Tucker-Davis Technologies TTank format:

Data collected or used by TDT software is stored in *tanks* that are logically divided into *blocks*. These blocks contain all the data from a single recording session such as an experimental run. Within each block different stores can record different types of events at different rates. This document is intended to provide the user with the details necessary to directly access data stored in these tanks.

The disk format for a TDT tank consists of a single folder (the tank directory) containing per-block subfolders. Hidden *desktop.ini* files are used to set the icon on all of these folders and as a means for TDT software to recognize tank folders. Within each block folder are four additional files each named according to "TANKNAME_BLOCKNAME.XXX" where XXX is one of the following extensions: TBK, TDX, TSQ, TEV.

The low level details of the TBK and TDX files are beyond the scope of this document. Briefly, the TBK contains block-level headers and offsets to the beginning of each second of data collection in the TSQ file. The TDX file contains epoch indexing information for faster access or filtering when dealing with epoch data. The remaining two files (the TSQ and TEV) contain the actual event headers and data meaning that usable TBK and TDX files can be reconstructed from these files with tools provided by TDT technical support if the TBK or TDX becomes lost or corrupted. The TSQ holds the header information for all the individual events whose data are in turn stored in the TEV.

The format of the TSQ file is simply a list of headers for each data segment or snippet stored in the TEV. Fields in the headers are little-endian, aligned at 32-bit boundaries (except for Sort Code), and are interpreted as follows:

Field	**Type**	**# Bytes**	**Comment**
Size	`long`	4	Size of data, with header, in longs.
Type	`long`	4	Snip, stream, scalar, etc.
Code	`long`	4	4-character event name (cast as long)
Channel	`unsigned short`	2	
Sort Code	`unsigned short`	2	Sort code for snip data.
Time Stamp	`double`	8	Event start time in seconds.
Event Offset or Strobe	`__int64 or double`	8	Offset in TEV file or raw value.
Data Format	`long`	4	float, long, etc.
Frequency	`float`	4	Sampling frequency

This results in each header being 10 words, or 40 bytes, long. This constant size allows for faster random access within the TSQ files. Data in the TEV file can be variable length, for example if different types of data were being recorded into the same block. Two fields also require additional explanation. The t*ype* field specifies the type of data this header contains or refers to (snippets, streaming, etc.) and the *data format* specifies the format of the data (float, int, etc.). The possible values are shown in the following 2 tables:

Type	**Value in Hexadecimal**
`EVTYPE_UNKNOWN`	`0x00000000`
`EVTYPE_STRON`	`0x00000101`
`EVTYPE_STROFF`	`0x00000102`
`EVTYPE_SCALER`	`0x00000201`
`EVTYPE_STREAM`	`0x00008101`
`EVTYPE_SNIP`	`0x00008201`
`EVTYPE_MARK`	`0x00008801`

Data Format	**Value**
`DFORM_FLOAT`	0
`DFORM_LONG`	1
`DFORM_SHORT`	2
`DFORM_BYTE`	3
`DFORM_DOUBLE`	4
`DFORM_QWORD`	5

In releases *after* version 70 a variation of the tank format is supported. In this variant the TEV file may not contain all of the collected data and missing channels can be found in specially named SEV files. This is done to allow users to copy or remove individual channels when dealing with very large data collections. The SEV file names start the same as the TEV files, but also contain the event name and channel number. For example, if the TEV file is named:

```
DEMOTANK2_Block-15.tev
```

then the SEV files will be named:

```
DEMOTANK2_Block-15_wavA_ch1.sev
DEMOTANK2_Block-15_wavA_ch2.sev
.
.
.
```

and so on with up to 1024 channels stored in individual SEV files. When accessing the tank through TDT's provided TTank interfaces this complication is hidden from the end user, however, when directly reading the files the user should check if these SEV files exist.

The remaining pages contain example C code that was created to access a tank in order to create a CSV file of the recorded data. Please feel free to copy this code and adjust it for your own uses.

```c
// Demo program showing how to read tsq file and tev file, and
// OpenSorter's sort code file.
#include <stdio.h>
#include <string.h>

// tank file structure
//---------------------
// tbk file has block events information and on second time offsets
// to efficiently locate events if the data is queried by time.
//
// tsq file is a list of event headers, each 40 bytes long,
// ordered strictly by time .
//
// tev file contains event binary data
//
// tev and tsq files work together to get an event's data and
//     attributes
//
// tdx file contains just information about epoc stores,
// is optionally generated after recording for fast retrieval
// of epoc information
//
// OpenSorter sort codes file structure
// ------------------------------------
// Sort codes saved by OpenSorter are stored in block subfolders such as
// Block-3\sort\USERDEFINED\EventName.SortResult.
//
// .SortResult files contain sort codes for 1 to all channels within
// the selected block.  Each file starts with a 1024 byte boolean channel
// map indicating which channel's sort codes have been saved in the file.
// Following this map, is a sort code field that maps 1:1 with the event
// ID for a given block.  The event ID is essentially the Nth occurance of
// an event on the data collection timeline. See lEventID below.

/* names and paths used in the code example to locate the tank files */
#define TANK_NAME "DEMOTANK2"
#define BLOCK_NAME "Block-3"
#define EV_NAME "LFPs"
//#define EV_NAME "eNe1"   // Spike event type.
#define CHANNEL 1
#define TANK_PATH "C:\\TDT\\OpenEx\\Tanks\\" TANK_NAME "\\" BLOCK_NAME "\\"

// Tank event types (tsqEventHeader.type).
#define EVTYPE_UNKNOWN  0x00000000
#define EVTYPE_STRON    0x00000101
#define EVTYPE_STROFF   0x00000102
#define EVTYPE_SCALAR   0x00000201
#define EVTYPE_STREAM   0x00008101
#define EVTYPE_SNIP           0x00008201
#define EVTYPE_MARK           0x00008801
#define EVTYPE_HASDATA  0x00008000

#define MAX_FILENAME_LEN 1000
/* TTank event header structure */
struct tsqEventHeader
{
    long           size;
    long           type;      /* event type: snip, pdec, epoc etc */
    long           code;      /* event name: must be 4 chars, cast as a long */
    unsigned short channel;    /* data acquisition channel */
    unsigned short sortcode;  /* sort code for snip data. See also OpenSorter
.SortResult file. */
    double         timestamp; /* time offset when even occurred */
    union
    {
        __int64    ev_offset; /* data offset in the TEV file */
        double     strobe;    /* raw data value */
    };
    long           format;    /* data format of event: byte, short, float
(usually), double */
    float          frequency; /* sampling frequency */
};

#define DATA_BUFFER_SIZE 10000
int main(int argc, char* argv[])
{
    FILE *tsq = NULL, *tev = NULL, *out = NULL;

    char bev[DATA_BUFFER_SIZE];
    char cTSQ[MAX_FILENAME_LEN];
    char cTEV[MAX_FILENAME_LEN];
    char cCSV[MAX_FILENAME_LEN];

    int     iEventCount = 0;
    long    lEventCodeOfInterest = *(long*)EV_NAME;

    sprintf(cTSQ,"%s",TANK_PATH TANK_NAME "_" BLOCK_NAME ".tsq");
    sprintf(cTEV,"%s",TANK_PATH TANK_NAME "_" BLOCK_NAME ".tev");
    sprintf(cCSV,"%s",TANK_PATH TANK_NAME "_" BLOCK_NAME ".csv");

    tsq = fopen(cTSQ, "r+b");
    tev = fopen(cTEV, "r+b");
    out = fopen(cCSV, "w");

    tsqEventHeader bsq;

    if (tsq != NULL && tev != NULL && out != NULL)
    {
        while(fread(&bsq, sizeof(bsq), 1, tsq) == 1)
        {
            iEventCount++;
            if ( bsq.code == lEventCodeOfInterest && bsq.channel == CHANNEL)
            {
                fsetpos(tev, &bsq.ev_offset);           // Move to signal data.

                long nPts = bsq.size - 10;              // Exclude header size
(longs).
                long nFetchSize = nPts*sizeof(float);   // 4 bytes per item.
                memset(bev,0,DATA_BUFFER_SIZE);
                fread(bev, nFetchSize, 1, tev);         // Read signal data.
                // Write to comma seperated file.
                for(long i = 0; i < nPts; i ++)
                    fprintf(out, "%E,", ((float*) bev)[i]);
                fprintf(out, "\r\n");
            }
        }
        // -----------------------------------------------
        // Read OpenSorter's sort code output file, if available.
        // Sort codes are only available for spike data types.
#define CHANNEL_MAP_SIZE 1024
        FILE *hSortResultIn  = NULL;
        FILE *hSortResultOut = NULL;
        char cSortResultIn[MAX_FILENAME_LEN];
        char cSortResultOut[MAX_FILENAME_LEN];

        // Boolean map of channel sort data in this file.
        char cChannelMap[CHANNEL_MAP_SIZE];

        unsigned char cSortCode = 0;
        long lEventID      = 0;
        long iCodeCount    = 0;
        long i = 0;

        rewind(tsq);    // Backup to beginning of event headers.

        // Typical OpenSorter output file.
        sprintf(cSortResultIn,"%s", TANK_PATH "sort\\CH1_SORT\\" EV_NAME
".SortResult");
        // Example output file.
        sprintf(cSortResultOut,"%s",TANK_PATH "sort\\CH1_SORT\\" EV_NAME
"_SORTCODES_OUT.txt");

        hSortResultIn  = fopen(cSortResultIn, "r+b");
        hSortResultOut = fopen(cSortResultOut, "w");

        // If .SortResult file is found...
        if ((hSortResultIn != NULL) && (hSortResultOut != NULL))
        {
            // Read channel map.
            fread(cChannelMap, CHANNEL_MAP_SIZE, 1, hSortResultIn );

            memset(&bsq, 0, sizeof(bsq));
            // Read next event header.
            while(fread(&bsq, sizeof(bsq), 1, tsq) == 1)
            {
                // If its the type we're looking for...
                if ((bsq.code == lEventCodeOfInterest) &&
                    (bsq.type == EVTYPE_SNIP))
                {
                    // If .sortresult file contains codes for this channel...
                    if (cChannelMap[bsq.channel])
                    {
                        // Seek into sort code field and read the code.
                        long SeekPoint = CHANNEL_MAP_SIZE + lEventID - 1;
                        fseek(hSortResultIn, SeekPoint, 0);
                        fread(&cSortCode, 1, 1, hSortResultIn);

                        fprintf(hSortResultOut, "%6d, %6d, %3d, %3d, %f\r\n",+
+iCodeCount, lEventID-1, bsq.channel, cSortCode, bsq.timestamp) ;
                    }
                }
                lEventID++; // Count all events in this block.
            }
        }
        if (hSortResultIn)
            fclose(hSortResultIn);
        if (hSortResultOut)
            fclose(hSortResultOut);

    }
    if (tsq)
        fclose(tsq);
    if (tev)
        fclose(tev);
    if (out)
        fclose(out);

    return 0;
}
```