SR Research LTD

EDF2Numpy Example Documentation

William McEchron
6-10-2024

Copyright (c) 1996-2024, SR Research Ltd., All Rights Reserved

Installation

This example requires that you install the EyeLink Developers Kit for your system. This package will contain the necessary libraries to unpack your EyeLink Data Files (EDFs). You will also need to install NumPy for your given python installation. It is highly recommended that you use an x64 based installation of python for better performance.

Note: This code was written for python 3.9 and testing in python 3.12 but may work in earlier versions of python.

Module: EyeLinkDataImporterExample

EyeLinkData2NumpyArray This code illustrates how to utilize the functions of the EDFACCESSwraper.py to unpack an EyeLink Data File (EDF) into a structured numpy data array. To utilize the code, one must first install the EyeLink Developers Kit: https://www.sr-research.com/support/thread-13.htmland will also need to install numpy for you python environment: https://numpy.org/install/.

Functions

Def main(*inputs*)

This function consumes the name of the EDF file you want to import and the optional input arguments from terminal and passes those arguments to the functions of the EDF2Numpy to process the EDF file. The section of the code that says 'Place your analysis code here' is where you would insert you own code to utilize the imported data.

Inputs

- **EDF_FileName**
- [*Optional arguments*]
    - **output_left_eye**: Left eye data enabled (1) or disabled (0).
    - **output_right_eye**: Right eye data enabled (1) or disabled (0).
    - **messages_enabled**: Message event data enabled (1) or disabled (0).
    - **ioevents_enabled**: Input/Output event data enabled (1) or disabled (0).
    - **recinfo_enabled**: Recording event data enabled (1) or disabled (0).
    - **gaze_data_type**: Sample gaze data type: Raw Data (0); HREF Data (1); Gaze Data (2).
    - **text_data_type**: any standard encoding: Standard Encodings
    - **output_data_ppd**: Pixel Per Degree of visual angle data enabled (1) or disabled (0).
    - **output_data_velocity**: Sample velocity data enabled (1) or disabled (0).
    - **output_data_pupilsize**: Pupil size data enabled (1) or disabled (0).
    - **output_data_debugflags**: output debugging flags and create a .debug output file enabled (1) or disabled (0).

- **output_dataviewer_commands**: 0: Mask DV commands from output; 1: Include DV commands in output.
- **enable_consistency_check**: 0: consistency check disabled; 1: enable consistency check and report; 2: enable consistency check and fix.
- **enable_failsafe**: 0: fail-safe mode disabled; 1: fail-safe enabled.
- **disable_large_timestamp_check**: 0: timestamp check enabled; 1: disable timestamp check flag.
- **events_enabled**: 0: Event data disabled; 1: Event Data Enabled.
- **output_eventtype_start**: 0: Start Events data disabled; 1: Start Events Data Enabled.
- **output_eventtype_end**: 0: End Events data disabled; 1: End Events Data Enabled.
- **output_eventtype_saccade**: 0: Saccade Events data disabled; 1: Saccade Events Data Enabled.
- **output_eventtype_fixation**: 0: Fixation Events data disabled;1: Fixation Events Data Enabled.
- **output_eventtype_fixupdate**: 0: FixUpdate Events data disabled;1: FixUpdate Events Data Enabled.
- **output_eventtype_blink**: 0: Blink Events data disabled; 1: Blink Events Data Enabled.
- **output_eventdata_parse**: 0: Parse Event Data disabled; 1: Parse Event Data enabled.
- **output_eventdata_averages**: 0: End Events data disabled; 1: End Events Data Enabled.
- **msg_offset_enabled**: 0: no integer offset applied; 1: integer offset applied to message events.
- **samples_enabled**: 0: Sample data disabled; 1: Sample Data Enabled.
- output_headtargetdata_enabled: 0: headTarget data disabled; 1: headTarget Data Enabled.
- **output_samplevel_model_type**: 0: Standard model; 1: Fast model.
- **output_sample_start_enabled**: 0: Start Sample data disabled; 1: Start Sample Data Enabled.
- **output_sample_end_enabled**: 0: End Sample data disabled; 1: End Sample Data Enabled.
- **trial_parse_start**: the string used to mark the start of the trial.
- **trial_parse_end**: the string used to mark the end of the trial.

Return

Returns 0 if the operation is successful.

Module: EDF2numpy

EDF2numpy This code illustrates how to utilize the functions of the EDFACCESSwraper.py to unpack an EyeLink Data File (EDF) into a structured numpy data array. To utilize the code, one must first install the EyeLink Developers Kit: https://www.sr-research.com/support/thread-13.html and will also need to install numpy for you python environment: https://numpy.org/install/.

Classes

Var EVENTdata

A structured numpy array that stores the information about Event data (Fixations, Saccades, Blinks, etc...) in the data set.

Fields

- **time**: The timestamp of when the event was generated.
- **eventType**: The type of event generated: StartBlink, EndBlink, StartSacc, EndSacc, StartFix, EndFix, FixUpdate, or MSG.
- **eyeTracked**: The eye that generated the event: Left, Right, or Binocular.
- **gazeType**: The gaze data type that generated the event: HREF or GAZE.
- **startTime**: The start time of the event.
- **startPosX**: The X gaze position at the start of the event: HREF or GAZE.
- **startPosY**: The Y gaze position at the start of the event: HREF or GAZE.
- **StartPupilSize**: The pupil size at the start of the event in arbitrary camera pixel units.
- **startVEL**: The gaze velocity at start of event type in degrees per second.
- **startPPDX**: The X pixels per degree at start of event type.
- **startPPDY**: The Y pixels per degree at start of event type.
- **endTime**: The end time of the event.
- **duration**: The total duration of event (endTime-startTime).
- **endPosX**: The X gaze position at the end of the event: HREF or GAZE.
- **endPosY**: The X gaze position at the end of the event: HREF or GAZE.
- **endPupilSize**: The pupil size at the end of the event in arbitrary camera pixel units.
- **endVEL**: The gaze velocity at end of event type in degrees per second.
- **endPPDX**: The X pixels per degree at end of event type.
- **endPPDY**: The Y pixels per degree at end of event type.
- **avgPosX**: The average X position for duration of event type: GAZE or HREF.
- **avgPosY**: The average Y position for duration of event type: GAZE or HREF.
- **avgPupilSize**: The average pupil size for duration of event type: GAZE or HREF.
- **avgVEL**: The average velocity for duration of event type.
- **peakVEL**: The peak velocity for duration of event type.
- **message**: String of message data received during event.
- **readFlags**: Flags generated from reading the message.
- **flags**: Flags generated from processing the event.
- **parsedby**: Which type of data was used for parsing events: RAW, HREF, or GAZE.
- **status**: The status of the event.

- **elementIndex**: The index of the data in EDF buffer.
- **eventIndex**: The index of the event in the EVENTdata structure.

Var HEADERdata

A string variable that stores the header data from the EDF file.

Var IOEVENTdata

A structured numpy array that stores the information about Input/Output events collected during the recording.

- **ioEventType**: The input/output event type (Button or INPUT).
- **time**: The timestamp of the input/output event in milliseconds.
- **IOData**: The data from IO event.
- **iotype**: The data type code for the event.
- **elementIndex**: The index of the data in EDF buffer.
- **ioEventIndex**: The index of the sample in the SAMPLEdata structure.

Var MESSAGEdata

A structured numpy array that stores the information about message events received in the EDF.

- **time**: The timestamp of when the message was received in milliseconds.
- **message**: The string data of the message event.
- **TimingCorrected**: If the message starts with an integer value and self.options['msg_offset_enabled'] is set to 1 then the integer value will be subtracted from the time and this value will be set to TRUE to reflect that an offset was applied to that message.
- **messageLength**: The length of the string contained in the message event.
- **readFlags**: Flags generated from reading the message.
- **flags**: Flags generated from processing the event.
- **parsedby**: Which type of data was used for parsing events: RAW, HREF, GAZE.
- **status**: The status of the event.
- **elementIndex**: The index of the data in EDF buffer.
- **msgIndex**: The index of the message event in the MESSAGEdata structure.

var RECORDINGdata

A structured numpy array that stores the information about the configuration of each recording in the data set.

- **samplingRate**: The Sampling Rate of the recording in Hertz.
- eyeTracked: Which type of Eye Data was recorded: Left, Right, or Binocular.
- **pupilDataType**: Which pupil size data type was recorded: Area or Diameter.
- **trackerState**: The state of the recording event: Start or End.
- **recordType**: Which data streams were recorded: Samples,Events, or Samples & Events.

- **parsedbyType**: Which type of data used for parsing events: RAW, HREF, GAZE.
- **filterType**: Which file Sample filter was used: Off,Standard or Extra.
- **recordingMode**: Which objects were used for gaze data: Pupil Only or Pupil-CR.
- **startflags**: Flags generated for start events.
- **endflags**: Flags generated for end events.
- **elementIndex**: The index of the data in EDF buffer.
- **recordingIndex**: The index of the recording in the RECORDINGdata structure.

Var SAMPLEdata

A structured numpy array that stores the information about each gaze sample collected during the recording.

- **time**: The timestamp of when the event was generated.
- **posXLeft**: The left_eye X gaze position: RAW, HREF, or GAZE.
- **posYLeft**: The Y gaze position: RAW, HREF, or GAZE.
- **pupilSizeLeft**: The left_eye pupil Area or Diameter in arbitrary camera pixel units.
- **posXRight**: The right_eye X gaze position: RAW, HREF, or GAZE.
- **posYRight**: The right_eye Y gaze position: RAW, HREF, or GAZE.
- **pupilSizeRight**: The right_eye pupil Area or Diameter in arbitrary camera pixel units.
- **PpdX**: The X pixels per degree.
- **PpdY**: The Y pixels per degree.
- **velXLeft**: The left_eye X velocity in degrees per second: standard velocity model or fast velocity model.
- **velYLeft**: The left_eye Y velocity in degrees per second: standard velocity model or fast velocity model.
- **velXRight**: The right_eye X velocity in degrees per second: standard velocity model or fast velocity model.
- **velYRight**: The right_eye Y velocity in degrees per second: standard velocity model or fast velocity model.
- **headTrackerType**: The data type of the head target data.
- **headTargetDataX**: The X position of the Head target sticker.
- **headTargetDataY**: The Y position of the Head target sticker.
- **headTargetDataZ**: The Z position of the Head target sticker.
- **headTargetDataFlags**: Flags generated from processing the head target data.
- **inputPortData**: Status of the input port.
- **buttonData**: State of the Button definitions.
- **flags**: Flags generated from processing the sample.
- **errors**: Error Flags generated from processing the sample.
- **elementIndex**: The index of the data in EDF buffer.
- **sampleIndex**: The index of the sample in the SAMPLEdata structure.

Var options

This dictionary contains a number of optional flags which will govern how the EDF data is processed. Most fields are binary with a value of 1 representing "Enabled" and 0 representing "Disabled." Integer inputs are noted in parentheses.

- **output_left_eye**: Left eye data enabled (1) or disabled (0).
- **output_right_eye**: Right eye data enabled (1) or disabled (0).
- **messages_enabled**: Message event data enabled (1) or disabled (0).
- **ioevents_enabled**: Input/Output event data enabled (1) or disabled (0).
- **recinfo_enabled**: Recording event data enabled (1) or disabled (0).
- **gaze_data_type**: Sample gaze data type: Raw Data (0); HREF Data (1); Gaze Data (2).
- **text_data_type**: any standard encoding: Standard Encodings.
- **output_data_ppd**: Pixel Per Degree of visual angle data enabled (1) or disabled (0).
- **output_data_velocity**: Sample velocity data enabled (1) or disabled (0).
- **output_data_pupilsize**: Pupil size data enabled (1) or disabled (0).
- **output_data_debugflags**: debugging flags and .debug output file enabled (1) or disabled (0).
- **output_dataviewer_commands**: 0: Mask DV commands from output; 1: Include DV commands in output.
- **enable_consistency_check**: 0: consistency check disabled; 1: enable consistency check and report; 2: enable consistency check and fix.
- **enable_failsafe**: 0: fail-safe mode disabled; 1: fail-safe enabled.
- **disable_large_timestamp_check**: 0: timestamp check enabled; 1: disable timestamp check flag.
- **events_enabled**: 0: Event data disabled; 1: Event Data Enabled.
- **output_eventtype_start**: 0: Start Events data disabled; 1: Start Events Data Enabled.
- **output_eventtype_end**: 0: End Events data disabled; 1: End Events Data Enabled.
- **output_eventtype_saccade**: 0: Saccade Events data disabled; 1: Saccade Events Data Enabled.
- **output_eventtype_fixation**: 0: Fixation Events data disabled;1: Fixation Events Data Enabled.
- **output_eventtype_fixupdate**: 0: FixUpdate Events data disabled;1: FixUpdate Events Data Enabled.
- **output_eventtype_blink**: 0: Blink Events data disabled; 1: Blink Events Data Enabled.
- **output_eventdata_parse**: 0: Parse Event Data disabled; 1: Parse Event Data enabled.
- **output_eventdata_averages**: 0: End Events data disabled; 1: End Events Data Enabled.
- **msg_offset_enabled**: 0: no integer offset applied; 1: integer offset applied to message events.
- **samples_enabled**: 0: Sample data disabled; 1: Sample Data Enabled.
- **output_headtargetdata_enabled**: 0: headTarget data disabled; 1: headTarget Data Enabled.
- **output_samplevel_model_type**: 0: Standard model; 1: Fast model.
- **output_sample_start_enabled**: 0: Start Sample data disabled; 1: Start Sample Data Enabled.

- **output_sample_end_enabled**: 0: End Sample data disabled; 1: End Sample Data Enabled.
- **trial_parse_start**: the string used to mark the start of the trial.
- **trial_parse_end**: the string used to mark the end of the trial.

Methods

Def appendDebugFile (fileHandle, data)

Appends new line to debug file.

Parameters

fileHandle: The handle of the debug file created by openDebugFile() data: The line you would like appended to the data file.

Return

Returns 0 if the operation is successful.

Def appendIOEvent (Data, index)

Updates event event data in the IOData structure.

Parameters

data: The event data from edf_get_float_data() that you would like appended to the IOData structure. index: The index of the IOData structure that you want to overwrite with the data values.

Return

Returns 0 if the operation is successful.

Def appendMessage (Data, index)

Updates event event data in the MESSAGEdata structure.

Parameters

data: The event data from edf_get_float_data() that you would like appended to the MESSAGEdata structure. index: The index of the MESSAGEdata structure that you want to overwrite with the data values.

Return

Returns 0 if the operation is successful.

Def appendRecording (Data, index)

Updates event event data in the RECORDINGdata structure.

Parameters

data: The event data from edf_get_float_data() that you would like appended to the RECORDINGdata structure. index: The index of the RECORDINGdata structure that you want to overwrite with the data values.

Return

Returns 0 if the operation is successful.

Def closeDebugFile (fileHandle)

Closed debug file.

Parameters

fileHandle: The handle of the debug file created by openDebugFile().

Return

Returns 0 if the operation is successful.

Def closeEDF (edfHandle)

Close out EDF file, and the debug file if one exists.

Parameters

edfHandle: the pointer to the EDF file created by openEDF.

Return

Returns 0 if the operation is successful.

Def combineConsistencyArgs ()

Combine different consistency flags contained in .options into one binary flag as required for openEDF().

Return

binary output of consistency flags.

Def consumeInputArgs (inputArgs)

Parse input arguments and reject bad value assignments.

Parameters

inputArgs: A list or string of input arguments to be updated in .options dictionary.

Return

Returns 0 if the operation is successful.

Def openDebugFile (Outputfilename)

Opens debug file.

Parameters

outputfilename: the name of the debug file.

Return

Returns handle of debug file.

Def openEDF (edfFilename)

Opens EDF file and returns buffer of data/contents.

Parameters

edfFilename: The path or filename of the EDF you want to open.

Return

EDFData: the handle for the EDF data contents.

Def prealocateArraySize (edfFilename)

Resize the data arrays to close to their expected size for better memory management. Note: may over-provision so make sure to trim the arrays afterwards.

Parameters

edfFilename: the handle to the EDF data contents generated by openEDF().

Return

Returns 0 if the operation is successful.

Def readEDF (edfFilename)

Read in and parse EDF file into data structures Note: Make sure to consume any input arguments before running this function to make sure .options is updated.

Parameters

edfFilename: the path/filename of the EDF you want to extract the contents of.

Return

Returns a Numpy array of structured numpy arrays: [HEADERdata,RECORDINGdata,MESSAGEdata,SAMPLEdata,EVENTdata,IOEVENTdata].

Def trimArray ()

Remove any empty rows from the data arrays to cut out the fat.

Return

Returns 0 if the operation is successful.

Def updateEvent (Data, eventtype, index)

Updates event event data in the EVENTdata structure.

Parameters

Data: The event data from edf_get_float_data() that you would like appended to the EVENTdata structure. eventtype: The event type code (SFIX,EFIX,FIXUPDATE,SSACC,ESACC,SBLINK,EBLINK) index: The index of the EVENTdata structure that you want to overwrite with the data values.

Return

Returns 0 if the operation is successful.

Def updateSample (Data, index)

Updates sample data in the SAMPLEdata structure.

Parameters

Data: The sample data from edf_get_float_data() that you would like appended to the SAMPLEdata structure. index: The index of the SAMPLEdata structure that you want to overwrite with the data values.

Return

Returns 0 if the operation is successful.

Module: EDFACCESSwrapper

EDFACCESSwrapper This code wraps the functions and structures defined in the EDFACCESS API C-based DLL into a python format. To utilize the code one must first install the EyeLink Developers Kit: https://www.sr-research.com/support/thread-13.html Once installed full documentation of the EDFACCESS API functions and structures can be found in the EDF Access C API user manual.pdf packaged with the API.

Classes

Class ALLF_DATA

A union structure for linking multiple data types. This is used to capture data from edf_get_float_data which can return multiple data types depending on where you are in the file.

- **FEVENT** - The event structure.
- **IMESSAGE** - The message structure.
- **IOEVENT** - The Input/output event structure.
- **FSAMPLE** - The sample structure.
- **RECORDINGS** - The recording structure.

Class FEVENT

A structure for storing the EDFaccess API's FEVENT structure this structure stored data for parser events(fixations, saccades, blinks, etc...).

- **time** - The timestamp of the Event.
- **etype** - The type of the event (SFIX, EFIX, SSACC, etc...).
- **read** - The flags of which items were included.
- **sttime** - The start time of the event.
- **entime** - The end time of the event.
- **hstx** - The HREF X start position.
- **hsty** - The HREF Y start position.
- **gstx** - The GAZE X start position.
- **gsty** - The GAZE Y start position.
- **sta** - The pupil size at the start of the event.
- **henx** - The HREF X end position.
- **heny** - The HREF Y end position.
- **genx** - The GAZE X end position.
- **geny** - The GAZE Y end position.
- **ena** - The pupil size at the end of the event.
- **havx** - The average HREF X gaze position.
- **havy** - The average HREF Y gaze position.
- **gavx** - The average GAZE X gaze position.
- **gavy** - The average GAZE Y gaze position.
- **ava** - The average pupil size during the event.

- **avel** - The average velocity during the event.
- **pvel** - The peak velocity during the event.
- **svel** - The velocity at the start of the event.
- **evel** - The velocity at the end of the event.
- **supd_x** - The pixels per degree at the start of the event.
- **eupd_x** - The pixels per degree at the end of the event.
- **eye** - The eye that generated the event.
- **status** - The status flags for the event.
- **flags** - The flags generated from processing the event.
- **input** - The state of the input port.
- **buttons** - The state of the button definitions.
- **parsedby** - The type of parser used to generate the event.
- **message** - The string data from a message event.

Class FSAMPLE

A structure for storing the EDFaccess API's FSAMPLE structure.

- **time** - The timestamp of the sample
- **px** - The RAW X gaze position
- **py** - The RAW Y gaze position.
- **hx** - The HREF X gaze position.
- **hY** - The HREF Y gaze position.
- **pa** - The pupil size.
- **gx** - The GAZE X position.
- **gy** - The GAZE Y position.
- **rx** - The pixels per degree X.
- **ry** - The pixels per degree Y.
- **gxvel** - The GAZE X velocity using the standard model.
- **gyvel** - The GAZE Y velocity using the standard model.
- **hxvel** - The HREF X velocity using the standard model.
- **hyvel** - The HREF Y velocity using the standard model.
- **rxvel** - The RAW X velocity using the standard model.
- **ryvel** - The RAW Y velocity using the standard model.
- **fgxvel** - The GAZE X velocity using the fast model.
- **fgyvel** - The GAZE Y velocity using the fast model.
- **fhxvel** - The HREF X velocity using the fast model.
- **fhyvel** - The HREF Y velocity using the fast model.
- **frxvel** - The RAW X velocity using the fast model.
- **fryvel** - The RAW Y velocity using the fast model.
- **hdata** - The head target data.
- **flags** - The flags from processing the sample.
- **inputs** - The state of the input port.
- **buttons** - The state of the button definitions.
- **htype** - The head target data type.
- **errors** - Errors generated from reading the samples.

Class IMESSAGE

A structure for storing the EDFaccess API's IMESSAGE Structure.

- **time** - The timestamp of the message event.
- **mtype** - The type of message.
- **length** - The length of the message string.
- **text** - The string data from the message.

Class IOEVENT

A structure for storing the EDFaccess API's IOEVENT Structure to store Input/Output events (input and button events).

- **time** - The timestamp of the input/output event.
- **itype** - The event type.
- **data** - Data from that event.

Class RECORDINGS

A structure for storing the EDFaccess API's RECORDINGS Structure This data is auto-populated each time a recording begins and ends.

- **time** - Timestamp of the recording event.
- **float sample_rate** - the sampling rate of the recording.
- **eflags** - end event processing flags.
- **sflags** - start event flags.
- **state** - the state of the recording (end or start).
- **record_type** - samples, events, or samples and events.
- **pupil_type** - Area or Diameter.
- **recording_mode** - Pupil only or Pupil-CR.
- **filter_type** - off, standard, or high sensitivity.
- **posType** - Raw, HREF, or GAZE.
- **eye** - Left, Right, or Binocular.

Class GAZEDATA

A structure for storing Left and Right eye data.

- **left** - gaze data for left eye.
- **right** - gaze data for right eye.

Class HDATA

A structure for storing head target data for remote mode sessions note some fields are not used but left for expansion purposes.

- **targetX** - the X position of the head target sticker.
- **targetY** - the Y position of the head target sticker.
- **targetDist** - the Z distance of the head target sticker.

- **targetFlags** - flags processing the head target data.
- **hdata5** - reserved for future use.
- **hdata6** - reserved for future use.
- **hdata7** - reserved for future use.
- **hdata8** - reserved for future use.

Class LSTRING

A structure for storing packed message data from message events.

- **length** - the length of the string
- **text** - a char array of the message text

Class BOOKMARK

A structure for storing bookmark data.

- **id** - bookmark ID value.

Methods

Def checkAPI()

Attempt to find API files and return correct location if found.

Return

EDFAPI.DLL location if found or 0 if not found.

Def edf_close_file(edfData)

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

- edfData: a valid pointer to EDFFILE structure. This should be created by calling edf_open_file ().

Returns

- Returns 0 if the operation is successful.

Def edf_free_bookmark(, edfData, bookmark)

Removes an existing bookmark

Parameters

- edfData: a valid pointer to EDFFILE structure. This should be created by calling edf_open_file.
- bookmark: 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 Bookmark should be initialized there.

Returns

- Returns 0 if the operation is successful.

Def edf_get_element_count(, edfData)

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

Parameters

- edfData: a valid pointer to EDFFILE structure. This should be created by calling edf_open_file.

Returns

- The number of elements in the EDF file

Def edf_get_end_trial_identifier(, edfData)

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

Parameters

- edfData: 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.

Def edf_get_float_data(, edfData)

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

- edfData: 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().

Def edf_get_next_data(, edfData)

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 when 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

- edfData: a valid pointer to EDFFILE structure. This handle should be created by calling edf_open_file().

Returns

- 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 recording info.
- NO_PENDING_ITEMS: no more data left.

Def edf_get_preamble_text(, edfData, 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

- edfData: a valid pointer to EDFFILE structure. This handle should be created by calling edf_open_file().
- length: length of the buffer.

Returns

- Returns 0 if the operation is successful.

Def edf_get_preamble_text_length(, edfData)

Returns the length of the preamble text.

Parameters

- edfData: 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.

Def edf_get_start_trial_identifier(, edfData)

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

Parameters

- edfData: 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.

Def edf_get_trial_count(, edfData)

Returns the number of trials in the EDF file.

Parameters

- edfData: 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.

Def edf_get_trial_header(, edfData, trial)

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

Parameters

- edfData: 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

- Returns 0 if the operation is successful.

Def edf_goto_bookmark(, edfData, bookmark)

Jumps to the given bookmark.

Parameters

- edfData: a valid pointer to EDFFILE structure. This should be created by calling edf_open_file.
- bookmark: 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 Bookmark should be initialized there.

Returns

- Returns 0 if the operation is successful.

Def edf_goto_next_trial(, edfData)

Jumps to the beginning of the next trial.

Parameters

- edfData: a valid pointer to EDFFILE structure. This should be created by calling edf_open_file().

Returns

- Returns 0 if the operation is successful.

Def edf_goto_previous_trial(, edfData)

Jumps to the beginning of the previous trial.

Parameters

- edfData: a valid pointer to EDFFILE structure. This should be created by calling edf_open_file().

Returns

- Returns 0 if the operation is successful.

Def edf_goto_trial_with_end_time(, edfData, end_time)

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

Parameters

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

Returns

- Returns 0 if the operation is successful.

defedf_goto_trial_with_start_time(, edfData, start_time)

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

Parameters

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

Returns

- Returns 0 if the operation is successful.

Def edf_jump_to_trial(, edfData, trial)

Jumps to the beginning of a given trial.

Parameters

- edfData: 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

- Returns 0 if the operation is successful.

Def edf_open_file(, edfFilename, consistency, loadevents, loadsamples)

Opens the EDF file passed in by edf_file_name and pre-processes the EDF file.

Parameters

- edfFilename: name of the EDF file to be opened.
- consistency: consistency 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.

Returns

- If successful a pointer to EDFFILE structure is returned. Otherwise, NULL is returned.

Def edf_set_bookmark(, edfData, bookmark)

Bookmark the current position of the edf file.

Parameters

- edfData: a valid pointer to EDFFILE structure. This should be created by calling edf_open_file.
- bookmark: a valid pointer to a valid BOOKMARK structure. This structure will be filled by this function. Bookmark should be initialized before being used by this function.

Returns

- Returns 0 if the operation is successful.

Def edf_set_trial_identifier(, edfData, start_marker_string, 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 that is considered start or end of the trial.

Parameters

- edfData: 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

- Returns 0 if the operation is successful.

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, the 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.

Def loadAPI()

Attempt to load the CDLL from the default EyeLink Developers Kit location.

Returns

- Returns 0 if the operation is successful.