Appendix B ANALYZE™ Coordinate System

The coordinate system employed by the *ANALYZE*™ programs is left-handed, with the coordinate origin in the lower left corner. Thus, with the subject lying supine, the coordinate origin is on the right side of the body *(x)*, at the back *(y)*, and at the feet *(z)*.

A major advantage of this convention is that the coordinate origin of each orthogonal orientation (transverse, coronal, and sagittal) lies in the lower left corner of the slice as it is displayed.

Orthogonal slices are numbered from one to the number of slices in that orientation. For example, a volume ***(x, y, z)*** dimensioned 128, 256, 48 has:

- 48 transverse slices numbered 1 through 48
- 128 sagittal slices numbered 1 through 128
- 256 coronal slices numbered 1 through 256

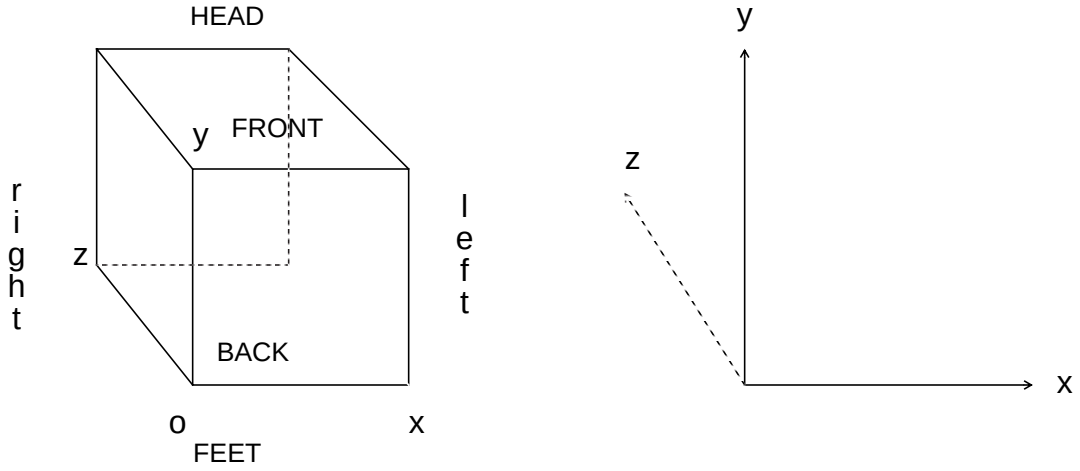
Pixel coordinates are made with reference to the slice numbers from which the pixels come. Thus, the first pixel in the volume is referenced *p(1,1,1)* and **not** at *p(0,0,0)*.

The names of the orthogonal planes can be changed in the *Configure* program.

This system is illustrated in the following diagrams.

Volume

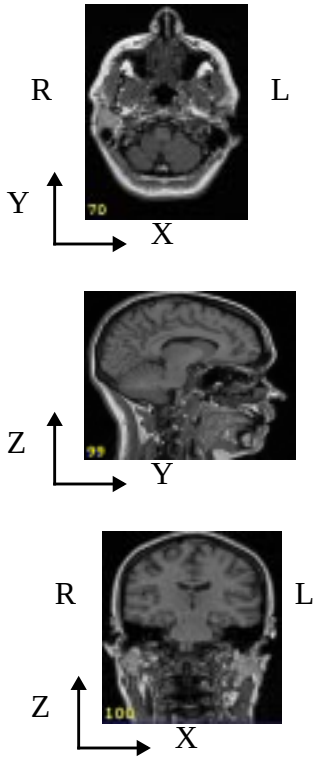
- Transverse slices are in the XY plane.
- Sagittal slices are in the ZY plane.
- Coronal slices are in the ZX plane.



Origin/Order Consistency

We refer to the *ANALYZE*™ coordinate system as an origin/order system because it is based on an unambiguous single 3-D origin in the image data, and on the idea that the order of pixels in one line of a section, the order of lines in a section, and the order of sections in a volume all proceed from the origin. As shown in Figure 2, the projection of the origin is always displayed at the lower left of the screen, and the order of slices is always from the origin outward. Image data in any of these three orientations (if properly identified) will be correctly rendered as a 3D image (i.e. left-sided defects will appear on the left side of the 3-D rendering). Flipping data in any of the three orientations about two axes will also result in a correctly rendered 3-D image, but the two axis flip will move the effective location of the origin, and the orthogonal sections may appear upside down or left/right reversed when displayed. Flipping the data about one or three axes will ruin the integrity of the data and cause 3-D renderings to appear mirror-reversed (i.e. right-sided lesions will appear to be on the left). Mislabeling transverse data sets as sagittal or vice-versa will not destroy the coherence of the data, and renderings will be correct although the labelling of axes will be in error. However, mislabeling transverse or sagittal images as coronal (or vice-versa) will cause renderings to be mirror reversed.

.



Transverse

View is from underneath.

First section is most inferior (bottom).

Sagittal

View is from right side.

First section is right side.

Coronal

View is from front.

First section is most posterior (back).

Figure 2. ANALYZE™ origin/order data convention.

It should be noted that the choice of a particular origin/order system is largely arbitrary, and that there are 16 possible origin/order systems based on the transverse orientation, and another 16 possible systems based on each of the other two orientations.

Appendix C ANALYZE™ Image Database

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:

- **1 bit** packed binary (slices must begin on byte boundaries)
- **8 bit** 8 bits per pixel (unsigned char)
- **16 bit** 16 bits per pixel (signed short)
- **32 bit** 32 bits per pixel signed integers, or floating point
- **64 bit** 64 bits per pixel; double precision, floating point, or complex.
- **24 bit** RGB, 8-bits per channel Red, Green, Blue.

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 (analyze_db.h)

```
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         */
          float funused8;                       /* 68 + 4          */
          float funused9;                       /* 72 + 4          */
          float funused10;                      /* 76 + 4          */
          float funused11;                      /* 80 + 4          */
          float funused12;                      /* 84 + 4          */
          float funused13;                      /* 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*/
```

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 */

0	Unknown data type
1	Binary (1 bit per voxel)
2	Unsigned character (8 bits per voxel)
4	Signed short (16 bits per voxel)
8	Signed integer (32 bits per voxel)
16	Floating point (32 bits per voxel)
32	Complex (64 bits per voxel; 2 floating point numbers)
64	Double precision (64 bits per voxel)

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 be 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");
}
```