Implementation of eye tracking functions in the Presentation interface for the EyeLink-II eye tracking system, Version 0.9-Beta

Missing functions and remarks are shown in red.

`new eye_tracker( string object_id ) : eye_tracker`

Creates a new eye_tracker object and returns a reference to it. "object_id" can be either the GUID of the object to be used (in the text format "{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}") or the friendly name given to the eye tracker extension when it was registered with Presentation.

`set_max_buffer_size( int data_type, int size ) : void`

Sets the maximum number of data samples of the specified data type that will be buffered in Presentation when eye_tracker::start_data(int, bool) is called with a store_data value of true or eye_tracker::start_data(int) is called. The maximum buffer size for all types is by default set to 100. The "size" argument must be between 1 and 100,000. The buffers are circular, meaning that when a buffer is filled new data begins overwriting the oldest data. Calling set_max_buffer_size clears the buffer of the specified type. Use one of the following pre-defined values for data_type: dt_position, dt_saccade, dt_fixation, dt_aoi, dt_pupil, dt_blink

`start_data( int data_type, bool store_data ) : void`

Instructs the eye tracker to start sending data of the specified type. If "store_data" is false, the eye_tracker object will only store the most recent data sample. Use the "new_xxx_data" methods to find out how many samples have been received since the buffer was last cleared. If "store_data" is true, the eye_tracker object will store incoming data in a buffer. Access this data using "get_xxx_data" methods. Presentation is not meant to be a data acquisition system. Do not store large volumes of position or pupil data as this data is stored in RAM. Any data of the specified type collected from a previous call to "start_data" will be erased. Use one of the following pre-defined values for data_type: dt_position, dt_pupil

`start_data( int data_type ) : void`

Instructs the eye tracker to start sending data of the specified type. Use the "new_xxx_data" methods to find out how many samples have been received since the buffer was last cleared. The eye_tracker object will store incoming data in a buffer. Access this data using "get_xxx_data" methods. Any data of the specified type collected from a previous call to "start_data" will be erased. Use one of the following pre-defined values for data_type: dt_saccade, dt_fixation, dt_aoi, dt_blink

`stop_data(int data_type) : void`

Instructs the eye tracker to stop sending data of the specified type. Use one of the following pre-defined values for data_type: dt_position, dt_saccade, dt_fixation, dt_aoi, dt_pupil, dt_blink

`clear_buffer(int data_type) : void`

Instructs Presentation to clear any stored data of the specified type. Use one of the following pre-defined values for data_type: dt_position, dt_saccade, dt_fixation, dt_aoi, dt_pupil, dt_blink.

`event_count(int data_type) : int`

Returns the number of events of the specified type that have been received since the buffer for the specified type was last cleared. The buffer for a given data type is cleared explicitly by a call to eye_tracker::clear_buffer or implicitly by calls to eye_tracker::start_data or eye_tracker::set_max_bufer_size. For the pupil_data and position_data event types, if start_data(int,int) is called with a "store_data" value of false, the return value of this method will still increase as each new event is received. However, since Presentation is only storing the most recent data sample, the buffer will always cantain only one item. Use one of the following pre-defined values for data_type: dt_position, dt_saccade, dt_fixation, dt_aoi, dt_pupil, dt_blink.

`buffer_position(int data_type) : int`

Returns the index of the last buffer slot to be filled with data of the specified type. If data of the specified types has been received, values will be between 1 and the maximum buffer size for the specified type. If no data has been received since the last call to start_data, clear_buffer, or set_max_buffer_size, the return value will be 0. Buffer sizes for all types default to 100 unless explicitly set by a call to eye_tracker::set_max_buffer_size. Use one of the following pre-defined values for data_type: dt_position, dt_saccade, dt_fixation, dt_aoi, dt_pupil, dt_blink.

`get_position_data( int index ) : eye_position_data`

Returns an eye_position_data with index "index" in the buffer of stored position data. If start_data was called with a "store_data" argument of false, the buffer will contain a maximum of 1 item. The index of the first item is 1.

`last_position_data() : eye_position_data`

Returns an eye_position_data reference containing the most recent eye position data received. It is an error to call this method if no data is available.

`new_position_data() : int`

Returns the number of new position data has been received since the last call to this method. Calling start_data, clear_buffer or set_max_buffer_size for this data type resets this value to 0. The value returned by this method will increase as each new data sample is received, even if start_data(int,int) is called with a "store_data" argument value of false. If Presentation is not buffering multiple position data samples, the buffer will still only contain the most recent sample.

`get_trigger() : int`

Returns a numerical code received from the eye tracker. The return value of trigger_count must be > 0 before calling this method. get_trigger returns codes in the order in which they were received.

Triggers not yet implemented.

`send_trigger(int code) : void`

Sends a trigger code to the eye tracker hardware. The effect is device dependent.

Triggers not yet implemented.

`trigger_count() : int`

Returns the number of numerical codes received from the eye tracker that are ready to be retrieved. Use the "get_trigger" method retrieve the codes. They must be read one by one.

Triggers not yet implemented.

`send_string(string message) : void`

Sends a string message to the eye tracker hardware; like eyemsg_printf().

`send_command(string message) : int`

Sends a command message to the eye tracker hardware. Returns a response code from the eye tracker; like eyecmd_printf().

`set_recording(bool recording_on) : void`

Instructs the eye tracker to record data (on the eye tracking computer) whenever it it tracking eye position. The effect is device dependent.

`is_recording() : bool`

Indicates whether or not the eye tracker is storing data on the eye tracker computer. This is independent of whether or not data is being sent to Presentation.

`start_tracking() : void`

Instructs the eye tracker to start tracking eye position data. This is independent of whether or not the data is being sent to Presentation.

`stop_tracking() : void`

Instructs the eye tracker to stop tracking eye position data.

`get_status() : int`

Retrieves a code that represents the status of the eye tracker hardware. The returned value will be equal to one of the following pre-defined PCL values: et_status_stopped, et_status_initializing, et_status_tracking, et_status_tracking_left, et_status_tracking_right. The meaning of these states is device-dependent. Check your vendor's documentation for details.

`supports( int feature_code) : bool`

Returns a value specifying whether or not the eye tracker extension supports a given feature. The available features to query are provided as the following PCL pre-defined values: et_feature_position, et_feature_pupil_diameter, et_feature_fixation, et_feature_saccade, et_feature_aoi, et_feature_blink, et_feature_multiple_eyes

`calibrate(int calibration_type, double parameter1, double parameter2, double parameter3) : void`

Instructs the eye tracker to perform a calibration operation. calibration_type can be one of the predefined variables et_calibrate_default or et_calibrate_drift_correct, or an integer provided by the eye tracker manufacturer to designate a vendor-specific calibration routine. The additional parameters are eye tracker specific. See your eye tracker documentation for the use, if any, of these parameters. This method does not return until the calibration procedure has completed. If you define a picture stimulus named "et_calibration", this picture stimulus will be used for the calibration display. The eye tracker extension controls the positioning of the picture parts in this stimulus during calibration. If you don't define a picture stimulus named "et_calibration", Presentation will use a default white cross on a black background. The eye tracker extension can optionally use its own graphic for calibration, overriding the picture parts you put in your "et_calibration" picture.

Currently, et_calibration_default switches the EyeLink system to setup mode; i.e. the experimenter can setup the cameras, set options, and do the calibration and validation procedures from the Operator PC.

Also, et_calibrate_drift_correct is implemented and triggers the usual drift correction procedure with one central target.

Parameters 1 to 3 have no effect yet.

`new_saccade_events() : int`

Returns the number of new saccade events that have been received since the last call to this method. Calling start_data, clear_buffer or set_max_buffer_size for this data type resets this value to 0.

`get_saccade_event( int index ) : saccade_event_data`

Returns saccade_event_data with index "index" in the buffer of stored saccade data. Call event_count to find out how many items have been received since the last call to start_data, clear_buffer or set_max_buffer_size for this data type. The index of the first item is 1.

`last_saccade_event() : saccade_event_data`

Returns a saccade_event_data reference containing the most recent saccade event received. It is an error to call this method if no data is available.

`new_fixation_events() : int`

Returns the number of new fixation events that have been received since the last call to this method. Calling start_data, clear_buffer or set_max_buffer_size for this data type resets this value to 0.

`get_fixation_event( int index ) : fixation_event_data`

Returns fixation_event_data with index "index" in the buffer of stored fixation event data. Call event_count to find out how many items have been received since the last call to start_data, clear_buffer or set_max_buffer_size for this data type. The index of the first item is 1.

`last_fixation_event() : fixation_event_data`

Returns a fixation_event_data reference containing the most recent fixation event data received. It is an error to call this method if no data is available.

`new_aoi_events() : int`

Returns the number of new aoi events that have been received since the last call to this method. Calling start_data, clear_buffer or set_max_buffer_size for this data type resets this value to 0.

AOIs not yet implemented.

`get_aoi_event( int index ) : aoi_event_data`

Returns aoi_event_data with index "index" in the buffer of stored aoi event data. Call event_count to find out how many items have been received since the last call to start_data, clear_buffer or set_max_buffer_size for this data type. The index of the first item is 1.

AOIs not yet implemented.

`last_aoi_event() : aoi_event_data`

Returns an aoi_event_data reference containing the most recent aoi event data received. It is an error to call this method if no data is available.

AOIs not yet implemented.

`set_aoi_set(int aoi_set) : void`

Sets the AOI set for which aoi_event_data will be returned by calls to the aoi methods. AOI sets are groups of AOI regions that are defined before a trial using your eye tracker vendor's software.

AOIs not yet implemented.

`new_pupil_data() : int`

Returns the number of new pupil data have been received since the last call to this method. Calling start_data, clear_buffer or set_max_buffer_size for this data type resets this value to 0. The value returned by this method will increase as each new data sample is received, even if start_data(int,int) is called with a "store_data" argument value of false. If Presentation is not buffering multiple position data samples, the buffer will still only contain the most recent sample.

`get_pupil_data( int index ) : pupil_data`

Returns pupil_data with index "index" in the buffer of stored pupil data. Call event_count to find out how many items have been received since the last call to start_data, clear_buffer or set_max_buffer_size for this data type. If start_data was called with a "store_data" argument of false, the buffer will contain a maximum of 1 item. The index of the first item is 1.

`last_pupil_data() : pupil_data`

Returns a pupil_data reference containing the most recent eye position data received. It is an error to call this method if no data is available.

`new_blink_events() : int`

Returns the number of new blink events that have been received since the last call to this method. Calling start_data, clear_buffer or set_max_buffer_size for this data type resets this value to 0.

`get_blink_event( int index ) : blink_event_data`

Returns blink_event_data with index "index" in the buffer of stored blink event data. Call event_count to find out how many items have been received since the last call to start_data, clear_buffer or set_max_buffer_size for this data type. The index of the first item is 1.

`last_blink_event() : blink_event_data`

Returns a blink_event_data reference containing the most recent eye position data received. It is an error to call this method if no data is available.