Final Report for NAS5-32670

Incorporating the APS Catalog of the POSS I and Image Archive in ADS.

P.I. Roberta M. Humphreys

An Astrophysics Data Program, Type 2 Contract.

The primary purpose of this contract was to develop the software to both create and access an on-line database of images from digital scans of the Palomar Sky Survey.

This required modifying our DBMS (called Star Base) to create an image database from the actual raw pixel data from the scans.

The digitized images are processed into a set of coordinate-reference index and pixel files that are stored in run-length files, thus achieving an efficient lossless compression. For efficiency and ease of referencing, each digitized POSS I plate is then divided into 900 subplates. Our custom DBMS maps each query into the corresponding POSS plate(s) and subplate(s). All images from the appropriate subplates are retrieved from disk with byte-offsets taken from the index files. These are assembled on-the-fly into a GIF image file for browser display, and a FITS format image file for retrieval. The FITS images have a pixel size of 0.33 arcseconds. The FITS header contains astrometric and photometric information. This method keeps the disk requirements manageable while allowing for future improvements.

When complete, the APS Image Database will contain over 130 Gb of data. A set of web pages query forms are available on-line, as well as an on-line tutorial and documentation. The database is distributed to the Internet by a high-speed SGI server and a high-bandwidth disk system.

URL is http://aps.umn.edu/IDB/

The image database software is written in perl and C and has been compiled on SGI computers with IRIX5.3. A copy of the written documentation is included and the software is on the accompanying exabyte tape.

Publications

"An On-line Database of APS POSS Images," C.C. Cornuelle, G.S. Aldering, A. Sourov, P. Thurmes, and R.M. Humphreys, 1996, in Astronomical Data Analysis Software and Systems V, ASP Conference Series, 101, p. 501.

"The APS Catalog of the POSS I and Image Database," C.S. Cornuelle, G. Aldering, R.M. Humphreys, J. Larsen, and J. Cabanela, 1998, in New Horizons from MultiWavelength Sky Surveys, IAU Symp. 179, in press.

"The APS Catalog of the POSS I, Images Database, and Luyten Proper Motion Catalog," C.S. Cornuelle, G. Aldering, R.M. Humphreys, J. Larsen, and J. Cabanela, 1997, in Proper Motion and Galactic Astronomy, ASP Conference Series, 127, 55.

How Image Database Queries are Processed

This document is not intended for novice users of the image database. The readers should be the ones who are working on or will work on maintaining and/or extending the APS image database. Hence, some tedious details are omitted. The main medium to access the APS image database is through the World Wide Web. The online query form can be found at : ***http://spihr.spa.umn.edu/IDB/****. Documentations and tutorial on how to use the tool are there, too.*

1. A Brief Introduction :

After clicking the submit button, the input data are packed and sent back to a Perl script called **idb.def.pl**. This script analyses the data, then calls **query_www.pl** for further processing. **query_www.pl** is also a Perl script which does further calculation on the data, invokes **mosaic_idb** to do the actual query processing, and produce the image in FITS or GIF format. After that, query_www.pl returns the values needed for idb.def.pl to find the output image files to return to the browser. **idb.def.pl** also does the appropriate compression before sending the file back or displaying the information on how to obtain the image file. On the following sections, we shall discuss in more details how each of these modules works.

2. The Input Parameters :

As can be seen from the web page, the input parameters to the query form are :

- **RA, DEC** of the central coordinates,
- **Equinox,**
- **Field Width** (in arcminutes),
- **Plate Emulsion** (O - blue, or E - red),
- **Data Correction Mode** (Full correction, Quick correction, or Raw - meaning no correction is applied)
- **File Compression Method** (gzip, unix compress or non).

3. How query_www.pl works ?

query_www.pl is invoked using the following syntax :

query_www.pl {O|E} {Ra} {Dec} {Width in arcminutes} {Ancillary file name} [-Tnf]

where the last parameter (either -T, -n or -f) is optional. From the input emulsion (O or E), the central query box position (Ra and Dec), the ancillrary file name (added by Juan), and the query box width this script will generate the apropriate FITS image file(s). The script print out the FITS file names at the last step. The flags are described as follows.

- **-T** original data will be standard APS stars, transits, and densities in their usual directories; T stands for Transmision.
- **-f** fully-corrected pixel mapped mosaic output.

- **-n** totally uncorrected mosaic output - bloody awkful.

Notice that the coordinates must be in hh:mm:ss sexagesimal format. This program now runs on SGI machine : **spihr.spa.umn.edu**. The script source code is at **/aps/image_database/src/querying/query_www.pl**

Examples of usage are:

- query_www.pl O 14:22:25.00 -30:20:28.00 5
- query_www.pl O 12:00:03.00 -33:00:00.1 15

Before examining the code, there are some parameters to remember as follows.

- **$ftp_www = "/faster/ftp"** : the ftp root directory.
- **$cgi_dir = "/usr/local/etc/httpd/cgi-bin"** : where all the cgi scripts are.
- **$ftp_idb = "sb/IDB"** : where all the idb ftp files are.
- **$ftp_dir = "$ftp_www/$ftp_dir"**.

Here is a trace of program execution :

1. At first, **widthToMinMax($ra, $dec, $width)** is called and the result is returned to **@radecMinMax**, where ra, dec and width are the input parameters to query_www.pl

   * **widthToMinMax** :
   Input : ra, dec, and width of the field.
   Output : an array with 4 values
      1. "ra_min dec_min",
      2. "ra_min dec_max",
      3. "ra_max dec_min",
      4. "ra_max dec_max"

2. Next, **pss_list** is called on every line of @radecMinMax with -p option. Basically, pss_list -p returns informations about plates that match those ra's and dec's. The result is stored in **@pssResult** and looks like follows.

```
POSS      RA(1950)     DEC(1950)  E/O        EPOCH    LONG     LAT
914 14:22:25.000 -30:20:28.00 S 86 17 Apr 1958 325.82  28.10
```

3. @filResult = &pssFilter();

   * **pssFilter** :
   Input : @radecMinMax and @pssResult
   Output : Using the POSS numbers in @pssResult and the ra, dec in @radecMinMax, this function returns a list of strings in the following form, which contains the project, plate, ra, and dec of the 4 corners of the queried box

```
P266 O_109 11:59:27.22 32:52:30.100
P267 O_1599 11:59:27.22 32:52:30.100
```

4. Next, **idb_convert** is invoked to produce a list of {x, y} coordinates instead of the tuple {project, plate, ra, dec}. **@conResult** contains the output of idb_convert. It looks like :

```
906989.6 838369.5
53449.9 851757.0
```

5. Now, execute **convFilter**

   * **convFilter** :
   Input : @conResult and @filResult
   Output : nothing
   Side Effect : cut out all the error messages, comments, etc. in @conResult and @filResult. Keep only the data in there. Notice that the number of lines in @conResult and @filResult should be the same.

6. Concatenate each line of @filResult with the corresponding line of @conResult. After doing this, @filResult will contains lines of tuples {project, plate, ra, dec, x, y} like follows.

```
P266 O_109  11:59:27.22 32:52:30.100 906989.6 838369.5
P267 O_1599 11:59:27.22 32:52:30.100 53449.9  851757.0
```

7. @ptdResult = &plate_twiddle();

   * **plate_twiddle** :
   Input : @filResult with the above format
   Output : a list of tuples containing the following elements

```
{project, plate, xmin, xmax, ymin, ymax, plate_twiddle_list.$plate.$process
```

   These xmin, xmax, ymin, ymax values are on every plate that intersects the queried box. We will have to query individual plate and combine the resulting images later to produce the last resulting image. Example of @ptdResult is as follows. Notice that process# is the process id of the current query_www.pl script.

```
P266 O_109 905712.6 943622.6 800476.6 838369.5 plate_twiddle_list.O_109.135
P267 O_1599 53449.9 91243.3 815187.7 852980.0 plate_twiddle_list.O_1599.135
```

8. Now, this step is fairly important : get the data from @ptdResult, run pss_list again to get more information on plates and produces **@maData** which looks like follows. Note that the example is just one line of @maData.

```
P266 14 May 1950 11:44:58.000 35:28:20.00 906989.6 838369.5 P267 7 May 1956
```

   The data manipulation is sort of messy, I didn't have anymore interest in improving Chris' code (not a good excuse, but that's what happened). @maData is used by **make_ancillary** to produce the appropriate ancillary file with the name provided as a parameter to query_www.pl.

   * **make_ancillary** :
   Input : @maData
   Output : nothing

Side Effect : write out needed data to the ancillary file with the file path name provided as parameter to the script. Here is one example of how the file is like. Notice that the data in the file is used later by **mosaic_idb** to produce the FITS image and to write things into the FITS file's header. Many of these parameters are too specific to astronomy, hence I can't explain further (my major is Computer Science :-).

```
ANCILLARY ancfile
Mon Jan  5 15:46:43 CST 1998
.//query_www.pl O 12:00:03 33:00:00.1 15
906989.6 - 943622.6 = -36633
838369.5 - 838369.5 = 37892.9
53449.9 - 91243.3 = -37793.4
851757.0 - 852980.0 = 36569.3
PROJECT P266
PLATE O_109
EPOCH_D 14
EPOCH_M May
EPOCH_Y 1950
CNTR_RA 12:00:03
CNTR_DEC 33:00:00.1
X_CNTR 906989.6
Y_CNTR 838369.5
WIDTH_M 15
X_MIN 905712.6
X_MAX 943622.6
Y_MIN 800476.6
Y_MAX 838369.5
CRVAL1 176.24167
CRPIX1 -1117
CRVAL2 35.47222
CRPIX2 1155
PROJECT P267
PLATE O_1599
EPOCH_D 7
EPOCH_M May
EPOCH_Y 1956
CNTR_RA 12:00:03
CNTR_DEC 33:00:00.1
X_CNTR 53449.9
Y_CNTR 851757.0
WIDTH_M 15
X_MIN 53449.9
X_MAX 91243.3
Y_MIN 815187.7
Y_MAX 852980.0
CRVAL1 183.19583
CRPIX1 -1152
CRVAL2 35.47139
CRPIX2 1115
```

9. The last step is to call **mosaic_idb** to produce the FITS image files, whose execution is discussed in the last section. query_www.pl return the FITS file name(s) produced.

There are four more helper functions in query_www.pl, which are described briefly here.

* **printUsage** : print out to stdout the usage of query_www.pl

* **toAnc** : take a string as input, write the string to the ancillary file, whose handle was opened at the beginning of the script.

* **errSys** : print out the system error and quit

* **cleanUp** : close the file descriptors, clean things up

4. How mosaic_idb works ?

mosaic_idb is a grogram written in C with the task of producing the FITS image files given some parameters. The source code of mosaic_idb is at :

```
/aps/image_database/src/mosaic_idb
```

The command line syntax of **mosaic_idb** is as follows

```
mosaic_idb  project_name plate_name fits_name
                    ancillary_name XMIN XMAX YMIN YMAX
```

or

```
mosaic_idb  project_name plate_name fits_name
                    ancillary_name XC YC Rad
```

Notice that the first format has 8 non-option arguments and the second one has 7 non-option arguments. The semantics of **project_name, plate_name** should be clear. **fits_name** is the name of the output FITS image file. **ancillary_name** contains the full path name to access the ancillary file. **XMIN, XMAX, YMIN, YMAX** are the bounding coordinates of the queried box within that particular plate. For the second form, **XC, YC** are the central coordinates of the queried box, **Rad** is the half-width of the square box centered at **XC, YC**.

A good understanding of FITS format is crucial to understand the code. Important FITS documentations are listed as follows.

- FITSIO homepage : http://heasarc.gsfc.nasa.gov/docs/software/fitsio
- C FITSIO user guide : http://heasarc.gsfc.nasa.gov/docs/software/fitsio/user_c/user_c.html
- The FITSIO cook book : http://heasarc.gsfc.nasa.gov/docs/software/fitsio/cookbook/cookbook.html

I appology for my lack of a thorough knowledge of exactly how the images are constructed. There are several things that I don't understand such as : image correction, skew values, or Gaussian weight table for pixel remapping. However, from a programmer point of view, I DO know how the whole thing works, only some details are not as clear to me as the others.

The program uses a lot of global variables, some of them are parameters hard-coded into constant variables such as DIR, PROJECT, etc. Follows are the list of the most important global variables used by mosaic_idb. They are defined in **mosaic_idb.h**.

```
/* starting path for all project/plate information */
```

```
#define DIR "/i"

/* Input region information: */
long XMIN, XMAX, YMIN, YMAX;

/* Project/Plate information */
char PROJECT[32];
char PLATE[32];
int LEVEL = 1;  /* 1 is standard APS */

/* Flags */
int STRIPE_RELATIVE;          /* stripe-relative stars in the output */
int SHORT_OUTPUT;             /* short output                        */
int APPLY_CORRECTIONS;        /* apply corrections                   */
int FULL_CORRECTIONS;         /* Pixel remapping too.                */
int DIAGNOSTICS;              /* Diagnostic output to a file.        */
int MBACK_FLAG;               /* Deduct background value from pixel. */

/* Mosaic raster info */
int           NCOLS;           /* number of columns                          */
int           NROWS;           /* number of rows                             */
unsigned char *OUTPUT_BUFFER; /* the output buffer for the final raster     */
int           ICOLS;           /* number of columns                          */
int           IROWS;           /* number of rows                             */
unsigned char *IMAGE_BUFFER;  /* the output buffer for the image raster     */
int           RCOLS;           /* number of columns                          */
int           RROWS;           /* number of rows                             */
unsigned char *RASTER_BUFFER; /* the buffer for the corrected image raster  */
int           MIN_PIXEL;       /* Minimum pixel value for a given image.     */
int           MAX_PIXEL;       /* Max pixel value over all, less than MAXD   */
long          PREV_IM_BYTES;   /* The largest previous image buffer size.    */
long          PREV_RAS_BYTES;  /* The largest prev. corrected raster buf size */
int GWEIGHTS[MAX_GWEIGHTS];    /* Gaussian weights for corrections.          */
double SKEW_ARRAY[32768];      /* Indexed in eres, fill with skews.          */
```

And here are some frequently used data structures :

```
typedef unsigned char IDB_DN;
typedef short IDB_TR;

typedef struct {
  int starnum;  /* the plate-relative star number from the old S_REC */
  int dia;      /* star diameter                                     */
  int transits; /* offset for lseek into the transits file           */
  int dens;     /* offset into the densitometry file                 */
  long xmin, xmax, ymin, ymax; /* bounding box parameters            */
} IDB_SREC;

/* For mback subtraction information. */
struct mback_struct {
  int **rast;
  int nr, nc, x0, y0;
  double dx, dy;
} mback;

struct heady {
  int n_columns;
  int n_rows;
};

struct info {
```

```
  int in_xc;
  int in_yc;
  char in_dec[12];
  char in_ra[12];
  char project[4];
  char plate[6];
  int plate_ra_h;
  int plate_ra_m;
  int plate_ra_s;
  int plate_dec_d;
  int plate_dec_m;
  int plate_dec_s;
};

struct key {
  char word[S_WORD];
  char data[S_DATA];
  char text[S_DATA];
};

struct key aps_key[N_KEYS] = {
  {"CNTR_RA","\0","Query central Right Ascension (h:m:s)"},
  {"CNTR_DEC","\0","Query central declination (d:m:s)"},
  {"WIDTH_M","\0","Query box width in arcminutes"},
  {"PROJECT","\0","APS Project number"},
  {"PLATE","\0","POSS Plate label ID"},
  {"X_MIN","\0","Minimum box x-value (microns)"},
  {"X_MAX","\0","Maximum box x-value (microns)"},
  {"Y_MIN","\0","Minimum box y-value (microns)"},
  {"Y_MAX","\0","Maximum box y-value (microns)"},
  {"EPOCH_D","\0","Epoch day"},
  {"EPOCH_M","\0","Epoch month"},
  {"EPOCH_Y","\0","Epoch year"},
  {"X_CNTR","\0","Central plate x-value (microns)"},
  {"Y_CNTR","\0","Central plate y-value (microns)"},
  {"CRVAL1","\0","Plate central Right Ascension (degrees)"},
  {"CRPIX1","\0","Reference pixel i"},
  {"CRVAL2","\0","Plate central declination (degrees)"},
  {"CRPIX2","\0","Reference pixel j"}
};
```

The **main** function is defined in mosaic_idb.c, which basically calls three other functions in order described below.

1. **setup(argc, argv)** (in setup.c)

   The main task of this function is to parse the command line parameters and options (if any), then set the appropriate global flags and variables. More specifically, the flags to be set are : MBACK_FLAG, DIAGNOSTICS, FULL_CORRECTIONS, APPLY_CORRECTIONS, and SHORT_OUTPUT. The meaning of these flags has been specified above.

   The global variables get their values from the command line parameters are : PROJECT, PLATE, F_NAME (FITS file name), G_NAME (GIF file name), A_NAME (ancillary file name), XMIN, XMAX, YMIN, YMAX. Notice that if there are only 7 non-option parameters, XMIN, XMAX, YMIN and YMAX can be calculated from XC, YC and Rad parameters.

After that, **get_plate_info** (in setup.c) is called to grab all the necessary plate information, e.g the state vector, ytop struct, the spacing between scan lines (DENS_X_SCALE), spacing between pixels on the same scan line (DENX_Y_SCALE)

There are some library function calls that I don't really understand in depth and couldn't find any documentations about them :

- get_st_plt : to get state vector.
- get_yt_plt : to get ytop struct.
- get_ddx_plt : to get spacing between scan lines
- get_corr_plt : to apply corrections (?)

There is one important directory : **DIR/PROJECT/PLATE/sv** that contains the data files necessary for filling up some conversion tables. Those tables are D2T_ARRAY (Densitity to Transmission table), TI_lut (Transmission to Intensity lookup table).

When all above have been done, **setup_raster** is invoked to allocate space for the raster and set up the raster header. The values to be set up are : DENS_OUTPUT, NCOLS, NROWS, and OUTPUT_BUFFER. Then, the **skew** array and the Gaussian weights table are filled up.

2. **mosaic_idb**() (in mosaic_idb.c)

The main task this function does is to produce a list of all stars whose rectangular frames intersect the queried box region (rectangular). Recall that the region is defined by global variables XMIN, XMAX, YMIN, YMAX initialized earlier. In addition to that, mosaic_idb fills up the OUTPUT_BUFFER with the image data which is then used to write out to the GIF and FITS files. Follows are the steps this function perform to

1. Determine the bounding stripes and boxes by using **in_which_box** like follows :

```
olap = in_which_box(&box0, &stripe0, XMIN, YMIN, &ST);
```

then

```
in_which_box(&boxn, &stripen, XMAX, YMAX, &ST);
```

Thus, now we know the box number and the stripe number of the lower left corner (box0, stripe0 - associated with XMIN, YMIN), plus the box number and the stripe number of the upper right corner (boxn, stripen - associated with XMAX, YMAX).

2. For each of the stripe **snum** from stripe0 to stripen, do all of the following steps. Notice that all the **load_idb???_1stripe** functions take the stripe number as parameter, return the data read from appropriate files into idb_srec of type IDB_SREC described above.
3. Load the star data for the current stripe by calling **load_idbstars_1stripe** (in loader.c). This will fill up IDB_SREC.starnum, IDB_SREC.dia, IDB_SREC.(xmin, xmax, ymin, ymax) of the idb_srec array passed to the function. **starnum** is the the plate-relative starnum from the old S_REC. **dia** is the diameter of the star. **xmin, xmax, ymin, ymax** are the star bounding box corner coordinates. The data file is at DIR/PROJECT/PLATE/stars/lev**LEVEL**/str.**snum**, where LEVEL is a global variable.

4. Load the stransits data for the current stripe by calling **load_idbtransits_1stripe** (in loader.c). This will fill up IDB_SREC.transits of the idb_srec array passed to the function. **transits** is the offset for lseek into the transits file. The data file is at DIR/PROJECT/PLATE/transits/lev**LEVEL**/str.**snum** where LEVEL is a global variable.
5. Load the densitometry data for the current stripe by calling **load_idbdens_1stripe** (in loader.c). This will fill up IDB_SREC.dens of the idb_srec array passed to the function. **dens** is the offset for lseek into the densitometry file. The data file is at DIR/PROJECT/PLATE/dens/str.**snum**
6. If the MBACK_FLAG is set, call **get_mback** (in loader.c) to load the mback for the file.
7. Now, for each bnum from box0 to boxn, call **load_box()** (in loader.c) to load data in the box. load_box takes stripe number, box number and return (by side effect) a list of star numbers which lie inside the box. The data is read from DIR/PROJECT/PLATE/boxes/stripe.**snum**/box.**bnum** . Then, for each of the star in the box, check if it intersects the queried box. If so, call **paste_star** (in pastry.c) to add the star to the raster output and print out the star number.

3. **wrap_up()** (in mosaic_idb.c)

What this does is of the most interest to us. wrap_up calls four other functions as described in order below to read the ancillary file, to do initializations of the FITS manipulation plus preparing the FITS header, to write the entire FITS image file to disk and finally to produce the associated GIF file.

- **read_ancillary()** (in do_fits_beta.c) :

  Input : there is no parameters. The input is the A_NAME (ancillary file name).
  Output : an array of **struct key** (defined above) of length N_KEYS. A key is a tuple of 3 elements : *a keyword, a key value, and a string of comment.* These information are needed for filling out the FITS header.

  Basically, read_ancillary reads ancillary information from the file given on the command line. Notice that *the first 5 lines are skipped, all lines after 5 + N_KEYS inclusively are ignored* (currently, N_KEYS = 18). All the information we need lies on those lines of the ancillary file. The pointer to the key array is returned to **ancillary** in wrap_up() function. The key array is then passed to **have_fits** as described next.

- **have_fits(...)** (in do_fits_beta.c) :

  have_fits has the following prototypes :

```
void have_fits(fitsfile **make_fptr, struct key **anc_info,
               int *make_status)
```

  Notice that make_fptr is a pointer to the pointer to a fitsfile type. The file hasn't been opened yet, that is why we have to pass it this way. anc_info is the pointer to the value returned by read_ancillary (another pointer). I'm not sure why this has to be done this

way. I might change it to a single pointer later. This function does the following :

1. Initialize the FITS file for output using F_NAME provided at the command line parameter.
2. Write the FITS header using the data gotten from ancillary file. The header contains 2 parts : a basic part (containing things like University of Minnesota, department of Astronomy, ...) and another part containing more specific information on the image. Two functions, namely **fits_head_basics** and **fits_head_ancillary** (both in do_fits_beta.c) are called respectively to do the job.

The header look like follows :

```
SIMPLE  =                    T / file does conform to FITS standard
BITPIX  =                    8 / number of bits per data pixel
NAXIS   =                    2 / number of data axes
NAXIS1  =                  293 / length of data axis   1
NAXIS2  =                  293 / length of data axis   2
COMMENT   FITS (Flexible Image Transport System) format defined in Ast
COMMENT   Astrophysics Supplement Series v44/p363, v44/p371, v73/p359,
COMMENT   Contact the NASA Science Office of Standards and Technology
COMMENT   FITS Definition document #100 and other FITS information.
CTYPE1  = 'RA---TAN'           / X-axis type
CTYPE2  = 'DEC--TAN'           / Y-axis type
CDELT1  =          0.000092972 / Degrees per pixel
CDELT2  =          0.000092972 / Degrees per pixel
CROTA1  =               0.0000 / Rotation (degrees)
EQUINOX =                 1950 / Equinox of coordinates
RADECSYS= 'APS POSS I'         / Coordinate reference frame
SURVEY  = 'POSS I/APS scanned plates' /
BLANK   =                    0 / Value for empty/null/error pixel
COMMENT
DATE    = '30/12/95'           / FITS file creation date (dd/mm/yy)
COMMENT
COMMENT   This file was produced via the APS Image Database from scans
COMMENT   I survey plates. Pixels are a relative density value compute
COMMENT   original beam transmission data.
COMMENT
COMMENT   For more information please contact:
COMMENT
COMMENT     APS Project
COMMENT     Astronomy Department
COMMENT     University of Minnesota
COMMENT     116 Church Street SE
COMMENT     Minneapolis, MN 55455 USA
COMMENT     (612) 624-9069, FAX (612) 626-2029
COMMENT     aps@aps.umn.edu
COMMENT     http://aps.umn.edu
COMMENT
COMMENT   The APS Image Database is supported by NASA ADP grant NAS5-3
COMMENT
CNTR_RA = '13:00:00'           / Query central Right Ascension (h:m:s)
CNTR_DEC= '30:00:00'           / Query central declination (d:m:s)
WIDTH_M = '4       '           / Query box width in arcminutes
PROJECT = 'P323    '           / APS Project number
PLATE   = 'E_1393  '           / POSS Plate label ID
X_MIN   =         118551.86719 / Minimum box x-value (microns)
X_MAX   =         122167.61719 / Maximum box x-value (microns)
Y_MIN   =         142289.53125 / Minimum box y-value (microns)
Y_MAX   =         145905.01562 / Maximum box y-value (microns)
```

```
EPOCH_D =                    15 / Epoch day
EPOCH_M = 'Apr      '           / Epoch month
EPOCH_Y =                  1955 / Epoch year
X_CNTR  =         173047.73438 / Central plate x-value (microns)
Y_CNTR  =         171809.84375 / Central plate y-value (microns)
CRVAL1  =            196.13750 / Plate central Right Ascension (degree
CRPIX1  =                 4240 / Reference pixel i
CRVAL2  =             29.49028 / Plate central declination (degrees)
CRPIX2  =                    0 / Reference pixel j
END
```

- **write_fits(...)** (in do_fits_beta.c) :

  As the name suggests, write_fits actually writes the image data to the FITS file. The data has been created and put into **OUTPUT_BUFFER** by calling function mosaic_idb described above. There is nothing more to say about write_fits. But there are some frequently used CFITSIO functions which are used in this program that I would like to introduce here. More information can be found at the FITS homepage. It is important to know some descriptions of useful FITS functions as follows.

  1. **ffopen** : Open an existing FITS file with readonly or readwrite access. The iomode parameter has allowed symbolic constant values of READONLY or READWRITE. If the filename = "-" then CFITSIO will read the FITS file from the stdin file stream rather than from a disk file. If the file to be opened resides in memory then a null file name may be given to indicate that the file already exists in memory; otherwise the specified diskfile (or the stdin stream if the file name = "-") will be copied into memory and all subsequent operations on the file will take place on the memory copy of the file. Note that any modifications to the memory file do not automatically get copied back to the disk file in this case.

     ```
     int fits_open_file / ffopen
      (fitsfile **fptr, char *filename, int iomode, > int *status)
     ```

  2. **ffinit** : Create and initialize a new empty FITS file. If the filename = "-" then the file will be written to the stdout file stream rather than to magnetic disk. (The file is actually created in memory and flushed to the stdout stream when the FITS file is closed). If a memory buffer has been allocated for the file then the file name is ignored unless it is '-' in which case the memroy file is copied to the stdout stream when the file is closed.

     ```
     int fits_create_file / ffinit
         (fitsfile **fptr, char *filename, > int *status)
     ```

  3. **ffgerr** : Return a descriptive text string corresponding to a CFITSIO error status code. The 30-character length string contains a brief description of the cause of the error.

     ```
     void fits_get_errstatus / ffgerr (int status, > char *err_text)
     ```

  4. **ffclos** : Close a previously opened FITS file. If the file resides in a user-allocated memory buffer (see fits_set_mem_buff, below) then the memory buffer is left unchanged and the application program must free the memory when it is no

longer needed.

```
int fits_close_file / ffclos
    (fitsfile *fptr, > int *status)
```

5. **ffphp?**: Put the primary header or IMAGE extension keywords into the CHU. The simpler ffphps routine is equivalent to calling ffphpr with the default values of simple = TRUE, pcount = 0, gcount = 1, and extend = TRUE. The PCOUNT, GCOUNT and EXTEND keywords are not required in the primary header and are only written if pcount is not equal to zero, gcount is not equal to zero or one, and if extend is TRUE, respectively. When writing to an IMAGE extension, the SIMPLE and EXTEND parameters are ignored. Refer to Chapter 9 of CFITIOS User's Guide for a list of pre-defined bitpix values.

```
int fits_write_imghdr / ffphps
    (fitsfile *fptr, int bitpix, int naxis, long *naxes, > int

int fits_write_grphdr / ffphpr
    (fitsfile *fptr, int simple, int bitpix, int naxis, long *n
     long pcount, long gcount, int extend, > int *status)
```

6. **ffpky?** : Put (append) a new keyword of the appropriate datatype into the CHU. There is a separate routine for each datatype. Note that ffpkys will only write string values up to 68 characters in length and longer strings will be truncated. The ffpkls routine can be used to write longer strings, using the non-standard FITS convention that was described in an earlier section.

```
int fits_write_key_str / ffpkys
    (fitsfile *fptr, char *keyname, char *value, char *comment,
     > int *status)

int fits_write_key_[log, lng] /  ffpky[lj]
    (fitsfile *fptr, char *keyname, DTYPE numval, char *comment
     > int *status)

int fits_write_key_[flt, dbl, fixflg, fixdbl] / ffpky[edfg]
    (fitsfile *fptr, char *keyname, DTYPE numval, int decimals,
     char *comment, > int *status)
```

7. **ffpcom** : Put (append) a COMMENT keyword into the CHU. The comment string will be split over multiple COMMENT keywords if it is longer than 70 characters.

```
int fits_write_comment / ffpcom
    (fitsfile *fptr, char *comment, > int *status)
```

8. **ffpnul** : Define the integer value to be used to signify undefined pixels in the primary array or image extension. This is only used if BITPIX = 8, 16, or 32. This does not create or change the value of the BLANK keyword in the header.

```
int fits_set_imgnul / ffpnul
    (fitsfile *fptr, long nulval, > int *status)
```

9. **ffpprb** : Put elements into the data array. The datatype is specified by the suffix

of the name of the routine.

```
int fits_write_img_[byt, sht, usht, lng, ulng, int, flt, dbl] /
    ffppr[b,i,ui,j,uj,k,e,d]
    (fitsfile *fptr, long group, long firstelem, long nelements
     DTYPE *array, > int *status);
```

- **make_gif(...)** (in do_gifs.c) : This function handles interaction with the netpbm routines that generate a GIF file, namely GIFEncode and the pre-compiled package. The main routines to use here are from the gifdraw package. The data is from OUTPUT_BUFFER as in the case of FITS file. Since creating GIF files is not our main concern now, so I'm going to delay describing this function in details.

  More information about GIF can be found at :
  - GIF draw library : http://www.boutell.com/gd/
  - GIF lib library : http://locke.ccil.org/~esr/giflib/

---

by **Hung Q. Ngo**
hngo@cs.umn.edu