# Appendix C

# Data File Header Information



Document number 2204, Revision C

# Structure of Data Headers

Neuroscan average (.AVG), coherence (.COH), compressed spectral array (.CSA), epoched EEG (.EEG), and continuous EEG (.CNT) files all share the same header structure. An SET setup file created in the ACQUIRE module is basically a header structure without data; the more recent AST setup files do not contain the 3.0 style headers that are in the other files. (You can export your AST file as a SET file, if needed; the AST format is proprietary, and not published).

The header consists of two parts: general, and channel-specific. The general part contains information that applies to all channels (such as digitization rate, pre-trigger interval, etc.), whereas the channel-specific part contains information which pertains to particular channels (such as electrode label, calibration factor, etc.). The size of the general part of the header is currently 900 bytes, and the size of the channel-specific part of the header is 75 bytes per channel. A C coded file called SETHEAD.H lists the full details of these header structures. You can find this file on your INSTALL CD in the Programming Examples subdirectory, or on the web site (http://www.neuro.com/neuroscan/download.htm#Conversion). Partial details of these structures are given below.

Beginning with version 4.1, a footer of varying length is appended to all data files. This contains information that is used by EDIT. Users can safely ignore this information. However, one is cautioned not to use the file size in determining the amount of waveform data contained in the file.

A C language structure called ELECTLOC for the electrode specific part of the header is defined below (see the SETHEAD.H file for details concerning the "reserved space"):

```
 /* STRUCTURE FOR INDIVIDUAL ELECTRODES */
typedef struct
{
  char lab[10];         /* Electrode label; last byte NULL   */
  char reserved1[5];    /* reserved space                    */
  short n;              /* # observations at each electrode  */
  char reserved2[30];   /* reserved space                    */
  short baseline;       /* baseline offset in raw ad units   */
  char reserved3[10];   /* reserved space                   */
  float sensitivity;    /* channel sensitivity               */
  char reserved4[8];    /* reserved space                    */
  float calib;          /* calibration coefficient      */
} ELECTLOC;
```

The structure for the general part of the header is called SETUP, and is defined below. (For full details regarding the "reserved space", please refer to the SETHEAD.H file.)



Document number 2204, Revision C

```
/* STRUCTURE FOR GENERAL PART OF ERP HEADER */
typedef struct
{
  char   rev[12];           /* Revision string              */
  char   type;              /* File type AVG=1, EEG=0       */
  char   id[20];            /* Patient ID                   */
  char   oper[20];          /* Operator ID                  */
  char   doctor[20];        /* Doctor ID                    */
  char   referral[20];      /* Referral ID                  */
  char    hospital[20];     /* Hospital ID                  */
  char    patient[20];      /* Subject name                 */
  short  age;               /* Subject Age                  */
  char   sex;               /* Subject Sex Male=M, Female=F */
  char   hand;              /* Handedness Mixed=M,Rt=R,lft=L*/
  char   med[20];           /* Medications                  */
  char   category[20];      /* Category                     */
  char   state[20];         /* Subject wakefulness          */
  char   label[20];         /* Session label                */
  char  date[10];           /* Session date string          */
  char  time[12];           /* Session time string          */
  char   reserved4[115];    /* reserved space               */
  short  compsweeps;        /* #  sweeps                    */
  short  acceptcnt;         /* # accepted sweeps            */
  short  rejectcnt;         /* # rejected sweeps            */
  short  pnts;              /* # points per waveform        */
  short  nchannels;         /* # active channels            */
  char   reserved5[3];      /* reserved space               */
  char   variance;          /* Variance data included flag  */
  unsigned short rate;      /* D-to-A rate (Hz)             */
  double scale;             /* scale factor for calibration */
  char  reserved6[111];     /* reserved space               */
  float dispmin;            /* display minimum              */
  float dispmax;            /* display maximum              */
  float xmin;               /* epoch start in seconds (neg) */
  float xmax;               /* epoch stop in seconds        */
  char reserved7[351];      /* reserved  space              */
  long NumSamples;          /* # samples in continuous file */
  char reserved8[18];       /* reserved space               */
  long EventTablePos;       /* offset to the event table    */
  float ContinousSeconds;  /* Number of seconds displayed per page  */
  long  ChannelOffset;     /* Block size of one channel in SYNAMPS  */
  char  AutoCorrectFlag;   /* Autocorrect of DC values              */
  char  DCThreshold;       /* Auto correct of DC level              */
} SETUP;
```

A fragment of C code to read the header of a file might look like the following:

```
#include <stdio.h>
#include <stdlib.h>
#define    N_ELECT  64
#include "sethead.h"

ELECTLOC  *channel[N_ELECT];
SETUP erp;      /* variable to contain header info   */
FILE *fp;       /* file pointer                       */
short i;

/* open file */
fp = fopen("test.eeg", "rb");

/* read general part of header */
fread(&erp, sizeof(SETUP), 1, fp);
/* read channel-specific part of header */
for  (i=0;i<erp.nchannels;i++)
{
     channel[i]=(ELECTLOC*)  malloc(sizeof(ELECTLOC));
     fread(channel[i],sizeof(ELECTLOC), 1, fp);
}
```

At this point, the erp structure contains all of the header information. The procedure for reading the header would be the same for .CNT, .AVG, .EEG, .CSA, .COH, and .SET files.

Actual data begins immediately after the header for .EEG, .AVG, and .CNT files. Thus, the offset to the beginning of data is 900 (=sizeof(SETUP)) + 75 (=sizeof(ELECTLOC)) * erp.nchannels bytes.

***.AVG Data.*** Average data is stored as 4-byte floats in vectored format for each channel. Each channel has a 5-byte header that is no longer used. Thus, after the main file header, there is an unused 5-byte header followed by erp.pnts of 4-byte floating point numbers for the first channel; then a 5-byte header for channel two followed by erp.pnts*sizeof(float) bytes, etc. Therefore, the total number of bytes after the main header is: erp.nchannels * (5 + erp.pnts*sizeof(float)). To scale a data point to microvolts, multiply by the channel-specific calibration factor (i.e, for electrode j: channel[j]->calib) and divide by the number of sweeps in the average (i.e., channel[j]->n);

***.COH data.*** Coherence data is the result of a pair-wise comparison between electrodes. Immediately following the data header is a comparison directory. This directory consists of a square matrix of offset numbers that point to the location of the coherency spectrum. It appears immediately after the data header. Here is a code fragment to read in the coherence directory:



Document number 2204, Revision C

```
// These declarations should appear at the
// the top of file
long ** CoherenceDirectory;
short int i;
short int size;

// Allocate memory and read - be sure to add protection
// for memory and read errors in your code!
size=sizeof(long)*erp.nchannels;
for(  i=0;i<erp.nchannels;i++)
{
     CoherenceDirectory[i]=(long)malloc(size);
     fread(CoherenceDirectory[i], size, 1, fp);
}
```

To access the coherency spectrum use the following code:

```
// declaration should appear at the top of your code
// row and column defines the electrode pair
short row;      //first electrode
short col;      //second electrode
float *real;    //holds real component
float *imag;    //holds imaginary component


// Allocate memory
real=(float*)malloc(sizeof(float)*erp.pnts);
imag=(float*)malloc(sizeof(float)*erp.pnts);

// Note - only non zero Directories are valid
// Main diagonal comparisons (row=col) are not valid
// This example selects a comparison of electrode # 1 with # 2
row = 1;
col = 2;

// Note - the extra 8 bytes added to the offset below is used
// to skip over a small unused header
fseek(fp,  CoherenenceDirectory[row][col]+8,SEEK_SET);

// Read in real array
fread(real, sizeof(float)*erp.pnts, 1, fp);

// Read in imaginary array
fread(imag, sizeof(float)*erp.pnts, 1, fp);

// To compute the coherence spectrum use the following code
// Note - results are placed in real array
for  (i=0;i<erp.pnts;i++)
```



Document number 2204, Revision C

```
    {
         real[i]=real[i]*real[i]+imag[i]*imag[i];
    }

    // To compute the coherence phase use the following code
    for  (i=0;i<erp.pnts;i++)
    {
         real[i]=atan2(imag[i]/real[i]);
    }
```

***.CSA Data***. Compressed spectral array data is stored as a three-dimensional sweep-by-electrode-by-point matrix of floating point numbers that start immediately after the data header. The following code fragment will read in a CSA matrix:

```
float  ***csa_buff;
short int size;   // assumes 32 bit integer!

//3d matrix allocation routine - see Numerical Recipes in C
// Note - 6 extra points are allocated for the mean frequency
// which is stuffed into erp.pnts+1 and 5 extra for expansion
matrix3_alloc(*csa_buff,erp.compsweeps,erp.nchannels,erp.pnts+6);

size=rp.compsweeps*(erp.pnts+6)*erp.nchannels*sizeof(float);
fread(csa_buff[0][0],size,1,fp);

// Data is stuffed into the csa_buff in the following order:
// sweep #,channel number, point
```

Values are stored as a floating point spectral numbers indexed from 0 to erp.pnts. The mean frequency of the spectrum is stored at location erp.pnts+1;

***.EEG Data.*** There are erp.compsweeps sweeps of data in an .EEG file. Each sweep of data consists of a sweep header followed by the EEG data. The sweep header could be characterized by the following structure:

```
typedef struct
{
  char  accept;   /* accept byte       */
  short ttype;    /* trial type        */
  short correct;  /* accuracy          */
  float rt;       /* reaction time     */
  short response; /* response type     */
  short reserved; /* not used          */
} SWEEP_HEAD;
```



Document number 2204, Revision C

After the sweep header, data is stored as 2-byte integers in multiplexed format; i.e., letting J=erp.nchannels and I=erp.pnts, the data points are as follows: data point #1 for channel #1, data point #1 for channel #2, . . ., data point #1 for channel #J; . . . ; data point #I for channel #1, data point #I for channel #2, . . ., data point #I for channel #J.

To scale a data point to microvolts for channel j, first subtract off the amplifier d.c. offset (if any) found in the variable channel[j]->baseline. Then multiply by the sensitivity (channel[j]- sensitivity) times the channel-specific scale factor (channel[j]->calib) divided by 204.8.

***.CNT (100 and 330 kHz modules)***. Data is stored as 2-byte (short) integers after the main header in continuous multiplexed format. Thus, the first data scan consists of erp.nchannels points (= 2*erp.nchannels bytes). The second data scan likewise contains erp.nchannels points (= 2*erp.nchannels bytes), etc., until the beginning of the event table is reached. (NOTE: This format differs from the 286 SCAN continuous format. Please refer to the 286 SCAN manual for details. Also, with ACQUIRE versions prior to 4.0 a different continuous format is used if you are using SYNAMPS with your SCAN system. Please see below for a description of the SYNAMP continuous file format)

Scaling of .CNT data to microvolts is identical to scaling of .EEG data — see above.

Following this data is the event table. The event table begins at a file offset that is given (in bytes) by the erp.EventTablePos variable. The following C definitions are used for the event table:

These are some preliminary definitions:

```
#define BYTE char
#define UBYTE unsigned char
#define WORD short
#define UWORD unsigned short

typedef enum
{
    TEEG_EVENT_TAB1=1,         //Event table tag type
    TEEG_EVENT_TAB2=2
}
TEEG_TYPE;
```

This structure describes a tag type 0 in a continuous file:

```
typedef struct
{
        TEEG_TYPE Teeg;
        long Size;
        union
```



Document number 2204, Revision C

```
        {
            void *Ptr;   // Memory pointer
            long Offset; // Relative file position
                         // 0 Means the data start immediately
                         // >0 Means the data starts at a
                         // relative offset
                         // from current position at the end
                         // of the tag
        };
} TEEG;
```

This structure describes an event type 1 in a continuous file:

```
typedef struct
{
  UWORD StimType; //range  0-65535
  UBYTE KeyBoard; //range  0-11 corresponding to function keys+1
  UBYTE KeyPad:4; //range  0-15  bit coded response pad
  UBYTE Accept:4; //values 0xd=Accept 0xc=Reject
  long Offset;    //file offset of event
} EVENT1;
```

This structure describes an event type 2 in a continuous file (it contains additional information regarding subject performance in a behavioral task):

```
typedef struct
{
  EVENT1  Event1;
  WORD    Type;
  WORD    Code;
  float   Latency;
  BYTE    EpochEvent;
  BYTE    Accept;
  BYTE    Accuracy;
} EVENT2;
```

At the beginning of the event table is a TEEG ("Tagged EEG") structure that is defined in the SETHEAD.H file. Following this structure is the event table proper. Currently, there are two types of event tables — the first with a minimum of event information, and the second with additional behavioral information. The Teeg variable in the TEEG structure indicates the type of event table.

Version 4.0 and earlier:

A raw data file (that has not been merged with a behavioral data file) is always of type 1. Depending of the event table type, the structure for each event is either EVENT1 or EVENT2.

Version 4.1 and later:

In previous versions, event tables contained Type 1 events until the file was merged with



Document number 2204, Revision C

behavioral data from STIM. Beginning with 4.1, ALL events are of Type 2 regardless of whether these files have been merged with behavioral data.

The number of events in the event table can be calculated by dividing the size variable in the TEEG structure by sizeof(EVENT1) (or sizeof(EVENT2), as the case may be). The events (of structure EVENT1 or EVENT2) immediately follow the TEEG structure.

***.CNT (SynAmps).*** The following applies to files created with ACQUIRE versions prior to 4.0. For files created with ACQUIRE version 4.1 and later see above.

When using SynAmps for continuous files, data are sent in a blocked rather than multiplexed format. Data are sent from the SynAmp in the following format:

| Block 1 | Block 2 |
|---|---|
| $X_{1,1}$ $X_{1,2}$ $X_{1,3}$....$X_{1,N}$ | $X_{1,1}$ $X_{1,2}$ $X_{1,3}$....$X_{1,N}$ |
| $X_{2,1}$ $X_{2,2}$ $X_{2,3}$....$X_{2,N}$ | $X_{2,1}$ $X_{2,2}$ $X_{2,3}$....$X_{2,N}$ |
| $X_{3,1}$ $X_{2,2}$ $X_{2,3}$....$X_{3,N}$ | $X_{3,1}$ $X_{2,2}$ $X_{2,3}$....$X_{3,N}$ |
| $X_{M,1}$ $X_{M,2}$ $X_{M,3}$....$X_{M,N}$ | $X_{M,1}$ $X_{M,2}$ $X_{M,3}$....$X_{M,N}$ |

Where `X` is a 16 bit integer, `N` refers to a data point and `M` the number of channels. Data are typically sent in blocks of N data points channel-by-channel. The size in bytes of one channel block is stored `erp.ChannelOffset` located in the SETUP structure (see above). To read these files you should allocate a buffer the size of `erp.ChannelOffset` and read each bock channel-by-channel. Since data is recorded continuously, you will need to concatenate each channel block into a continuous stream keeping track of channel alignment. Events are stored in an identical manner to the 100 and 330kHz modules (see above).

## *3DSpace .TRI file format*

Below you will find the description of the TRI file format as used in the 3D Space program. The TRI file contains the description of a triangulated head surface. The description uses C/C++ variables to describe elements.

Each file starts with a header, which looks like this:

**Long** ID (should be 100003, or 100004. (or 100002) )
**Short** Filetype (=2 for triangle file)
**short** revision
**float** electrodethickness
**float** electrodediameter
**BYTE** reserved[4080]

Then the facet and vertex information follows:

**short** number_of_facets
**short** number_of_vertices



Document number 2204, Revision C

Then for ***all*** the facets give by number_of_facets:

the centroid (centre of facet) x,y,z coordinates (unit vector) and it's length are writen as four **floats**.

Then follows the facet vertex coordinates for ***all*** vertices (given by number_of_vertices):

four **floats**: x, y, z, (normalized) and it's length

Then for ***all*** the facets (give by number_of_facets):

the three vertices that belong to this facet. These are three **shorts**, and are index values to the proper vertice as listed above (largest number is given by number_of_vertices).

Then the number of electrodes follows:

(**unsigned short**) number_of_electrodes

Then for ***all*** electrodes (given by number_of_electrodes):

**char** label[10] (label of electrode, max 9 chars + \0)
**short** key (normally = 'e' for electrode)
**float** x, y,z (position)
**unsigned short** ix (electrode index number)

## *Frequency domain EEG file format*

The frequency domain epoched EEG format has the same header as that described in the appendix of the SCAN manual (see the file sethead.h on the web at (http://www.neuro.com/neuroscan/download.htm#Conversion). For each sweep of data, there is a sweep header that is identical to that described on Page 6 above.

At this point there is a difference. After the sweep header, the frequency domain data for each sweep has the following format:

for each channel (erp.nchannels):

for each frequency bin (erp.pnts):

real value of the FFT stored as a 4-byte float;

for each frequency bin (erp.pnts):

imaginary value of the FFT stored as a 4-byte float;

The data have been scaled to microvolts prior to the FFT, and it is these results which are stored to the file.



Document number 2204, Revision C