EyeLink ® EDF Access API

Generated by Doxygen 1.8.11

Contents

Chapter 1

EyeLink EDF Access API

1.1 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 support@sr-research.com.

Chapter 2

classIntro

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.

Chapter 3

Compling

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 "<Program files (x86)>\SR Research\EyeLink\libs" you will find edfapi.dll and edfapi.lib for 32bit windows and under the directory "<Program files (x86)>\SR Research\EyeLink\libs\x64" you will find edfapi64.dll and edfapi64.lib for 64bit windows

Also, under the directory "<Program files (x86)>\SR Research\EyeLink\includes\eyelink" you will find the following header files:

1. edf.h
2. edf_data.h
3. edftypes.h

In your program you will include edf.h, which contains all the edfapi function declarations. The edf_data.h and edftypes.h contain necessary data structures and types.

3.1.1 Visual Studio

While linking under visual studio you may either include edfapi.lib/edfapi64.lib as one of your source file or pass edfapi.lib/edfapi64.lib as an argument to the linker. For the latter approach, please note the object/library modules settings in the screen shot below.

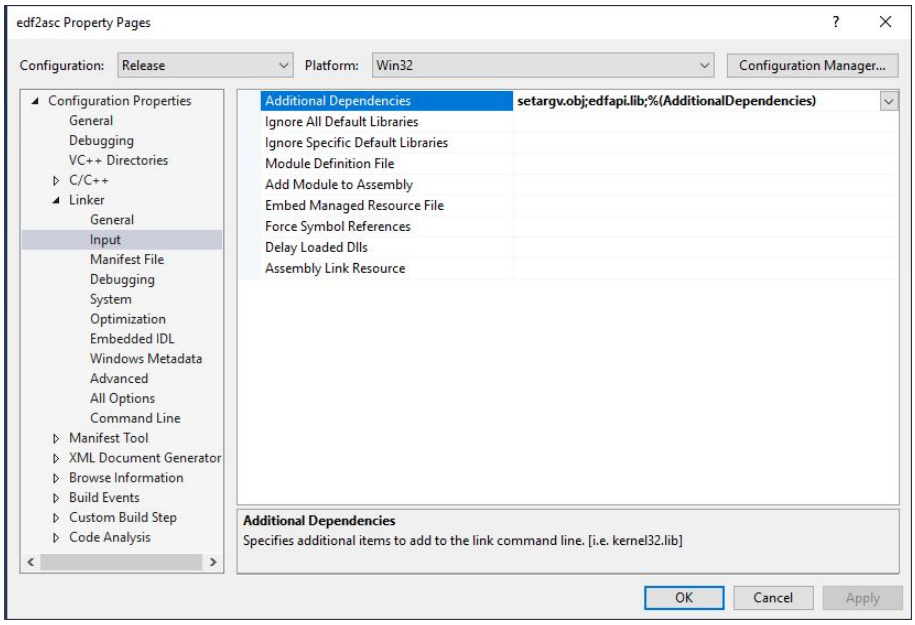

Figure 3.1 Visual Studio 2015 Link Settings for edf2asc

The Includes/Library file search path settings are shown below.

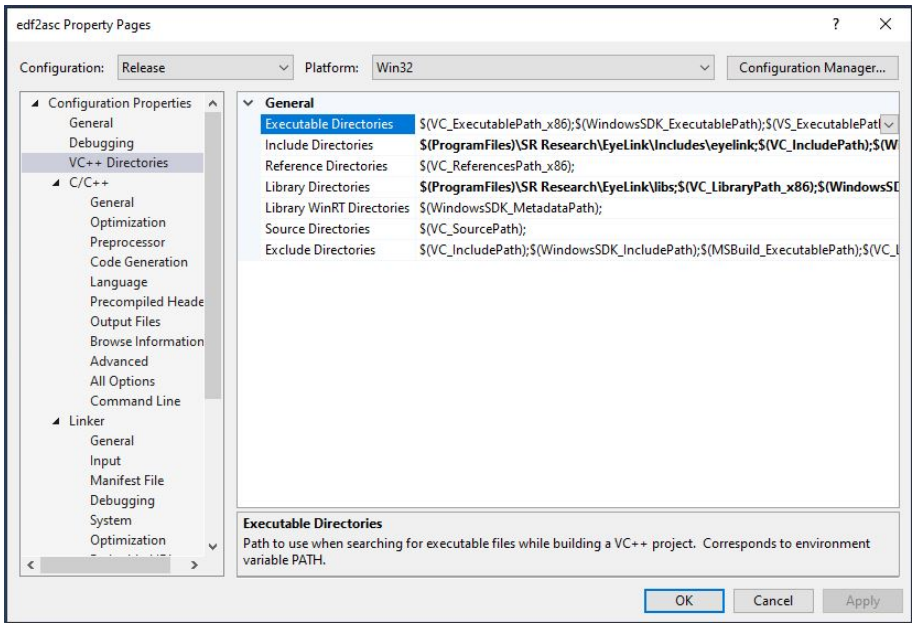

Figure 3.2 Visual Studio 2015 Includes/Link File Search Path for edf2asc

3.2 Linux

Under the directory "/usr/lib/x86_64-linux-gnu" , you will find libedfapi.so and libedfapi.a.

Also, under the directory "/usr/include/EyeLink" you will find the following header files:

1. edf.h
2. edf_data.h
3. edftypes.h

In your program you will include edf.h, which contains all the edfapi function declarations. The edf_data.h and edftypes.h contain necessary data structures 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> -ledfapi -lm

Shared compilation:
gcc <gcc options> -o output_file_name <your sources to compile> -ledfapi -lm

3.3 macOS

The edfapi.framework/Headers installed by default under /Library, contains the following header files:

1. edf.h
2. edf_data.h
3. edftypes.h

In your program you will include edf.h, which contains all the edfapi function declarations. The edf_data.h and edftypes.h contain necessary data structures and types.

Please note that we no longer provide libedfapi.dylib and libedfapi.a for macOS. Also note we no longer provide Makefile for macOS. Please refer to the provided XCode project in CExample folder.

Chapter 4

Module Index

4.1 Modules

Here is a list of all modules:

Chapter 5

Class Index

5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 6

Module Documentation

6.1 General EDF Data Access Functions

Functions

- PUBLIC_FN EDFFILE * edf_open_file (const char *fname, int consistency, int loadevents, int loadsamples, int *errval)

  Opens the EDF file passed in by edf_file_name and preprocesses the EDF file.
- PUBLIC_FN EDFFILE * edf_open_file_ex (const char *fname, int consistency, int loadevents, int loadsamples, CONFIG *config, int *errval)

  Opens the EDF file passed in by edf_file_name and preprocesses the EDF file.
- PUBLIC_FN int edf_close_file (EDFFILE *ef)

  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.
- PUBLIC_FN int **edf_open_logfile** (EDFFILE *ef, const char *fname)
- PUBLIC_FN int **edf_set_extra_message_file** (const char *fname)
- PUBLIC_FN const char * **edf_get_logmsg** (EDFFILE *ef)
- PUBLIC_FN PUBLIC_FN int edf_get_next_data (EDFFILE *ef)

  *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.*
- PUBLIC_FN ALLF_DATA * edf_get_float_data (EDFFILE *ef)
- PUBLIC_FN unsigned int edf_get_element_count (EDFFILE *ef)
- PUBLIC_FN int edf_get_preamble_text (EDFFILE *ef, char *buffer, int length)
- PUBLIC_FN int edf_get_preamble_text_length (EDFFILE *edf)

6.1.1 Detailed Description

6.1.2 Function Documentation

6.1.2.1 PUBLIC_FN EDFFILE* edf_open_file (const char * *fname*, int *consistency*, int *loadevents*, int *loadsamples*, int * *errval*)

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

Parameters

fname	name of the EDF file to be opened.
consistency	onsistency check control (for the time stamps of the start and end events, etc). `0`, no consistency check. `1`, check consistency and report. `2`, check consistency and fix.
loadevents	load/skip loading events `0`, do not load events. `1`, load events.
loadsamples	load/skip loading of samples `0`, do not 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.

6.1.2.2 PUBLIC_FN EDFFILE* edf_open_file_ex (const char * *fname*, int *consistency*, int *loadevents*, int *loadsamples*, CONFIG * *config*, int * *errval*)

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

Parameters

fname	name of the EDF file to be opened.
consistency	onsistency check control (for the time stamps of the start and end events, etc). `0`, no consistency check. `1`, check consistency and report. `2`, check consistency and fix.
loadevents	load/skip loading events `0`, do not load events. `1`, load events.
loadsamples	load/skip loading of samples `0`, do not load samples. `1`, load samples.
config	Currently only used to set the default resolution. In the future, this will be used to pass more configuration parameters.
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.

6.1.2.3 PUBLIC_FN int edf_close_file (EDFFILE * *ef*)

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.

Parameters

ef	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.

6.1.2.4 PUBLIC_FN PUBLIC_FN int edf_get_next_data (EDFFILE * *ef*)

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.

Parameters

ef	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.

6.1.2.5 PUBLIC_FN ALLF_DATA* edf_get_float_data (EDFFILE * *ef*)

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.

Parameters

ef	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().

6.1.2.6 PUBLIC_FN unsigned int edf_get_element_count (EDFFILE * *ef*)

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

Parameters

ef	a valid pointer to EDFFILE structure. This should be created by calling edf_open_file.

Returns

the number of elements in the EDF file.

6.1.2.7 PUBLIC_FN int edf_get_preamble_text (EDFFILE * *ef*, 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.

Parameters

ef	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.

6.1.2.8 PUBLIC_FN int edf_get_preamble_text_length (EDFFILE * *edf*)

Returns the length of the preamble text.

Parameters

edf	a valid pointer to c EDFFILE structure. This handle should be created by calling edf_open_file().

Returns

An integer for the length of preamble text.

6.2 Trial Related Functions

Functions

- PUBLIC_FN 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.
- PUBLIC_FN char * edf_get_start_trial_identifier (EDFFILE *ef)

  Returns the trial identifier that marks the beginning of a trial.
- PUBLIC_FN char * edf_get_end_trial_identifier (EDFFILE *ef)

  Returns the trial identifier that marks the end of a trial.
- PUBLIC_FN int edf_get_trial_count (EDFFILE *edf)

  Returns the number of trials in the EDF file.
- PUBLIC_FN int edf_jump_to_trial (EDFFILE *edf, int trial)
- PUBLIC_FN int edf_get_trial_header (EDFFILE *edf, TRIAL *trial)

  Returns the trial specific information. See the TRIAL structure for more details.
- PUBLIC_FN int edf_goto_previous_trial (EDFFILE *edf)

  Jumps to the beginning of the previous trial.
- PUBLIC_FN int edf_goto_next_trial (EDFFILE *edf)

  Jumps to the beginning of the next trial.
- PUBLIC_FN 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.
- PUBLIC_FN int edf_goto_trial_with_end_time (EDFFILE *edf, unsigned int end_time)

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

6.2.1 Detailed Description

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().

6.2.2 Function Documentation

6.2.2.1 PUBLIC_FN 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.

Parameters

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.

Remarks

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.

6.2.2.2 PUBLIC_FN char* edf_get_start_trial_identifier (EDFFILE * *ef*)

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

Parameters

ef	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.

6.2.2.3 PUBLIC_FN char* edf_get_end_trial_identifier (EDFFILE * *ef*)

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

Parameters

ef	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.

6.2.2.4 PUBLIC_FN int edf_get_trial_count (EDFFILE * *edf*)

Returns the number of trials in the EDF file.

Parameters

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.

6.2.2.5 PUBLIC_FN int edf_jump_to_trial (EDFFILE * *edf*, int *trial*)

Jumps to the beginning of a given trial.

Parameters

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.

6.2.2.6 PUBLIC_FN int edf_get_trial_header (EDFFILE * *edf*, TRIAL * *trial*)

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

Parameters

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 a 0.

6.2.2.7 PUBLIC_FN int edf_goto_previous_trial (EDFFILE * *edf*)

Jumps to the beginning of the previous trial.

Parameters

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`.

6.2.2.8 PUBLIC_FN int edf_goto_next_trial (EDFFILE * *edf*)

Jumps to the beginning of the next trial.

Parameters

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`.

6.2.2.9 PUBLIC_FN 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.

Parameters

edf	a valid pointer to `EDFFILE` structure. This should be created by calling edf_open_file().
start_time	start time of the EDF trial

Returns

unless there are any errors it returns 0.

6.2.2.10 PUBLIC_FN int edf_goto_trial_with_end_time (EDFFILE * *edf*, unsigned int *end_time*)

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

Parameters

edf	a valid pointer to `EDFFILE` structure. This should be created by calling edf_open_file().
end_time	end time of the EDF trial

Returns

unless there are any errors it returns `0`.

6.3 Bookmark Related Functions

Functions

- PUBLIC_FN int edf_set_bookmark (EDFFILE *ef, BOOKMARK *bm)
- PUBLIC_FN int edf_free_bookmark (EDFFILE *ef, BOOKMARK *bm)
- PUBLIC_FN int edf_goto_bookmark (EDFFILE *ef, BOOKMARK *bm)

6.3.1 Detailed Description

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.

6.3.2 Function Documentation

6.3.2.1 PUBLIC_FN int edf_set_bookmark (EDFFILE * *ef*, BOOKMARK * *bm*)

Bookmark the current position of the edf file.

Parameters

ef	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.

6.3.2.2 PUBLIC_FN int edf_free_bookmark (EDFFILE * *ef*, BOOKMARK * *bm*)

Removes an existing bookmark

Parameters

ef	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.

6.3.2.3 PUBLIC_FN int edf_goto_bookmark (EDFFILE * *ef*, BOOKMARK * *bm*)

Jumps to the given bookmark.

Parameters

ef	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.

6.4 EDF Specific Functions

Functions

- PUBLIC_FN const char * edf_get_version ()

6.4.1 Detailed Description

6.4.2 Function Documentation

6.4.2.1 PUBLIC_FN const char* edf_get_version ( )

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

Returns

a string indicating the version of EDFAPI library used.

Chapter 7

Class Documentation

7.1 ALLF_DATA Union Reference

Public Attributes

- FEVENT **fe**
- IMESSAGE **im**
- IOEVENT **io**
- FSAMPLE **fs**
- RECORDINGS **rec**

7.1.1 Detailed Description

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:

7.2 BOOKMARK Struct Reference

Public Attributes

- unsigned int **id**

7.2.1 Detailed Description

BOOKMARK is a dummy structure that holds a bookmark handle.

7.3 CONFIG Struct Reference

Public Attributes

- float default_rx
- float default_ry

7.3.1 Detailed Description

The CONFIG structure holds optional configuration information for EDF file processing. User configured parameters or application specif parameters can be passed to edfapi via this configuration.

7.3.2 Member Data Documentation

7.3.2.1 float CONFIG::default_rx

user defined resolution rx value

7.3.2.2 float CONFIG::default_ry

user defined resolution ry value

7.4 EDFFILE Struct Reference

7.4.1 Detailed Description

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

7.5 FEVENT Struct Reference

Public Attributes

- UINT32 time
- INT16 type
- UINT16 read
- UINT32 sttime
- UINT32 entime
- float hstx
- float hsty
- float gstx
- float gsty
- float sta
- float henx
- float heny
- float genx
- float geny
- float ena
- float havx
- float havy
- float gavx
- float gavy
- float ava
- float avel

- float pvel
- float svel
- float evel
- float supd_x
- float eupd_x
- float supd_y
- float eupd_y
- INT16 eye
- UINT16 status
- UINT16 flags
- UINT16 **input**
- UINT16 **buttons**
- UINT16 parsedby
- LSTRING * message

7.5.1 Detailed Description

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.

7.5.2 Member Data Documentation

7.5.2.1 UINT32 FEVENT::time

effective time of event

7.5.2.2 INT16 FEVENT::type

event type

7.5.2.3 UINT16 FEVENT::read

flags which items were included

7.5.2.4 UINT32 FEVENT::sttime

start time of the event

7.5.2.5 UINT32 FEVENT::entime

end time of the event

7.5.2.6 float FEVENT::hstx

headref starting points

7.5.2.7 float FEVENT::hsty

headref starting points

7.5.2.8 float FEVENT::gstx

gaze starting points

7.5.2.9 float FEVENT::gsty

gaze starting points

7.5.2.10 float FEVENT::sta

pupil size at start

7.5.2.11 float FEVENT::henx

headref ending points

7.5.2.12 float FEVENT::heny

headref ending points

7.5.2.13 float FEVENT::genx

gaze ending points

7.5.2.14 float FEVENT::geny

gaze ending points

7.5.2.15 float FEVENT::ena

pupil size at end

7.5.2.16 float FEVENT::havx

headref averages

7.5.2.17 float FEVENT::havy

headref averages

7.5.2.18 float FEVENT::gavx

gaze averages

7.5.2.19 float FEVENT::gavy

gaze averages

7.5.2.20 float FEVENT::ava

average pupil size

7.5.2.21 float FEVENT::avel

accumulated average velocity

7.5.2.22 float FEVENT::pvel

accumulated peak velocity

7.5.2.23 float FEVENT::svel

start velocity

7.5.2.24 float FEVENT::evel

end velocity

7.5.2.25 float FEVENT::supd_x

start units-per-degree

7.5.2.26 float FEVENT::eupd_x

end units-per-degree

7.5.2.27 float FEVENT::supd_y

start units-per-degree

7.5.2.28 float FEVENT::eupd_y

end units-per-degree

7.5.2.29 INT16 FEVENT::eye

eye: 0=left,1=right

7.5.2.30 UINT16 FEVENT::status

error, warning flags

7.5.2.31 UINT16 FEVENT::flags

error, warning flags

7.5.2.32 UINT16 FEVENT::parsedby

7 bits of flags: PARSEDBY codes

7.5.2.33 LSTRING* FEVENT::message

any message string

7.6 FSAMPLE Struct Reference

Public Attributes

- UINT32 time
- float px [2]
- float py [2]
- float hx [2]
- float hy [2]
- float pa [2]
- float gx [2]
- float gy [2]
- float rx
- float ry
- float gxvel [2]
- float gyvel [2]

- float hxvel [2]
- float hyvel [2]
- float rxvel [2]
- float ryvel [2]
- float fgxvel [2]
- float fgyvel [2]
- float fhxvel [2]
- float fhyvel [2]
- float frxvel [2]
- float fryvel [2]
- INT16 hdata [8]
- UINT16 flags
- UINT16 input
- UINT16 buttons
- INT16 htype
- UINT16 errors

7.6.1 Detailed Description

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.

7.6.2 Member Data Documentation

7.6.2.1 UINT32 FSAMPLE::time

time stamp of sample

7.6.2.2 float FSAMPLE::px[2]

pupil x

7.6.2.3 float FSAMPLE::py[2]

pupil y

7.6.2.4 float FSAMPLE::hx[2]

headref x

7.6.2.5 float FSAMPLE::hy[2]

headref y

7.6.2.6 float FSAMPLE::pa[2]

pupil size or area

7.6.2.7 float FSAMPLE::gx[2]

screen gaze x

7.6.2.8 float FSAMPLE::gy[2]

screen gaze y

7.6.2.9 float FSAMPLE::rx

screen pixels per degree

7.6.2.10 float FSAMPLE::ry

screen pixels per degree

7.6.2.11 float FSAMPLE::gxvel[2]

gaze x velocity

7.6.2.12 float FSAMPLE::gyvel[2]

gaze y velocity

7.6.2.13 float FSAMPLE::hxvel[2]

headref x velocity

7.6.2.14 float FSAMPLE::hyvel[2]

headref y velocity

7.6.2.15 float FSAMPLE::rxvel[2]

raw x velocity

7.6.2.16 float FSAMPLE::ryvel[2]

raw y velocity

7.6.2.17 float FSAMPLE::fgxvel[2]

fast gaze x velocity

7.6.2.18 float FSAMPLE::fgyvel[2]

fast gaze y velocity

7.6.2.19 float FSAMPLE::fhxvel[2]

fast headref x velocity

7.6.2.20 float FSAMPLE::fhyvel[2]

fast headref y velocity

7.6.2.21 float FSAMPLE::frxvel[2]

fast raw x velocity

7.6.2.22 float FSAMPLE::fryvel[2]

fast raw y velocity

7.6.2.23 INT16 FSAMPLE::hdata[8]

head-tracker data (not pre-scaled)

7.6.2.24 UINT16 FSAMPLE::flags

flags to indicate contents

7.6.2.25 UINT16 FSAMPLE::input

extra (input word)

7.6.2.26 UINT16 FSAMPLE::buttons

button state & changes

7.6.2.27 INT16 FSAMPLE::htype

head-tracker data type (0=none)

7.6.2.28 UINT16 FSAMPLE::errors

process error flags

7.7 RECORDINGS Struct Reference

Public Attributes

- UINT32 time
- float sample_rate
- UINT16 eflags
- UINT16 sflags
- byte state
- byte record_type
- byte pupil_type
- byte recording_mode
- byte filter_type
- byte pos_type
- byte eye

7.7.1 Detailed Description

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.

7.7.2 Member Data Documentation

7.7.2.1 UINT32 RECORDINGS::time

start time or end time

7.7.2.2 float RECORDINGS::sample_rate

250 or 500 or 1000

7.7.2.3 UINT16 RECORDINGS::eflags

to hold extra information about events

7.7.2.4 UINT16 RECORDINGS::sflags

to hold extra information about samples

7.7.2.5 byte RECORDINGS::state

0 = END, 1=START

7.7.2.6 byte RECORDINGS::record_type

1 = SAMPLES, 2= EVENTS, 3= SAMPLES and EVENTS

7.7.2.7 byte RECORDINGS::pupil_type

0 = AREA, 1 = DIAMETER

7.7.2.8 byte RECORDINGS::recording_mode

0 = PUPIL, 1 = CR

7.7.2.9 byte RECORDINGS::filter_type

1,2,3

7.7.2.10 byte RECORDINGS::pos_type

0 = GAZE, 1= HREF, 2 = RAW

7.7.2.11 byte RECORDINGS::eye

1=LEFT, 2=RIGHT, 3=LEFT and RIGHT

7.8 TRIAL Struct Reference

Public Attributes

- RECORDINGS * rec
- unsigned int duration
- unsigned int starttime
- unsigned int endtime

7.8.1 Detailed Description

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.

7.8.2 Member Data Documentation

7.8.2.1 RECORDINGS* TRIAL::rec

recording information about the current trial

7.8.2.2 unsigned int TRIAL::duration

duration of the current trial

7.8.2.3 unsigned int TRIAL::starttime

start time of the trial

7.8.2.4 unsigned int TRIAL::endtime

end time of the trial

Index