EyeLink® EDF Access API

Beta Version 0.9

*** IMPORTANT: This is a Beta Version of the EDF Access API documentation. Great effort has been made to ensure that this release is as functional and as bug free as possible. ***

Table of Contents

Chapter 1. Introduction

The EyeLink EDF Access API is a set of C functions that provide access to EyeLink EDF files. The access method is similar to that of the online data access API, where the program performs a set of eyelink_get_next_data() and eyelink_get_float_data() calls to step through the data.

The EDF Access API also provides functions for setting bookmarks within an EDF file, and for automatically parsing an EDF file into a set of trials, with functions for stepping through the trial set.

As an example use for the API, the edf2asc translator program has been re-written to use the API for EDF data access. The source code for this edf2asc program is included with the API distribution.

This is the first release of the EDF Access API and should be considered a beta release.

Please report any functionality comments or bugs to sr-research@eyelinkinfo.com.

Chapter 2. API References for Programming in C

2.1 Data Access Structures

The EyeLink EDF access API (edfapi.dll) library defines a number of data types that are used for data reading, found in eye_data.h and edf.h. The useful parts of these structures are discussed in the following sections.

2.1.1 FSAMPLE Structure

The FSAMPLE structure holds information for a sample in the EDF file. Depending on the recording options set for the recording session, some of the fields may be empty.

```
typedef struct {
        UINT32 time;        /* time stamp of sample                 */
        UINT16 flags;       /* flags to indicate contents           */
        float px [2], py [2];   /* pupil x, y                       */
        float hx [2], hy [2];   /* headref x, y                     */
        float pa [2];       /* pupil size or area                   */
        float gx [2], gy [2];   /* screen gaze x, y                 */
        float rx, ry;       /* screen pixels per degree             */
        UINT16 status;      /* tracker status flags                 */
        UINT16 input;       /* extra (input word)                   */
        UINT16 buttons;     /* button state & changes               */
        INT16 htype;        /* head-tracker data type (0=none)      */
        INT16 hdata [8];    /* head-tracker data (not pre-scaled)   */
        UINT16 errors;      /* process error flags                  */
        float gxvel [2];    /* gaze x velocity                      */
        float gyvel [2];    /* gaze y velocity                      */
        float hxvel [2];    /* headref x velocity                   */
        float hyvel [2];    /* headref y velocity                   */
        float rxvel [2];    /* raw x velocity                       */
        float ryvel [2];    /* raw y velocity                       */
        float fgxvel [2];   /* fast gaze x velocity                 */
        float fgyvel [2];   /* fast gaze y velocity                 */
        float fhxvel [2];   /* fast headref x velocity              */
        float fhyvel [2];   /* fast headref y velocity              */
        float frxvel [2];   /* fast raw x velocity                  */
        float fryvel [2];   /* fast raw y velocity                  */

} FSAMPLE;
```

2.1.2 FEVENT Structure

The FEVENT structure holds information for an event in the EDF file. Depending on the recording options set for the recording session and the event type, some of the fields may be empty.

```
typedef struct {
        UINT32 time;            /* effective time of event              */
        INT16 type;             /* event type                           */
        UINT16 read;            /* flags which items were included      */
        INT16 eye;              /* eye: 0=left, 1=right                 */
        UINT32 sttime;          /* start time of the event              */
        UINT32 entime;          /* end time of the event                */
        float hstx, hsty;       /* headref starting points              */
        float gstx, gsty;       /* gaze starting points                 */
        float sta;
        float henx, heny;       /* headref ending points                */
        float genx, geny;       /* gaze ending points                   */
        float ena;
        float havx, havy;       /* headref averages                     */
        float gavx, gavy;       /* gaze averages                        */
        float ava;
        float avel;             /* accumulated average velocity         */
        float pvel;             /* accumulated peak velocity            */
        float svel, evel;       /* start, end velocity                  */
        float supd_x, eupd_x;   /* start, end units-per-degree          */
        float supd_y, eupd_y;   /* start, end units-per-degree          */
        UINT16 status;          /* error, warning flags                 */
        UINT16 flags;           /* error, warning flags                 */
        UINT16 input;
        UINT16 buttons;
        UINT16 parsedby;        /* 7 bits of flags: PARSEDBY codes*/
        LSTRING *message;       /* any message string                   */
} FEVENT;
```

2.1.3 RECORDINGS Structure

The RECORDINGS structure holds information about a recording block in an EDF file. A RECORDINGS structure is present at the start of recording and the end of recording. Conceptually a RECORDINGS structure is similar to the START and END lines inserted in an EyeLink ASC file. RECORDINGS with a state field = 0 represent the end of a recording block, and contain information regarding the recording options set before recording was initiated.

```
typedef struct
```

```
{
        UINT32 time;                    /* start time or end time               */
        byte state;                     * 0 = END, 1=START                      */
                /* 1 = SAMPLES, 2= EVENTS, 3= SAMPLES and EVENTS*/
        byte record_type;
        byte pupil_type;                /* 0 = AREA, 1 = DIAMETER               */
        byte recording_mode;            /*0 = PUPIL, 1 = CR                     */
        byte filter_type;               /*1, 2, 3                               */
        float sample_rate;              /* 250 or 500                           */
        byte pos_type;                  /* 0 = GAZE, 1= HREF, 2 = RAW           */
        byte eye;               /* 1=LEFT, 2=RIGHT, 3=LEFT and RIGHT */
} RECORDINGS;
```

2.1.4 ALLF_DATA structure

Any one of the above three data types can be read into a buffer of type ALLF_DATA, which is a union of the event, sample, and recording buffer formats:

```
typedef union {
            FEVENT    fe;
            FSAMPLE   fs;
            RECORDINGS rec;
          } ALLF_DATA;
```

2.1.5 EDFFILE Structure

EDFFILE is a dummy structure that holds an EDF file handle.

2.1.6 TRIAL Structure

The TRIAL structure is used to access a block of data within an EDF file that is considered to be a trial within the experimental session. The start time and end time of a TRIAL are defined using the edf_set_trial_identifier() function, where a start and end message text token is specified.

```
typedef struct {
        RECORDINGS *    rec; /* recording information about the current trial */
        unsigned int    duration;       /* duration of the current trial        */
        unsigned int    starttime;      /* start time of the trial              */
        unsigned int    endtime;        /* end time of the trial                */
} TRIAL;
```

2.1.7 BOOKMARK Structure

BOOKMARK is a dummy structure that holds a bookmark handle.

2.2 EDF Specific Functions

2.2.1 edf_get_version ()

char * **edf_get_version** ();

Returns a string which indicates the version of EDFAPI.dll library used.

Arguments: none

Returns: a string indicating the version of EDFAPI library used.

2.3 General EDF Data Access Functions

2.3.1 edf_open_file ()

EDFFILE * **edf_open_file** (
const char *edf_file_name,
int consistency,
int load_events,
int load_samples,
int *errval);

Opens the EDF file passed in by edf_file_name and preprocesses the EDF file.

Arguments: < edf_file_name >: name of the EDF file to be opened.
<consistency>: consistency check control (for the time stamps of the start and end events, etc).
consistency = 0 – no consistency check
consistency = 1 – check consistency and report
consistency = 2 – check consistency and fix.
<load_events>: load/skip loading events
load_events = 0 – do not load events
load_events = 1 – load events
<load_samples>: load/skip loading of samples
load_samples=0 – do not load samples
load_samples=1 – load samples

<errval>: This parameter is used for returning error value. The pointer should be a valid pointer to an integer. If the returned value is not 0 then an error occurred.

Returns: if successful a pointer to EDFFILE structure is returned. Otherwise NULL is returned.

2.3.2 edf_close_file ()

int **edf_close_file** (EDFFILE * edf);

Closes an EDF file pointed to by the given EDFFILE pointer and releases all of the resources (memory and physical file) related to this EDF file.

Arguments: <edf>: a valid pointer to EDFFILE structure. This should be created by calling edf_open_file ().

Returns: if successful it returns 0, otherwise a non zero is returned.

2.3.3 edf_get_next_data ()

int **edf_get_next_data** (EDFFILE *edf);

Returns the type of the next data element in the EDF file pointed to by *edf. Each call to edf_get_next_data() will retrieve the next data element within the data file. The contents of the data element are not accessed using this method, only the type of the element is provided. Use edf_get_float_data() instead to access the contents of the data element.

Arguments: <edf>: a valid pointer to EDFFILE structure. This handle should be created by calling edf_open_file().

Returns: One of the following values:

STARTBLINK	– the upcoming data is a start blink event
STARTSACC	– the upcoming data is a start saccade event
STARTFIX	– the upcoming data is a start fixation event
STARTSAMPLES	– the upcoming data is a start samples event
STARTEVENTS	– the upcoming data is a start events event
STARTPARSE	– the upcoming data is a start parse event
ENDBLINK	– the upcoming data is an end blink event
ENDSACC	– the upcoming data is an end saccade event
ENDFIX	– the upcoming data is an end fixation event
ENDSAMPLES	– the upcoming data is an end samples event
ENDEVENTS	– the upcoming data is an end events event

ENDPARSE – the upcoming data is an end parse event

FIXUPDATE – the upcoming data is a fixation update event
BREAKPARSE – the upcoming data is a break parse event
BUTTONEVENT – the upcoming data is a button event
INPUTEVENT – the upcoming data is an input event
MESSAGEEVENT – the upcoming data is a message event

SAMPLE_TYPE – the upcoming data is a sample
RECORDING_INFO – the upcoming data is a recording info
NO_PENDING_ITEMS – no more data left.

2.3.4 edf_get_float_data ()

ALLF_DATA *** edf_get_float_data** (EDFFILE *edf);

Returns the float data with the type returned by edf_get_next_data(). This function does not move the current data access pointer to the next element; use edf_get_next_data() instead to step through the data elements.

Arguments: <edf>: a valid pointer to EDFFILE structure. This handle should be created by calling edf_open_file().

Returns: Returns a pointer to the ALLF_DATA structure with the type returned by edf_get_next_data().

2.3.5 edf_get_preamble_text ()

int **edf_get_preamble_text** (EDFFILE *edf, char * buffer, int length);

Copies the preamble text into the given buffer. If the preamble text is longer than the length the text will be truncated. The returned content will always be null terminated.

Arguments: <edf>: a valid pointer to EDFFILE structure. This handle should be created by calling edf_open_file().
<buffer>: a character array to be filled by the preamble text.
<length>: length of the buffer.

Returns: returns 0 if the operation is successful.

2.3.6 edf_get_preamble_text_length ()

int **edf_get_preamble_text_length** (EDFFILE * edf);

Returns the length of the preamble text.

Arguments: <edf>: a valid pointer to EDFFILE structure. This handle should be created by calling edf_open_file().

Returns: An integer for the length of preamble text

2.3.7 edf_get_element_count ()

int **edf_get_element_count** (EDFFILE *edf);

Returns the number of elements (samples, eye events, messages, buttons, etc) in the EDF file.

Arguments: <edf>: a valid pointer to EDFFILE structure. This should be created by calling edf_open_file.

Returns: the number of elements in the EDF file.

2.4 Trial Related Functions

The EDF access API also provides the following trial related functions for the ease of counting the total number the trials in the recording file and navigating between different trials. To use this functionality, it is desirable that the user first define the trial start/end identifier strings with edf_set_trial_identifier(). [The identifier string settings can be checked with the edf_get_start_trial_identifier() and edf_get_end_trial_identifier() functions]. Use edf_jump_to_trial(), edf_goto_previous_trial(), edf_goto_next_trial(), edf_goto_trial_with_start_time(), or edf_goto_trial_with_end_time() functions to go to a target trial. The recording and start/end time of the target trial can be checked with edf_get_trial_header().

2.4.1 edf_set_trial_identifier ()

int **edf_set_trial_identifier** (EDFFILE * edf, char *start_marker_string, char * end_marker_string);

Sets the message strings that mark the beginning and the end of a trial. The message event that contains the marker string is considered start or end of the trial.

Arguments: <edf>: a valid pointer to EDFFILE structure. This should be created by calling edf_open_file().
<start_marker_string>: string that contains the marker for beginning of a trial.
<end_marker_string>: string that contains the marker for end of the trial.

Returns: 0 if no error occurred.

Note: The following restrictions apply for collecting the trials.

1. The <start_marker_string> message should be before the start recording (indicated by message "START").
2. The <end_marker_string> message should be after the end recording (indicated by message "END").
3. If the <start_marker_string> is not found before start recording or if the <start_marker_string> is null, start recording will be the starting position of the trial.
4. If the <end_marker_string> is not found after the end recording, end recording will be the ending position of the trial.
5. If <start_marker_string> is not specified the string "TRIALID", if found, will be used as the <start_marker_string>.
6. If the <end_marker_string> is not specified, the beginning of the next trial is the end of the current trial.

2.4.2 edf_get_start_trial_identifier ()

char* **edf_get_start_trial_identifier** (EDFFILE * edf);

Returns the trial identifier that marks the beginning of a trial.

Arguments: <edf>: a valid pointer to EDFFILE structure. This should be created by calling edf_open_file().

Returns: a string that marks the beginning of a trial.

2.4.3 edf_get_end_trial_identifier ()

char* **edf_get_end_trial_identifier** (EDFFILE * edf);

Returns the trial identifier that marks the end of a trial.

Arguments: <edf>: a valid pointer to EDFFILE structure. This should be created by calling edf_open_file().

Returns: a string that marks the end of a trial.

2.4.4 edf_get_trial_count ()

int **edf_get_trial_count** (EDFFILE *edf);

Returns the number of trials in the EDF file.

Arguments: <edf>: a valid pointer to EDFFILE structure. This should be created by calling edf_open_file().

Returns: an integer for the number of trials in the EDF file.

2.4.5 edf_jump_to_trial ()

int **edf_jump_to_trial** (EDFFILE * edf, int trial);

Jumps to the beginning of a given trial.

Arguments: <edf>: a valid pointer to EDFFILE structure. This should be created by calling edf_open_file().
<trial>: trial number. This should be a value between 0 and edf_get_trial_count ()-1.

Returns: unless there are any errors it returns a 0.

2.4.6 edf_get_trial_headers ()

int **edf_get_trial_header** (EDFFILE * edf, TRIAL *trial);

Returns the trial specific information. See the TRIAL structure for more details.

Arguments: <edf>: a valid pointer to EDFFILE structure. This should be created by calling edf_open_file().
<trial>: pointer to a valid TRIAL structure (note <trial> must be initialized before being used as a parameter for this function). This pointer is used to hold information of the current trial.

Returns: unless there are any errors it returns 0.

2.4.7 edf_goto_previous_trial ()

int **edf_goto_previous_trial** (EDFFILE * edf);

Jumps to the beginning of the previous trial.

Arguments: <edf>: a valid pointer to EDFFILE structure. This should be created by calling edf_open_file().

Returns: unless there are any errors it returns 0.

2.4.8 edf_goto_next_trial ()

int **edf_goto_next_trial** (EDFFILE * edf);

Jumps to the beginning of the next trial.

Arguments: <edf>: a valid pointer to EDFFILE structure. This should be created by calling edf_open_file().

Returns: unless there are any errors it returns 0.

2.4.9 edf_goto_trial_with_start_time ()

int **edf_goto_trial_with_start_time** (EDFFILE * edf,
unsigned int start_time);

Jumps to the trial that has the same start time as the given start time.

Arguments: <edf>: a valid pointer to EDFFILE structure. This should be created by calling edf_open_file().

Returns: unless there are any errors it returns 0.

2.4.10 edf_goto_trial_with_end_time ()

int **edf_goto_trial_with_start_time** (EDFFILE * edf,
unsigned int end_time);

Jumps to the trial that has the same start time as the given end time.

Arguments: <edf>: a valid pointer to EDFFILE structure. This should be created by calling edf_open_file().

Returns: unless there are any errors it returns 0.

2.5 Bookmark Related Functions

In addition to navigation between different trials in an EDF recording file with the functions provided in the previous section, the EDF access API also allows the user to "bookmark" any position of the EDF file using the edf_set_bookmark() function. The bookmarks can be revisited with edf_goto_bookmark(). Finally, the bookmarks should be freed with the edf_free_bookmark() function call.

2.5.1 edf_set_bookmark ()

int **edf_set_bookmark** (EDFFILE *edf, BOOKMARK *bm);

Bookmark the current position of the edf file.

Arguments: <edf>: a valid pointer to EDFFILE structure. This should be created by calling edf_open_file.
<bm>: pointer to a valid BOOKMARK structure. This structure will be filled by this function. <bm> should be initialized before being used by this function.

Returns: unless there are any errors it returns 0.

2.5.2 edf_free_bookmark ()

int **edf_free_bookmark** (EDFFILE *ef, BOOKMARK *bm);

Removes an existing bookmark.

Arguments: <edf>: a valid pointer to EDFFILE structure. This should be created by calling edf_open_file.
<bm>: pointer to a valid BOOKMARK structure. This structure will be filled by this function. Before calling this function edf_set_bookmark should be called and bm should be initialized there.

Returns: unless there are any errors it returns 0.

2.5.3 edf_goto_bookmark ()

int **edf_goto_bookmark** (EDFFILE *ef, BOOKMARK *bm);

Jumps to the given bookmark.

Arguments: <edf>: a valid pointer to EDFFILE structure. This should be created by calling edf_open_file.
<bm>: pointer to a valid BOOKMARK structure. This structure will be filled by this function. Before calling this function edf_set_bookmark should be called and bm should be initialized there.

Returns: unless there are any errors it returns 0.

Chapter 3. Compiling

Ensure that you have copied the library for the operating system you are using to a directory that is in your library path.

3.1 Windows

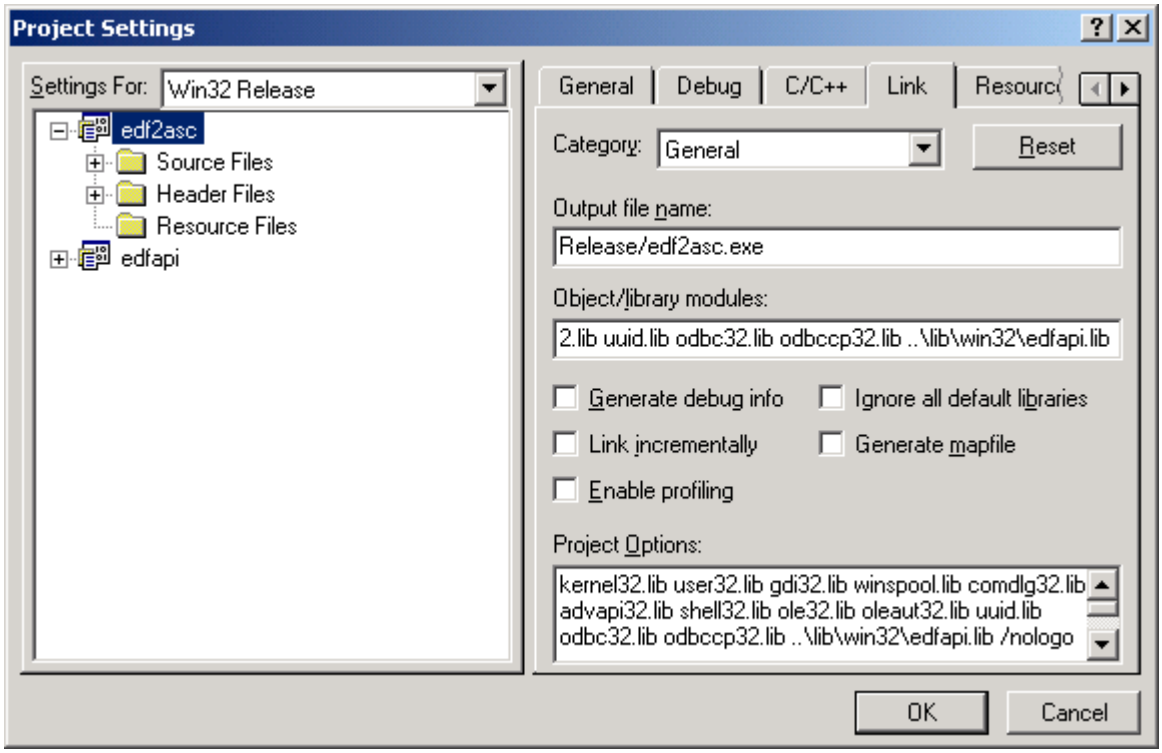
Under the directory “EDF_Access_API\lib\win32” you will find edfapi.dll and edfapi.lib. Also, under the directory “EDF_Access_API\Example” you will find the following header files:

1. edf.h
2. eye_data.h
3. eyetypes.h

In your program you will include edf.h, which contains all the edfapi function declarations. The eye_data.h and eyetypes.h contains necessary data structure and types.

3.1.1 Visual Studio

While linking under visual studio you may either include edfapi.lib as one of your source file or pass edfapi.lib as an argument to the linker. For the latter approach, please note the object/library modules settings in the screen shot below.



3.2 Linux

Under the directory "EDF_Access_API\lib\linux", you will find libedfapi.so and libedfapi.a. Also, under the directory "EDF_Access_API\Example" you will find the following header files:

1. edf.h
2. eye_data.h
3. eyetypes.h

In your program you will include edf.h, which contains all the edfapi function declarations. The eye_data.h and eyetypes.h contains necessary data structure and types.

3.2.1 Gcc

See Makefile.linux for an example of compiling and linking with gcc compiler under linux platform.

Static compilation:

gcc -static <gcc options> -o output_file_name <your sources to compile>
-L$(INSTALLDIR)/../lib/linux/ -ledfapi –lm

Shared compilation:

gcc <gcc options> -o output_file_name <your sources to compile>
-L$(INSTALLDIR)/../lib/linux/ -ledfapi -lm

3.3 MacOS X

Under the directory "EDF_Access_API\lib\macosx", you will find libedfapi.dylib and libedfapi.a. Also, under the directory "EDF_Access_API\Example" you will find the following header files:

1. edf.h
2. eye_data.h
3. eyetypes.h

In your program you will include edf.h, which contains all the edfapi function declarations. The eye_data.h and eyetypes.h contains necessary data structure and types.

3.3.1 Gcc

See Makefile.osx for a complete example of compiling and linking with gcc compiler under osx platform.

Dynamic compilation:

gcc <gcc options> -o output_file_name <your sources to compile>
-L$(INSTALLDIR)/../lib/macosx/ -dynamic -ledfapi

Static compilation:

gcc <gcc options> -o output_file_name <your sources to compile> \
-L$(INSTALLDIR)/../lib/macosx/ -static -ledfapi