ANALYZE™ 7.5 File Format

The image database is the system of files that the ANALYZE™ package uses to organize and access image data on the disk. Facilities are provided for converting data from a number of sources for use with the package. A description of the database format is provided to aid developers in porting images from other sources for use with the ANALYZE™ system. An ANALYZE™ image database consists of at least two files:

- an image file
- a header file

The files have the same name being distinguished by the extensions .img for the image file and .hdr for the header file. Thus, for the image database heart, there are the UNIX files heart.img and heart.hdr. The ANALYZE™ programs all refer to this pair of files as a single entity named heart.

Image File

The format of the image file is very simple containing usually uncompressed pixel data for the images in one of several possible pixel formats:

Header File

The header file is represented here as a `C' structure which describes the dimensions and history of the pixel data. The header structure consists of three substructures:

- header_key describes the header
- image_dimension describes image sizes
- data_history optional

```
/*    ANALYZE™ Header File Format
*
*  (c) Copyright, 1986-1995
*  Biomedical Imaging Resource
*  Mayo Foundation
*
*  dbh.h
*
*  databse sub-definitions
*/

struct header_key               /* header key      */
     {                          /* off + size      */
     int sizeof_hdr             /* 0 + 4           */
     char data_type[10];        /* 4 + 10          */
     char db_name[18];          /* 14 + 18          */
     int extents;               /* 32 + 4          */
     short int session_error;   /* 36 + 2           */
     char regular;              /* 38 + 1           */
     char hkey_un0;                 /* 39 + 1       */
  };                        /* total=40 bytes  */
```

```
struct image_dimension
    {                                   /* off + size       */
    short int dim[8];                   /* 0 + 16           */
    short int unused8;                  /* 16 + 2           */
    short int unused9;                  /* 18 + 2           */
    short int unused10;                 /* 20 + 2           */
    short int unused11;                 /* 22 + 2           */
    short int unused12;                 /* 24 + 2           */
    short int unused13;                 /* 26 + 2           */
    short int unused14;                 /* 28 + 2           */
    short int datatype;                 /* 30 + 2           */
    short int bitpix;                   /* 32 + 2           */
    short int dim_un0;                  /* 34 + 2           */
    float pixdim[8];                    /* 36 + 32          */
                /*
                    pixdim[] specifies the voxel dimensitons:
                    pixdim[1] - voxel width
                    pixdim[2] - voxel height
                    pixdim[3] - interslice distance
                        ...etc
                */
    float vox_offset;                   /* 68 + 4                   */
    float funused1;                     /* 72 + 4                   */
    float funused2;                     /* 76 + 4                   */
    float funused3;                     /* 80 + 4                   */
    float cal_max;                      /* 84 + 4                   */
    float cal_min;                      /* 88 + 4                   */
    float compressed;                   /* 92 + 4                   */
    float verified;                     /* 96 + 4                   */
    int glmax,glmin;                    /* 100 + 8                  */
    };                                  /* total=108 bytes          */

struct data_history
    {                                   /* off + size       */
    char descrip[80];                   /* 0 + 80           */
    char aux_file[24];                  /* 80 + 24          */
    char orient;                        /* 104 + 1          */
    char originator[10];                /* 105 + 10         */
    char generated[10];                 /* 115 + 10         */
    char scannum[10];                   /* 125 + 10         */
    char patient_id[10];                /* 135 + 10         */
    char exp_date[10];                  /* 145 + 10         */
    char exp_time[10];                  /* 155 + 10         */
    char hist_un0[3];                   /* 165 + 3          */
    int views                           /* 168 + 4          */
    int vols_added;                     /* 172 + 4          */
    int start_field;                    /* 176 + 4          */
    int field_skip;                     /* 180 + 4          */
    int omax, omin;                     /* 184 + 8          */
    int smax, smin;                     /* 192 + 8          */
    };
```

```
struct dsr
     {
     struct header_key hk;                /* 0 + 40                */
     struct image_dimension dime;         /* 40 + 108              */
     struct data_history hist;            /* 148 + 200             */
     };                                   /* total= 348 bytes      */

/* Acceptable values for datatype */

#define DT_NONE                          0
#define DT_UNKNOWN                       0
#define DT_BINARY                        1
#define DT_UNSIGNED_CHAR                 2
#define DT_SIGNED_SHORT                  4
#define DT_SIGNED_INT                    8
#define DT_FLOAT                         16
#define DT_COMPLEX                       32
#define DT_DOUBLE                        64
#define DT_RGB                           128
#define DT_ALL                           255

typedef struct
     {
      float real;
      float imag;
     } COMPLEX;
```

Comments

The header format is flexible and can be extended for new user-defined data types. The essential structures of the header are the header_key and the image_dimension. The required elements in the header_key substructure are:

- int sizeof_header Must indicate the byte size of the header file.
- int extents Should be 16384, the image file is created as contiguous with a minimum extent size.
- char regular Must be `r' to indicate that all images and volumes are the same size.

The image_dimension substructure describes the organization and size of the images. These elements enable the database to reference images by volume and slice number. Explanation of each element follows:

- short int dim[]; array of the image dimensions
- dim[0] Number of dimensions in database; usually 4
- dim[1] Image X dimension; number of pixels in an image row
- dim[2] Image Y dimension; number of pixel rows in slice
- dim[3] Volume Z dimension; number of slices in a volume
- dim[4] Time points, number of volumes in database.
- char vox_units[4] specifies the spatial units of measure for a voxel
- char cal_units[4] specifies the name of the calibration unit
- short int datatype datatype for this image set

  Acceptable values for datatype are

  #define DT_NONE 0

```
#define DT_UNKNOWN          0    Unknown data type
#define DT_BINARY           1    Binary (1 bit per voxel)
#define DT_UNSIGNED_CHAR    2    Unsigned character (8 bits per voxel)
#define DT_SIGNED_SHORT     4    Signed short (16 bits per voxel)
#define DT_SIGNED_INT       8    Signed integer (32 bits per voxel)
#define DT_FLOAT            16   Floating point (32 bits per voxel)
#define DT_COMPLEX          32   Complex (64 bits per voxel)
#define DT_DOUBLE           64   Double precision (64 bits per voxel)
#define DT_RGB              128
#define DT_ALL              255
```

- short int bitpix; number of bits per pixel; 1, 8, 16, 32, or 64.
- short int dim_un0; unused
- float pixdim[]; Parallel array to dim[], giving real world measurements in mm. and ms.
- pixdim[1]; voxel width in mm.
- pixdim[2]; voxel height in mm.
- pixdim[3]; slice thickness in mm.
- float vox_offset; byte offset in the .img file at which voxels start. This value can negative to specify that the absolute value is applied for every image in the file.
- float calibrated Max, Min specify the range of calibration values
- int glmax, glmin; The maximum and minimum pixel values for the entire database.

The data_history substructure is not required, but the orient field is used to indicate individual slice orientation and determines whether the Movie program will attempt to flip the images before displaying a movie sequence.

- orient: slice orientation for this dataset.
  - 0 transverse unflipped
  - 1 coronal unflipped
  - 2 sagittal unflipped
  - 3 transverse flipped
  - 4 coronal flipped
  - 5 sagittal flipped

Sample Program

Any image data can be ported to the ANALYZE™ system by creating the appropriate image and header files. Although header files can be created with the Header Edit program, the following C program is provided to illustrate how to make an ANALYZE™ image database header file given the critical image dimensions as parameters. For example;

make_header heart.hdr 128 128 97 3 CHAR 255 0

Makes the header file heart.hdr with the following dimensions:

x dimension = 128
y dimension = 128
slices/volume = 97
volumes in file = 3
bits/pixel = 8
global max = 255
global min = 0

```
/* This program creates an ANALYZE™ database header  */
/*
* (c) Copyright, 1986-1995
* Biomedical Imaging Resource
* Mayo Foundation
*
* to compile:
*
*    cc -o make_hdr make_hdr.c
*/
#include <stdio.h>
#include "dbh.h"

main(argc,argv) /* file x y z t datatype max min */
int argc;
char **argv;
{
  int i;
  struct dsr hdr;
  FILE *fp;
  static char DataTypes[9][12] = {"UNKNOWN", "BINARY",
       "CHAR", "SHORT", "INT","FLOAT", "COMPLEX",
       "DOUBLE", "RGB"};

  static int DataTypeSizes[9] = {0,1,8,16,32,32,64,64,24};

  if(argc != 9)
  {
                usage();
    exit(0);
  }
  memset(&hdr,0, sizeof(struct dsr));
  for(i=0;i<8;i++)
          hdr.dime.pixdim[i] = 0.0;

  hdr.dime.vox_offset  = 0.0;
  hdr.dime.funused1    = 0.0;
  hdr.dime.funused2    = 0.0;
  hdr.dime.funused3    = 0.0;
  hdr.dime.cal_max     = 0.0;
  hdr.dime.cal_min     = 0.0;


  hdr.dime.datatype = -1;

  for(i=1;i<=8;i++)
          if(!strcmp(argv[6],DataTypes[i]))
          {
                hdr.dime.datatype = (1<<(i-1));
                hdr.dime.bitpix = DataTypeSizes[i];
                break;
          }
```

```
  if(hdr.dime.datatype <= 0)
  {
          printf("<%s> is an unacceptable datatype \n\n", argv[6]);
          usage();
     exit(0);
  }

  if((fp=fopen(argv[1],"w"))==0)
  {
     printf("unable to create: %s\n",argv[1]);
     exit(0);
  }

  hdr.dime.dim[0] = 4;  /* all Analyze images are taken as 4 dimensional */
  hdr.hk.regular = 'r';
  hdr.hk.sizeof_hdr = sizeof(struct dsr);

  hdr.dime.dim[1] = atoi(argv[2]);  /* slice width  in pixels */
  hdr.dime.dim[2] = atoi(argv[3]);  /* slice height in pixels */
  hdr.dime.dim[3] = atoi(argv[4]);  /* volume depth in slices */
  hdr.dime.dim[4] = atoi(argv[5]);  /* number of volumes per file */

  hdr.dime.glmax  = atoi(argv[7]);  /* maximum voxel value  */
  hdr.dime.glmin  = atoi(argv[8]);  /* minimum voxel value */

/*     Set the voxel dimension fields:
      A value of 0.0 for these fields implies that the value is unknown.
        Change these values to what is appropriate for your data
        or pass additional command line arguments     */

  hdr.dime.pixdim[1] = 0.0; /* voxel x dimension */
  hdr.dime.pixdim[2] = 0.0; /* voxel y dimension */
  hdr.dime.pixdim[3] = 0.0; /* pixel z dimension, slice thickness */

/*   Assume zero offset in .img file, byte at which pixel
     data starts in the image file */

  hdr.dime.vox_offset = 0.0;

/*   Planar Orientation;    */
/*   Movie flag OFF: 0 = transverse, 1 = coronal, 2 = sagittal
    Movie flag ON:  3 = transverse, 4 = coronal, 5 = sagittal  */

  hdr.hist.orient     = 0;

/*   up to 3 characters for the voxels units label; i.e. mm., um., cm. */              */

  strcpy(hdr.dime.vox_units," ");

/*   up to 7 characters for the calibration units label; i.e. HU */

  strcpy(hdr.dime.cal_units," ");
```

```
/*     Calibration maximum and minimum values;
       values of 0.0 for both fields imply that no
       calibration max and min values are used    */

   hdr.dime.cal_max = 0.0;
   hdr.dime.cal_min = 0.0;

   fwrite(&hdr,sizeof(struct dsr),1,fp);
   fclose(fp);
}

usage()
{
  printf("usage:  make_hdr name.hdr x y z t datatype max min \n\n");
  printf("  name.hdr = the name of the header file\n");
  printf("  x = width, y = height,  z = depth,  t = number of volumes\n");
  printf("  acceptable datatype values are: BINARY, CHAR, SHORT,\n");
  printf("                INT, FLOAT, COMPLEX, DOUBLE, and RGB\n");
  printf("  max = maximum voxel value,  min = minimum voxel value\n");
}
```

The following program displays information in an AnalyzeTM header file.

```
#include <stdio.h>
#include "dbh.h"

void ShowHdr(char *, struct dsr *);
void swap_long(unsigned char *);
void swap_short(unsigned char *);

main(argc,argv)
int argc;
char **argv;
  {
  struct dsr hdr;
  int size;
  double cmax, cmin;
  FILE *fp;

    if((fp=fopen(argv[1],"r"))==NULL)
  {
     fprintf(stderr,"Can't open:<%s>\n", argv[1]);
     exit(0);
  }
  fread(&hdr,1,sizeof(struct dsr),fp);

    if(hdr.dime.dim[0] < 0 || hdr.dime.dim[0] > 15)
          swap_hdr(&hdr);

   ShowHdr(argv[1], &hdr);
   }

void ShowHdr(fileName,hdr)
struct dsr *hdr;
char *fileName;
{
int i;
char string[128];
printf("Analyze Header Dump of: <%s> \n", fileName);
/* Header Key */
printf("sizeof_hdr: <%d> \n", hdr->hk.sizeof_hdr);
printf("data_type:  <%s> \n", hdr->hk.data_type);
printf("db_name:    <%s> \n", hdr->hk.db_name);
printf("extents:    <%d> \n", hdr->hk.extents);
printf("session_error: <%d> \n", hdr->hk.session_error);
printf("regular:  <%c> \n", hdr->hk.regular);
printf("hkey_un0: <%c> \n", hdr->hk.hkey_un0);

/* Image Dimension */
for(i=0;i<8;i++)
     printf("dim[%d]: <%d> \n", i, hdr->dime.dim[i]);

     strncpy(string,hdr->dime.vox_units,4);
     printf("vox_units:  <%s> \n", string);
```

```
    strncpy(string,hdr->dime.cal_units,8);
    printf("cal_units: <%s> \n", string);
    printf("unused1:   <%d> \n", hdr->dime.unused1);
    printf("datatype:  <%d> \n", hdr->dime.datatype);
    printf("bitpix:    <%d> \n", hdr->dime.bitpix);

for(i=0;i<8;i++)
    printf("pixdim[%d]: <%6.4f> \n",i, hdr->dime.pixdim[i]);

printf("vox_offset: <%6.4> \n",  hdr->dime.vox_offset);
printf("funused1:   <%6.4f> \n", hdr->dime.funused1);
printf("funused2:   <%6.4f> \n", hdr->dime.funused2);
printf("funused3:   <%6.4f> \n", hdr->dime.funused3);
printf("cal_max:    <%6.4f> \n", hdr->dime.cal_max);
printf("cal_min:    <%6.4f> \n", hdr->dime.cal_min);
printf("compressed: <%d> \n", hdr->dime.compressed);
printf("verified:   <%d> \n", hdr->dime.verified);
printf("glmax:      <%d> \n", hdr->dime.glmax);
printf("glmin:      <%d> \n", hdr->dime.glmin);

/* Data History */
strncpy(string,hdr->hist.descrip,80);
printf("descrip:  <%s> \n", string);
strncpy(string,hdr->hist.aux_file,24);
printf("aux_file: <%s> \n", string);
printf("orient:   <%d> \n", hdr->hist.orient);

strncpy(string,hdr->hist.originator,10);
printf("originator: <%s> \n", string);

strncpy(string,hdr->hist.generated,10);
printf("generated: <%s> \n", string);


strncpy(string,hdr->hist.scannum,10);
printf("scannum: <%s> \n", string);

strncpy(string,hdr->hist.patient_id,10);
printf("patient_id: <%s> \n", string);

strncpy(string,hdr->hist.exp_date,10);
printf("exp_date: <%s> \n", string);

strncpy(string,hdr->hist.exp_time,10);
printf("exp_time: <%s> \n", string);

strncpy(string,hdr->hist.hist_un0,10);
printf("hist_un0: <%s> \n", string);

printf("views:      <%d> \n", hdr->hist.views);
printf("vols_added: <%d> \n", hdr->hist.vols_added);
printf("start_field:<%d> \n", hdr->hist.start_field);
printf("field_skip: <%d> \n", hdr->hist.field_skip);
printf("omax: <%d> \n", hdr->hist.omax);
printf("omin: <%d> \n", hdr->hist.omin);
printf("smin: <%d> \n", hdr->hist.smax);
```

```
printf("smin: <%d> \n", hdr->hist.smin);

}

swap_hdr(pntr)
struct dsr *pntr;
    {
    swap_long(&pntr->hk.sizeof_hdr) ;
    swap_long(&pntr->hk.extents) ;
    swap_short(&pntr->hk.session_error) ;
    swap_short(&pntr->dime.dim[0]) ;
    swap_short(&pntr->dime.dim[1]) ;
    swap_short(&pntr->dime.dim[2]) ;
    swap_short(&pntr->dime.dim[3]) ;
    swap_short(&pntr->dime.dim[4]) ;
    swap_short(&pntr->dime.dim[5]) ;
    swap_short(&pntr->dime.dim[6]) ;
    swap_short(&pntr->dime.dim[7]) ;
    swap_short(&pntr->dime.unused1) ;
    swap_short(&pntr->dime.datatype) ;
    swap_short(&pntr->dime.bitpix) ;
    swap_long(&pntr->dime.pixdim[0]) ;
    swap_long(&pntr->dime.pixdim[1]) ;
    swap_long(&pntr->dime.pixdim[2]) ;
    swap_long(&pntr->dime.pixdim[3]) ;
    swap_long(&pntr->dime.pixdim[4]) ;
    swap_long(&pntr->dime.pixdim[5]) ;
    swap_long(&pntr->dime.pixdim[6]) ;
    swap_long(&pntr->dime.pixdim[7]) ;
    swap_long(&pntr->dime.vox_offset) ;
    swap_long(&pntr->dime.funused1) ;
    swap_long(&pntr->dime.funused2) ;
    swap_long(&pntr->dime.cal_max) ;
    swap_long(&pntr->dime.cal_min) ;
    swap_long(&pntr->dime.compressed) ;
    swap_long(&pntr->dime.verified) ;
    swap_short(&pntr->dime.dim_un0) ;
    swap_long(&pntr->dime.glmax) ;
    swap_long(&pntr->dime.glmin) ;
    }
```

```
swap_long(pntr)
unsigned char *pntr;
	{
	unsigned char b0, b1, b2, b3;

	b0 = *pntr;
	b1 = *(pntr+1);
	b2 = *(pntr+2);
	b3 = *(pntr+3);

	*pntr = b3;
	*(pntr+1) = b2;
	*(pntr+2) = b1;
	*(pntr+3) = b0;
	}

swap_short(pntr)
unsigned char *pntr;
	{
	unsigned char b0, b1;

	b0 = *pntr;
	b1 = *(pntr+1);

	*pntr = b1;
	*(pntr+1) = b0;
	}
```