National Optical Astronomy Observatories

MEMORANDUM

TO: distribution **DATE:** May 7, 1985

FROM: Doug Tody

SUBJECT: New release of image i/o, etc. [NOTE: Good overview paper.]

The new release of IMIO has been installed in the system for a week now and appears to be bug free. This memo summarizes the changes/additions in this new version of the interface, and introduces the new "image database" tools *hedit* and *hselect* as well.

1. Summary of Changes in the Current Release

The following changes or additions have been made to the IMIO interface and the *images* package.

- IMIO now has the ability to perform (optionally) automatic boundary extension to satisfy out of bounds pixel references.
- A preliminary database interface has been added to IMIO.
- Image headers are now variable length and use only the amount of disk space required to store the header (excluding byte packing).
- Two new database utility tasks *hedit* and *hselect* have been added to the *images* package. Both use the new library subroutine *evexpr*, now installed in the FMTIO package.
- A new task *imshift* has been added to the *images* package to perform shifts of two dimensional images using full two dimensional interpolation. The related tasks *geomap* and *geotran* are currently being worked on. Some filtering and convolution tasks should also be available soon. All of these tasks use the new boundary extension feature of IMIO.

The new release of IMIO is upward compatible with previous versions and should require no changes to or even recompilation of existing code. The basic image header structure is unchanged hence existing images and image headers are still accessible. Copying of old images still on disk with *imcopy* may however be desirable to reduce disk consumption (the old headers were wasteful of storage).

This release of IMIO introduces some database tools and concepts which should help us understand the role the DBIO interface and DBMS package will play in image processing applications in the near future. The current database interface has serious functional limitations and is inefficient for operations which operate upon entire databases (e.g., the *select* operation), but does provide a basic and much needed image database capability.

2. Planned Future Developments

This new release of IMIO is expected to remain unchanged until DBIO is completed, at which time a new version of the interface will be released. This next release is expected to be upward compatible with the current interface except in cases where the applications task has detailed knowledge of the current image header structure. Applications which directly access the "user area" of the current header, or which use certain header fields such as IM_HISTORY, will have to be modified as these data structures will change in the next release.

Applications which use only *immap*, *imunmap*, IM_PIXTYPE, IM_NDIM, IM_LEN, and the basic i/o procedures should not have to be changed. The new interface will provide different facilities to do the same things but we can probably emulate the old interface to allow plenty of time to convert the old code. Of course, the new interface will provide new facilities which we did not formerly have and which we will want to use, and therefore we will eventually have to modify all existing image tasks.

Perhaps more seriously, we are not going to be able to maintain the ability to read the existing binary image files when the DBIO version of IMIO is released. At that time, all disk resident images will have to be processed to FITS format and thence into the new DBIO image format. We will keep the old binary for the FITS writer task around for an indefinite period after the changeover to make this possible.

3. Modifications to the Current Interface

3.1. Boundary extension

Automatic boundary extension is useful in applications such as filtering via convolution, since the convolution kernel will extend beyond the boundary of the image when near the boundary, and in applications which operate upon subrasters, for the same reason. When reading from an image with boundary extension in effect, IMIO will generate artificial values for the out of bounds pixels using one of the techniques listed below. When writing to an image with boundary extension in effect, the out of bounds pixels are discarded.

By default, an out of bounds pixel reference will result in an error message from IMIO. Consider an image line of length 5 pixels. The statement

```
buf = imgs1r (im, -1, 7)
```

references out of bounds by 2 pixels on either end of the image line, referencing a total of 5+2+2=9 pixels. If boundary extension is enabled and the get section completes successfully then *Memr[buf]* will reference the pixel at X = -1, and *Memr[buf+2]* will reference the first inbounds pixel.

When an image is first opened zero pixels of boundary extension are selected, and any out of bounds references will result in an error. To enable boundary extension *imseti* must be called on the open image to specify the number of pixels of boundary extension permitted before an out of bounds error occurs.

```
include <imset.h>
call imseti (im, IM_NBNDRYPIX, 2)
```

If boundary extension is enabled the type of boundary extension desired should also be set. The possibilities are defined in <*imset.h*> and are summarized below.

```
BT_CONSTANT     return constant if out of bounds
BT_NEAREST      return nearest boundary pixel
BT_REFLECT      reflect back into image
BT_WRAP         wrap around to other side
BT_PROJECT      project about boundary
```

Types of Boundary Extension

The type of boundary extension is set with the imset parameter IM_TYBNDRY. If the BT_CONSTANT option is selected the constant value should be set with an *imseti* or *imsetr* call to set the parameter IM_BNDRYPIXVAL. Boundary extension works for images of any dimension up to 7 (the current IMIO limit). A single IM_NBNDRYPIX value is used for all dimensions. This value is used only for bounds checking, hence the value should be set to the maximum out of bounds reference expected for any dimension. Larger values do not "cost more" than small ones. An actual out of bounds reference is however more expensive than an inbounds reference.

3.2. Image Database Interface

The image database interface is the IMIO interface to the database containing the image headers. In this implementation the image header is a variable length binary structure. The first, fixed format, part of the image header contains the standard fields in binary and is fixed in size. This is followed by the so called "user area", a string buffer containing a sequence of variable length, newline delimited FITS format keyword=value header cards. When an image is opened a large user area is allocated to permit the addition of new parameters without filling up the buffer. When the header is subsequently updated on disk only as much disk space is used as is needed to store the actual header.

The new header format is upwards compatible with the old image header format, hence old images and programs do not have to be modified to use the latest release of IMIO. In the future image headers will be maintained under DBIO, but the routines in the image header database interface described in this section are not exected to change. The actual disk format of images will of course change when we switch over to the DBIO headers. While the physical storage format of header will change completely under DBIO, the logical schema will change very little, i.e., our mental picture of an image header will be much as it is now. The main difference will be the consolidation of many images into a few files, and real support in the image header for bad pixels, history, and coordinate transformations. In addition a number of restrictions on the "user fields" will be lifted, the remaining distinctions between the standard and user fields will disappear, and database operations will be much more efficient than they are now.

3.2.1. Library Procedures

The IMIO library procedures comprising the current image database interface are summarized in the table below.

```
          value = imget[bcsilrd_] (im, field)
                          imgstr (im, field, outstr, maxch)
                  imput[bcsilrd_] (im, field, value)
                          impstr (im, field, value)
                  imadd[bcsilrd_] (im, field, def_value)
                          imastr (im, field, def_value)
                          imaddf (im, field, datatype)
                          imdelf (im, field)
                     y/n = imaccf (im, field)

                    list = imofnl[su] (im, template)
                nchars/EOF = imgnfn (list, fieldname, maxch)
                          imcfnl (list)

    where
          pointer    im, list
          char[]     field, outstr, datatype, template, fieldname
```

Image Database Interface Procedures

New parameters will typically be added to the image header with either one of the typed *imadd* procedures or with the lower level *imaddf* procedure. The former procedures permit the parameter to be created and the value initialized all in one call, while the latter only creates the parameter. In addition, the typed *imadd* procedures may be used to update the values of existing parameters, i.e., it is not considered an error if the parameter already exists. The principal limitation of the typed procedures is that they may only be used to add or set parameters of a standard datatype. The *imaddf* procedure will permit creation of parameters with more descriptive datatypes (abstract datatypes or domains) when the interface is recut upon DBIO. There is no support in the current interface for domains.

The value of any parameter may be fetched with one of the *imget* functions. *Be careful not to confuse* **imgets** *with* **imgstr** *(or* **imputs** *with* **impstr***) when fetching or storing the string value of a field*. Full automatic type conversion is provided. Any field may be read or written as a string, and the usual type conversions are permitted for the numeric datatypes.

The *imaccf* function may be used (like the FIO *access* procedure) to determine whether a field exists. Fields are deleted with *imdelf*; it is an error to attempt to delete a nonexistent field.

The field name list procedures *imofnl[su]*, *imgnfn*, and *imcfnl* procedures are similar to the familiar file template facilities, except that the @file notation is not supported. The template is expanded upon an image header rather than a directory. Unsorted lists are the most useful for image header fields. If sorting is enabled each comma delimited pattern in the template is sorted separately, rather than globally sorting the entire template after expansion. Minimum match is permitted when expanding the template, another difference from file templates. Only actual, full length field names are placed in the output list.

3.2.2. Standard Fields

The database interface may be used to access any field of the image header, including the following standard fields. Note that the nomenclature has been changed slightly to make it more consistent with FITS. Additional standard fields will be defined in the future. These names and their usage may change in the next release of IMIO.

keyword	*type*	*description*
i_ctime	l	time of image creation
i_history	s	history string buffer
i_limtime	l	time when limits (minmax) were last updated
i_maxpixval	r	maximum pixel value
i_minpixval	r	minimum pixel value
i_mtime	l	time of last modify
i_naxis	i	number of axes (dimensionality)
i_naxis[1-7]	l	length of an axis ("i_naxis1", etc.)
i_pixfile	s	pixel storage file
i_pixtype	i	pixel datatype (SPP integer code)
i_title	s	title string

Standard Header Fields

The names of the standard fields share an "i_" prefix to reduce the possibility of collisions with user field names, to identify the standard fields in sorted listings, to allow use of pattern matching to discriminate between the standard fields and user fields, and so on. For the convenience of the user, the "i_" prefix may be omitted provided the resultant name does not match the name of a user parameter. It is however recommended that the full name be used in all applications software.

3.2.3. Restrictions

The use of FITS format as the internal format for storing fields in this version of the interface places restrictions on the size of field names and of the string value of string valued parameters. Field names are currently limited to eight characters or less and case is ignored (since FITS requires upper case). The eight character limit does not apply to the standard fields. String values are limited to at most 68 characters. If put string is passed a longer string it will be silently truncated. Trailing whitespace and newlines are chopped when a string value is read.

4. Database Utility Tasks

Two image database utility tasks have been implemented, *hedit* and *hselect. Hedit* is the so called header editor, used to modify, add, or delete selected fields of selected images. The *hselect* task is used to select images that satisfy a selection criteria given as a boolean expression, printing a subset of the fields of these images on the standard output in list form. Manual pages are attached.

Both of these tasks gain most of their power from use of the *evexpr* utility procedure, now available in FMTIO. The *evexpr* procedure takes as input an algebraic expression (character string), parses and evaluates the expression, and returns as output the value of the expression.

```
        include <evexpr.h>
        pointer    evexpr()

        o = evexpr (expr, getop, ufcn)

    where
        o     Is a pointer to an operand structure
        expr  Is a character string
        getop Is either NULL or the locpr address
                of a user supplied procedure called during
                expression evaluation to get the value of
                an external operand.
        ufcn  Is either NULL or the locpr address
                of a user supplied procedure called during
                expression evaluation to satisfy a call to
                an external function.
```

The operand structure is defined in **<evexpr.h>**. The best documentation currently available for the operators and functions provided by *evexpr* will be found in the manual page(s) for *hedit*. Additional documentation will be found with the sources. The expression evaluation procedure is probably the single largest procedure in the system (in terms of kilobytes added to an executable) and should not be used unless it is needed, but it can greatly increase the power of a task in the right application.

Copy to: IRAF
Larry Goad
George Jacoby
Richard Wolff
Steve Ridgway (fyi)
Jeanette Barnes (fyi)
Ed Anderson (fyi)