

Kelunji Data File Formats

This document describes the format of Kelunji data files. There are a number of different formats, depending on which type of recorder was used and what processing has been performed on the file.

KELUNJI CLASSIC

The Kelunji Classic has its own custom file format which it uses to store all files internally. This is also the format the files will be in when they are copied from the recorder. Two variations are used: one when the file was recorded using a KA1 (12 bit with gain ranging) analogue board; and the other when using a KA2 (16 bit) analogue board.

Each individual file consists of a header followed immediately by the channel data. The channels are multiplexed together: that is all the data corresponding to one instant in time is stored together.

Kelunji Classic multi-byte values are stored in Little-Endian format; that is, the least significant byte is stored at the lowest address.

Kelunji Classic Header

Each file from a Kelunji Classic has a 256 byte header. This header is easiest to describe using the C structures which are actually present in the program.

The structures used within the header are:

```
typedef struct
{
    int8    century;
    int8    year;
    int8    month;
    int8    day;
    int8    hour;
    int8    minute;
    int8    second;
    int8    fill;
    int32   microsec;
} DATE_TIME;

typedef struct
{
    u_int32 time;
    u_int32 uncertainty;
    char    quality;
    char    first_motion;
    char    phase[8];
    char    picker[4];
    char    fill[2];
} ARRIVAL;

#define L_FORMAT      (20)
```

```
typedef struct
{
    int16    sample_period;
    int16    sample_rate;
    uint8    numb_chan;
    uint8    bytes_per_sample;
    char     format[L_FORMAT];
    uint16   min_exp;
} FORMAT;
```

```
typedef struct
{
    int32    correction;
    char     source[4];
} SYNC_VAL;
```

```
#define HD1_VERS      (4)
#define L_AUTHORITY  (4)
#define L_SITE_NAME  (4)
#define L_SITE_NUMB  (4)
#define L_OWNER      (6)
```

Using these sub-structures, it is now possible to define the complete 256 byte header.

```
typedef struct
{
    uint8    version;           // Header version
    char     rec_type;          // File type
    char     auth[L_AUTHORITY]; // The operating authority
    char     s_name[L_SITE_NAME]; // Site name
    char     s_num[L_SITE_NUMB]; // Site number
    uint16   recorder_num;      // Recorder number
    DATE_TIME cre_time;         // File creation/mod time
    FORMAT   format;            // The recording format
    DATE_TIME start;            // Record start time
    uint32   start_unc;         // and uncertainty
    DATE_TIME time;             // The event date and time
    char     place[40];         // The event place name
    SYNC_VAL sync;              // Sync to apply to the record
    char     prog_name[6];      // Prog which created the file
    char     prog_vers[6];      // and its version
    uint16   comp_type;         // The type of compression used
    uint16   trig_num;          // The trigger number
    uint32   length;            // Record length in samples
    ARRIVAL  p_arriv;           // The P arrival
    ARRIVAL  s_arriv;           // The S arrival
    uint32   finish;            // Finish time in samples
    int16    trig_type;         // Trigger type
    int16    trig_spare;        // and a spare
    int16    tv[8];             // Trigger values
    uint32   pre_trig_lev;      // Pre-trigger signal level
    int32    maximum_amp;       // The max amp of all channels
    uint32   pl_first_sample;   // First sample for plotting
    uint32   pl_numb_sample;    // Number of samples to plot
    uint32   pl_disp_amp;       // Display amplitude
    char     stype;              // Signal type (t, l, r etc.)
    char     fill[7];
    uint16   next_header_flag;  // If next header block
    char     hfill[2];
} HD1;
```

The most important values in the header are:

s_name	The site name
start	The file start time
length	The record length in samples
format.sample_rate	The sample rate
format.bytes_per_sample	The number of bytes per sample
format.format	A string describing the data format
format.min_exp	The minimum exponent used (see below)

The sample rate is given in samples per second. The format.format field will be one of the following strings (blank padded):

4E3(12N)	KA1	Four exponent bits then three channels of 12 bits
4E1(12N)	KA1	Four exponent bits then one channel of 12 bits
x(16N)	KA2	X channels (one to six) of 16 bits

KA1 Data Format

As indicated by the two possible format strings above, KA1 format files contain either two or five bytes per sample. Details will be shown for the five byte format, the two byte format is the same as the first two bytes of this. The five bytes are defined as follows (byte 0 is the earliest byte in the file):

Byte 4		Byte 3		Byte 2		Byte 1		Byte 0	
8 MSB Z		4 LSB Z, 4 MSB Y		8 LSB Y		8 MSB X		4 LSB X, 4 bit exp	

The value for each channel is then found from

True value = (12 bit value) << (exp - minimum exp)

What follows is some code from one of our programs which reads these files.

```

u_int8 *ip;
int32 *op;
int16 exp;
ip      = (u_int8*)(*theData);           // Where its coming from
op      = (int32*)(*aCH->fRealData)->data; // Where its going to
for (i=0; i<aCH->fRecordLength; i++)
{
    exp      = (ip[0] & 0x0F) - theH.format.min_exp;
    switch (chanNum)
    {
    case 1:
        value = ((ip[0]>>4) + (ip[1]<<4));
        break;
    case 2:
        value = (ip[2] + (ip[3]<<8));
        break;
    case 3:
        value = ((ip[3]>>4) + (ip[4]<<4));
        break;
    }
    value    &= 0xFFF;
    if (value & 0x0800) value -= 0x1000;
    value    <<= exp;
    *op++    = value;
    ip      += theH.format.bytes_per_sample;
}

```

KA2 Data Format

The KA2 data format is much simpler than the KA1 format. Each channel is a sixteen bit number which is stored as two bytes, least significant byte first. The channels are stored in the order they come from the KA2, that is X, then Y, then Z.

KELUNJI D SERIES

The Kelunji D series recorders store files in the PC Suds standard format. This standard is available from IASPEI.

A PC Suds file is a linked list of structures, each structure defines one aspect of the file and may or may not be followed by extra data.

In time, the SRC will use comments and other means to add additional information to PC Suds files.

BLOCK FILES

The Kelunji Classic and D Series both use a special file format for copying multiple files using the XModem file transfer protocol. This protocol does not support transferring multiple files in one session, therefore the SRC developed a format to transfer a number of individual files all grouped together in to one file.

The file consists of a header followed by each file in turn. Both the header and each file are padded to the XModem block size; that is either 128 or 1024 bytes. The header is a structure defined as:

```
struct file_contents
{
    u_int32  codeword;           // Must equal "FHBK" = 0x4B424846
    u_int16  version;          // Must equal 1
    u_int16  totNumKB;         // Total size of what's coming
    u_int16  numFile;          // Number of files to follow
    u_int16  blockSize;        // The XModem block size used
    u_int16  numHeadBlock;     // Number of XModem blocks in header
    u_int16  filler[5];        // Unused
    u_int16  fileSize[]       // The size of each file in XModem blocks
};
```

SEISMAC DATA FILES

When the SRC program SeisMac saves a seismogram, it does so using its own custom format. This format is described in a separate document.