->nlines; ++i)
            printf("%s\n",h->array[i]);
    printf("\n");
}

void showFloatData(FloatTwoDArray *x, char *title) {
    int i;
    printf("%s:\n",title);

    printf("(%d, 0) %g ",0,PPix(x,0,0));
     for (i = 1; i < 5; ++i) printf("%g ",PPix(x,i,0));
     printf(" ... ");
    printf("(%d, 0) %g ",(x->nx - 5),PPix(x,(x->nx - 5),0));
     for (i = x->nx - 4; i < x->nx; ++i) printf("%g ",PPix(x,i,0));
     printf("\n");

    printf("(%d, 1) %g ",0,PPix(x,0,1));
     for (i = 1; i < 5; ++i) printf("%g ",PPix(x,i,1));
     printf(" ... ");
    printf("(%d, 1) %g ",(x->nx - 5),PPix(x,(x->nx - 5),1));
     for (i = x->nx - 4; i < x->nx; ++i) printf("%g ",PPix(x,i,1));
     printf("\n");

    printf("(%d, max - 2) %g ",0,PPix(x,0,x->ny - 2));
     for (i = 1; i < 5; ++i) printf("%g ",PPix(x,i,x->ny - 2));
     printf(" ... ");
    printf("(%d, max - 2) %g ",(x->nx - 5),PPix(x,(x->nx - 5),x->ny - 2));
     for (i = x->nx - 4; i < x->nx; ++i) printf("%g ",PPix(x,i,x->ny - 2));
     printf("\n");

    printf("(%d, max - 1) %g ",0,PPix(x,0,x->ny - 1));
     for (i = 1; i < 5; ++i) printf("%g ",PPix(x,i,x->ny - 1));
     printf(" ... ");
    printf("(%d, max - 1) %g ",(x->nx - 5),PPix(x,(x->nx - 5),x->ny - 1));
     for (i = x->nx - 4; i < x->nx; ++i) printf("%g ",PPix(x,i,x->ny - 1));
     printf("\n");
    printf("\n");
}

void showShortData(ShortTwoDArray *x, char *title) {
    int i;
    printf("%s:\n",title);

    printf("(%d, 0) %d ",0,PPix(x,0,0));
```

```
		for (i = 1; i < 5; ++i) printf("%d ",PPix(x,i,0));
		printf(" ... ");
	printf("(%d, 0) %d ",(x->nx - 5),PPix(x,(x->nx - 5),0));
		for (i = x->nx - 4; i < x->nx; ++i) printf("%d ",PPix(x,i,0));
		printf("\n");

	printf("(%d, 1) %d ",0,PPix(x,0,1));
		for (i = 1; i < 5; ++i) printf("%d ",PPix(x,i,1));
		printf(" ... ");
	printf("(%d, 1) %d ",(x->nx - 5),PPix(x,(x->nx - 5),1));
		for (i = x->nx - 4; i < x->nx; ++i) printf("%d ",PPix(x,i,1));
		printf("\n");

	printf("(%d, max - 2) %d ",0,PPix(x,0,x->ny - 2));
		for (i = 1; i < 5; ++i) printf("%d ",PPix(x,i,x->ny - 2));
		printf(" ... ");
	printf("(%d, max - 2) %d ",(x->nx - 5),PPix(x,(x->nx - 5),x->ny - 2));
		for (i = x->nx - 4; i < x->nx; ++i) printf("%d ",PPix(x,i,x->ny - 2));
		printf("\n");

	printf("(%d, max - 1) %d ",0,PPix(x,0,x->ny - 1));
		for (i = 1; i < 5; ++i) printf("%d ",PPix(x,i,x->ny - 1));
		printf(" ... ");
	printf("(%d, max - 1) %d ",(x->nx - 5),PPix(x,(x->nx - 5),x->ny - 1));
		for (i = x->nx - 4; i < x->nx; ++i) printf("%d ",PPix(x,i,x->ny - 1));
		printf("\n");
	printf("\n");
}

void show_iodesc(IODescPtr x) {
	printf("IODesc: filename = %s extname = %s extver = %d\n\tnaxis1 = %d \
naxis2 = %d data type = %d\n", getFilename(x), getExtname(x), getExtver(x),
		getNaxis1(x), getNaxis2(x), getType(x));
}

void detect_err() {
	printf("HSTIO Error (%d): %s\n",hstio_err(),hstio_errmsg());
	exit(0);
}

int main(int argc, char **argv) {
	IODescPtr pri, sci, dq, err;
#ifdef EX3
	IODescPtr outpri, outsci, outdq, outerr;
#endif

	Hdr prihdr, scihdr, dqhdr, errhdr;
	FloatTwoDArray scidata;
	ShortTwoDArray dqdata;
	FloatTwoDArray errdata;
	int i;
	char *tmp;

	SciHdrData scihdrdata;
	DQHdrData  dqhdrdata;
	ErrHdrData errhdrdata;
	SingleGroup sg;
	MultiGroup mg;

	int A = 1; /* low-level routines */
	int B = 3; /* high-level XXHdrData routines */
	int C = 4; /* high-level SingleGroup */
	int D = 5; /* high-level MultiGroup */

	/*
	** Command line arguments:
	**	1. input file name
	**	2. output file name
```

```
	*/
#ifdef EX3
	if (argc != 3) {
		printf("Wrong number of input arguments: ex3 input output\n");
		exit(0);
	}
	printf("Executing program ex3 -- test of pipeline I/O routines\n");
	printf("Input: input file reading 6 groups of Sci, Err, DQ.\n");
	printf("Output: output file creating 6 groups of Sci, Err, DQ.\n");
#else
	if (argc != 2) {
		printf("Wrong number of input arguments: ex2 input\n");
		exit(0);
	}
	printf("Executing program ex2 -- test of pipeline I/O routines\n");
	printf("Input: input file reading 6 groups of Sci, Err, DQ.\n");
#endif

	/* install the error handler */
	push_hstioerr(detect_err);

	/*
	** Initialize the data structures.
	*/
	initFloatData(&scidata);
	initShortData(&dqdata);
	initFloatData(&errdata);
	initHdr(&prihdr);
	initHdr(&scihdr);
	initHdr(&dqhdr);
	initHdr(&errhdr);

	initFloatHdrData(&scihdrdata);
	initShortHdrData(&dqhdrdata);
	initFloatHdrData(&errhdrdata);
	initSingleGroup(&sg);
	initMultiGroup(&mg);

	/*
	** First, we will use the low-level I/O routines to open
	** and read the primary header, and the first three
	** extension image arrays.
	*/
	printf("\n======= Primary header: read =========\n");
	pri = openInputImage(argv[1],"",0);
	getHeader(pri,&prihdr);
	show_iodesc(pri);
	showHdr(&prihdr,"Primary Header Keywords");
	printf("======= Primary header: read complete =========\n");
#ifdef EX3
	outpri = openOutputImage(argv[2],"",0,&prihdr, 0, 0, 0);
	closeImage(outpri);
#endif
	closeImage(pri);
	freeHdr(&prihdr);

    for (i = A; i <= A + 1; ++i) {
	printf("\n======= Science data: read =========\n");
	sci = openInputImage(argv[1],"SCI",i);
	getHeader(sci,&scihdr);
	show_iodesc(sci);
	showHdr(&scihdr,"Science Header Keywords");
	getFloatData(sci,&scidata);
	showFloatData(&scidata,"Selected Science Data");
	printf("======= Science data: read complete =========\n");
#ifdef EX3
	outsci = openOutputImage(argv[2],"sci",i,&scihdr,
		getNaxis1(sci), getNaxis2(sci),0);
```

```
        putFloatData(outsci,&scidata);
        closeImage(outsci);
#endif
        closeImage(sci);
        freeFloatData(&scidata);
        freeHdr(&scihdr);

        printf("\n======= Error data: read =========\n");
        err = openInputImage(argv[1],"ERR",i);
        getHeader(err,&errhdr);
        showHdr(&errhdr,"Error Header Keywords");
        getFloatData(err,&errdata);
        show_iodesc(err);
        showFloatData(&errdata,"Selected Error Data");
        printf("======= Error data: read complete =========\n");
#ifdef EX3
        outerr = openOutputImage(argv[2],"err",i,&errhdr,
                errdata.nx, errdata.ny,0);
        putFloatData(outerr,&errdata);
        closeImage(outerr);
#endif
        closeImage(err);
        freeFloatData(&errdata);
        freeHdr(&errhdr);

        printf("\n======= Data Quality data: read =========\n");
        dq = openInputImage(argv[1],"DQ",i);
        getHeader(dq,&dqhdr);
        showHdr(&dqhdr,"Data Quality Header Keywords");
        getShortData(dq,&dqdata);
        show_iodesc(dq);
        showShortData(&dqdata,"Selected Data Quality Data");
        printf("\n======= Data Quality data: read complete =========\n");
#ifdef EX3
        outdq = openOutputImage(argv[2],"dq",i,&dqhdr,
                dqdata.nx, dqdata.ny,0);
        putShortData(outdq,&dqdata);
        closeImage(outdq);
#endif
        closeImage(dq);
        freeShortData(&dqdata);
        freeHdr(&dqhdr);

    }

        /*
        ** Now, we will test the high-level routines.
        ** First, test getSci, getErr, and getDQ.
        */
        printf("\n======= Science data: read =========\n");
        getSci(argv[1],B,&scihdrdata);
        show_iodesc(scihdrdata.iodesc);
        showHdr(&scihdrdata.hdr,"Science Header Keywords");
        showFloatData(&scihdrdata.data,"Science Data");
        printf("======= Science data: read complete =========\n");
#ifdef EX3
        putSci(argv[2],B,&scihdrdata,0);
#endif
        freeFloatHdrData(&scihdrdata);

        printf("\n======= Error data: read =========\n");
        getErr(argv[1],B,&errhdrdata);
        show_iodesc(errhdrdata.iodesc);
        showHdr(&errhdrdata.hdr,"Error Header Keywords");
        showFloatData(&errhdrdata.data,"Error Data");
        printf("======= Error data: read complete =========\n");
#ifdef EX3
        putErr(argv[2],B,&errhdrdata,0);
```

```
#endif
        freeFloatHdrData(&errhdrdata);

        printf("\n======= Data Quality data: read =========\n");
        getDQ(argv[1],B,&dqhdrdata);
        show_iodesc(dqhdrdata.iodesc);
        showHdr(&dqhdrdata.hdr,"DQ Header Keywords");
        showShortData(&dqhdrdata.data,"Data Quality Data");
        printf("======= Data Quality data: read complete =========\n");
#ifdef EX3
        putDQ(argv[2],B,&dqhdrdata,0);
#endif
        freeShortHdrData(&dqhdrdata);

        /*
        ** Now, test getSingleGroup.
        */
        printf("\n======= Single group data: read =========\n");
        getSingleGroup(argv[1],C,&sg);
        printf("Filename: %s group %d\n",sg.filename,sg.group_num);
        showHdr(sg.globalhdr,"Global header");
        showHdr(&sg.sci.hdr,"Science Header Keywords");
        showFloatData(&sg.sci.data,"Science Data");
        showHdr(&sg.err.hdr,"Error Header Keywords");
        showFloatData(&sg.err.data, "Error Data");
        showHdr(&sg.dq.hdr,"DQ Header Keywords");
        showShortData(&sg.dq.data, "Data Quality Data");
        printf("======= Single group data: read complete =========\n");
#ifdef EX3
        putSingleGroup(argv[2],C,&sg,0);
#endif
        freeSingleGroup(&sg);

        /*
        ** Finally, test getMultiGroup.
        */
        printf("\n======= Multiple group data: read =========\n");
        allocMultiGroup(&mg,4);
        getMultiGroupHdr(argv[1],&mg);
        for (i = 0; i < 2; ++i)
            { getMultiGroup(&mg,i,(i + D)); }
        printf("Filename: %s\n",mg.group[0].filename);
        showHdr(mg.group[0].globalhdr,"Global header");
        for (i = 0; i < 2; ++i) {
            showHdr(&mg.group[i].sci.hdr,"Science Header Keywords");
            showFloatData(&mg.group[i].sci.data,"Science Data");
            showHdr(&mg.group[i].err.hdr,"Error Header Keywords");
            showFloatData(&mg.group[i].err.data,"Error Data");
            showHdr(&mg.group[i].dq.hdr,"DQ Header Keywords");
            showShortData(&mg.group[i].dq.data,"Data Quality Data");
        }
        printf("======= Multi group data: read complete =========\n");
#ifdef EX3
        for (i = 0; i < 2; ++i)
            putMultiGroup(argv[2],(i + D),&mg,i,0);
#endif
        freeMultiGroup(&mg);

        return 0;
}
```

9.2 Illustration of Low-Level I/O Routines

```
# include <hstio.h>
# include <stdio.h>

/***********************************************************************

        Illustration of the low-level I/O routines
                        using
        Data Structures and I/O Function Declarations for
        STScI Calibration Pipeline Software for STIS and NICMOS

Input data structures, such as FloatTwoDArray and Hdr, each have
initilization, allocation, and free routines: initFloatData,
allocFloatData, freeFloatData.

On input, if the data structures already have storage allocated and that
storage is the same size as the requested data, that data storage is
reused.  If it is not the same size, the old storage is freed and new
storage is allocated.  If storage has not been allocated, then the input
routines allocate it.

It is the responsibility of the application program to free the
allocated storage.

***********************************************************************/

void detect_err() {
        printf("HSTIO Error (%d): %s\n",hstio_err(),hstio_errmsg());
        exit(0);
}

int main(int argc, char **argv) {
        IODescPtr in;
        FloatTwoDArray x = IFloatData;
        Hdr h = IHdr;
        IODescPtr out;
        FloatTwoDArray z = IFloatData;
        int i, j;

        /* install the error handler */
        push_hstioerr(detect_err);

        initFloatData(&x); /* Initialize the data structures.  Failure to */
        initFloatData(&z); /* initilize the data structures increases the */
        initHdr(&h);     /* probablilty of getting a core dump.         */

        /* This section illustrates the input case. */
        in = openInputImage("infile.fit","",0);
        getFloatData(in,&x);
        getHeader(in,&h);
        closeImage(in);
        for (j = x.ny - 1; j >= 0; --j) {       /* Do something with    */
            for (i = 0; i < x.nx; ++i)          /* the data,            */
                printf("%g ", Pix(x,i,j));
            printf("\n");
        }
        for (i = 0; i < h.nlines; ++i)          /* and the header.      */
                printf("%s\n", h.array[i]);
        freeFloatData(&x);                      /* The application must */
        freeHdr(&h);                            /* free storage.        */

        /* This section illustrates the output case. */
        allocFloatData(&z,5,5);                 /* Storage must be      */
        allocHdr(&h,25);                        /* allocated.           */
        for (j = z.ny - 1; j >= 0; --j)         /* Fill the data array  */
            for (i = 0; i < z.nx; ++i)
```

```
                Pix(z,i,j) = (i + 1) * 1000.0 + j + 1;
        /* And, also fill the header array */
        addFloatKw(&h,"CRVAL1",1.5," ");
        addFloatKw(&h,"CRVAL2",2.5," ");
        addFloatKw(&h,"CRPIX1",3.5," ");
        addFloatKw(&h,"CRPIX2",4.5," ");
        addFloatKw(&h,"CD1_1",5.5," ");
        addFloatKw(&h,"CD1_2",6.5," ");
        addFloatKw(&h,"CD2_1",7.5," ");
        addFloatKw(&h,"CD2_2",8.5," ");
        addIntKw(&h,"TEST1",34," ");
        addBoolKw(&h,"TEST2",True," ");

        out = openOutputImage("outfile1.fit","sci",0,&h,5,5,0);
        /* the openOutputImage routine writes the header */
        putFloatData(out,&z);                   /* write the data.      */
        closeImage(out);
        freeFloatData(&z);                      /* Then, free the       */
        freeHdr(&h);                            /* storage.             */

        /* This section illustrates copying data from one file to another */
        in = openInputImage("infile.fit","sci",0);
        getHeader(in,&h);
        getFloatData(in,&x);
        closeImage(in);
        out = openOutputImage("outfile2.fit","sci",0,&h,
                getNaxis1(in),getNaxis2(in),0);
        putFloatData(out,&x);
        closeImage(out);
        freeFloatData(&x);
        freeHdr(&h);

        return 0;
}
```