LiveTrack Library

API for C, MATLAB, Python

(M0203.1, M0209.1, M0204, M0206)

Version 1

Unit 78-80 Riverside, Sir Thomas Longley Road
Rochester, Kent, ME2 4BH
United Kingdom

This page intentionally left blank

Title: **LiveTrack Library Guide**
Document Number: **CR-000289-MS**
Version: **1**
Last issuer: **Robert Lee**
Issue date: **2021-03-08**

Contents

1 Introduction

1.1 Document owner

Job Title
Staff scientists

1.2 Referenced documents

Reference	Title	Document Number
1.	LiveTrack Library Guide	CR-000289-MS
2.	LiveTrack AV for fMRI (Presto) Installation and User Guide	CR-000288-MS
3.	LiveTrack FM (M0206) User Manual	CR-000131-MS

1.3 Scope

Covers the LiveTrack C API, LiveTrack Toolbox for MATLAB and LiveTrack bindings for Python, distributed with and used with the LiveTrack family of eyetracker products.

1.4 Purpose

Describes the functions contained in the APIs, for use by customers when integrating the LiveTrack into their own experiment code. It is intended to complement the user guide appropriate to the specific LiveTrack product that you are using.

2 C API

2.1 SDK for Windows

The LiveTrack SDK is designed to integrate LiveTrack Eye Tracker with a custom application. The SDK supports C/C++ and any other language that supports Windows DLL libraries. The SDK includes LIB files to allow the library to link with Microsoft Visual C and C++ Builder applications which link with the libLiveTrack.dll library. This library and its dependencies will be present on your system after installing the LiveTrack Toolbox (which also installs the LiveTrack Viewer) and are contained in the LiveTrack Viewer installation folder which is typically:
C:\Program Files\Cambridge Research Systems\LiveTrack Viewer
Or for 32 bit installs on 64-bit Windows:
C:\Program Files (x86)\Cambridge Research Systems\LiveTrack Viewer x86
To satisfy dependency requirements, the Windows environment path is configured to include the above path.
NOTE: Distributions of custom applications based on the LiveTrack SDK will require a LiveTrack Viewer installation or the contents of the LiveTrack Viewer application path.

2.1.1 32/64 Bit Support

The LiveTrack Viewer application is supplied in both 32 and 64-bit architectures; both of which are supported on 64-bit Windows operating systems and can be installed side by side. Note that the SDK architecture (32/64 bit) will depend on the Windows environment path. This will be configured by the LiveTrack Viewer installer. Therefore, the SDK architecture depends on the last LiveTrack Viewer installation's architecture.

2.1.2 SDK Directory Contents

2.1.2.1 Documentation

2.1.2.2 lib/c++builder

C++ Builder LIB file. Add this to your C++ Builder project to link with the LiveTrack library.

2.1.2.3 lib/msvc

Microsoft Visual Studio LIB file. Add this to your C++ Builder project to link with the LiveTrack library.

2.1.2.4 Demo1, Demo2, Demo 3

Microsoft Visual Studio C++ demonstration projects.

2.1.2.5 Include

Contains the libLiveTrack.h header file for C/C++ support.

2.2 Using the API in Linux

The libraries required to build and run applications can be installed from the Presto MSD drive or obtained from the CRS website. Packages are provided in .deb format, and installation instructions are provided in a readme file.

2.3 Function Error Return Codes

`LIVETRACK_ERROR_OK` - Command succeeded.
`LIVETRACK_ERROR_HID HID` - Connection or command failure.
`LIVETRACK_ERROR_TIMEOUT` - Command timed out.
`LIVETRACK_ERROR_PARAM` - Command supplied with an invalid parameter.
`LIVETRACK_ERROR_NOT_SUPPORTED` - Functionality not supported.
`LIVETRACK_ERROR_FILE_WRITE` - Failed to write to the specified file (check path and path permissions).
`LIVETRACK_ERROR_FILE_READ` - Failed to open file (check that the file exists and that the permissions are correct).
`LIVETRACK_ERROR_FILE_VERSION` - File is incompatible with the current library version.
`LIVETRACK_ERROR_FILE_CORRUPT` - File does not contain the expected data (possibly corrupted file specified).
`LIVETRACK_ERROR_FILE_BAD_HEADER` - File does not contain the expected header (wrong file type specified).
`LIVETRACK_ERROR_INTERNAL` - Unexpected internal error, please report this.

`LIVETRACK_ERROR_NO_RESULT` - No results are available.
`LIVETRACK_ERROR_CALIBRATION_VERIFICATION` - Calibration data not successfully received by the LiveTrack application.
`LIVETRACK_ERROR_NOT_CONNECTED` - No LiveTrack device detected.
`LIVETRACK_ERROR_UNKNOWN` - Unexpected error, please report this.

2.4 API Functionality

2.4.1 crsLiveTrackInit

This function initialises LiveTrack.

C / C++ Syntax

```
int32_t crsLiveTrackInit(void);
```

Input Parameters

No input parameters required.

Return values

An integer with one of the following values:
`LIVETRACK_ERROR_NOT_CONNECTED`
1 - LiveTrack FM initialised.
2 - LiveTrack AP initialised.
3 - LiveTrack AV initialised.
4 - LiveTrack Presto initialised.

2.4.2 crsLiveTrackClose

This function closes LiveTrack. This function returns `LIVETRACK_ERROR_OK` if successful.

C / C++ Syntax

```
int32_t crsLiveTrackClose(void);
```

Input Parameters

No input parameters required.

Return values

`LIVETRACK_ERROR_OK` indicates LiveTrack has been successfully closed.

2.4.3 crsLiveTrackSetResultsType

This function sets the type of eye tracking results to be returned from LiveTrack.

C / C++ Syntax

```
int32_t crsLiveTrackSetResultsType(int32_t type);
```

Input Parameters

`type` – An integer that defines whether the results are to be returned as raw or calibrated. This must be set to either: `0` for raw data or `1` for calibrated data.
Raw data will return pupil position, gaze position and pupil major and minor axes in image pixel co-ordinates.
Calibrated data will return gaze position in screen co-ordinates and gaze direction in terms of Fick (Latitude, Longitude) and Helmholtz (Azimuth, Elevation). The calibrated setting will also return pupil major and minor axes in millimetres

Return values

This function returns `LIVETRACK_ERROR_PARAM` if the input parameter is not a recognised results type and returns `LIVETRACK_ERROR_OK` if the results type has been successfully set.

2.4.4 crsLiveTrackStartTracking

This function starts buffering (raw or calibrated) eye-tracking data and will continue to do so until crsLiveTrackStopTracking is called.

C / C++ Syntax

```
int32_t crsLiveTrackStartTracking(void)
```

Input Parameters

No input parameters required.

Return values

A return value of `LIVETRACK_ERROR_OK` indicates that eye tracking has successfully started.

2.4.5 crsLiveTrackStopTracking

This function stops eye tracking and stops recording results to the data buffer.

C / C++ Syntax

```
int32_t crsLiveTrackStopTracking(void)
```

Input Parameters

No input parameters required.

Return values

A return value of `LIVETRACK_ERROR_OK` indicates that eye tracking has successfully stopped.

2.4.6 crsLiveTrackGetResultsCount

This function returns the number of results currently stored in the crsLiveTrack results buffer.

C / C++ Syntax

```
int32_t crsLiveTrackGetResultsCount (void)
```

Input Parameters

No input parameters required.

Return values

An integer representing the number of eyetracking results currently held in the results buffer.

2.4.7 crsLiveTrackSetBufferPosition

This function sets the crsLiveTrack results buffer position.

C / C++ Syntax

```
int32_t crsLiveTrackSetBufferPosition(int32_t index);
```

Input Parameters

`index` -- required buffer position.

Return values

This function returns the integer `LIVETRACK_ERROR_OK` if successful or returns `LIVETRACK_ERROR_NO_RESULT` if the position is invalid.

2.4.8 crsLiveTrackGetBufferPosition

This function returns the current crsLiveTrack results buffer position.

C / C++ Syntax

```
int32_t crsLiveTrackGetBufferPosition(void);
```

Input Parameters

No input parameters required.

Return values

This function returns the current crsLiveTrack results buffer position.

2.4.9 crsLiveTrackGetBufferedResult

This function returns the result at the current crsLiveTrack buffer position (this position can be defined by crsLiveTrackSetBufferPosition).

C / C++ Syntax

```
int32_t crsLiveTrackGetBufferedResult(T_RESULTS_STRUCT* result , bool removeFromBuffer);
```

Input Parameters

`result` -- A pointer to the results data (see below)

`removeFromBuffer` – A Boolean variable indicating whether the returned results are to be removed from the buffer.

Return values

This function returns `LIVETRACK_ERROR_OK` if successful or returns `LIVETRACK_ERROR_NO_RESULT` if no result is returned.

The results are returned as a pointer to a structure which is of the form:

```
typedef struct T_RESULTS_STRUCT{
      uint16_t Size;          -- size of entire T_Results structure (bytes)
      uint64_t TimeStamp;     -- video frame time stamp (microseconds)
      uint16_t ResultsType;   -- see below for results types
      T_EYE_DATA Eye[2];      -- eye-tracking data sub structure
      uint16_t DigitalIO;     -- input / output trigger (see manual for details)
      bool FixationDetected;  -- fixation detected / not detected
}T_RESULTS_STRUCT;
```

Where `ResultsType` specifies raw or calibrated data (see crsLiveTrackSetResultsType for more information). The eye-tracking data is itself returned as a secondary `T_EYE_DATA` structure (one for each eye) which is of the form:

```
typedef struct {
      float GlintX, GlintY;          -- raw glint position (pixels)
      float PupilX, PupilY;          -- raw pupil position (pixels)
      float PupilMajorAxis, PupilMinorAxis;-- pupil ellipse axes (pixels)
      float VectX, VectY; -- distance between pupil and glint centres (pixels)
      float GazeX, GazeY, GazeZ;     -- calibrated gaze position (mm)
      float GazeAzimuth, GazeElevation;    -- calibrated gaze (degrees)
      float GazeLongitude, GazeLatitude;  -- calibrated gaze (degrees)
      bool Tracked;                        -- eye tracked / not tracked
      bool Enabled;                  -- eyetracking enabled / not enabled
      bool Calibrated;         -- eyetracking calibrated / not calibrated
      bool Dropped;                  -- eyetracking dropped / not dropped
      uint8_t ROI; -- this feature will be added in a future update
} T_EYE_DATA;
```

2.4.10 crsLiveTrackClearDataBuffer

This function clears all results from the crsLiveTrack data buffer.

C / C++ Syntax

```
int32_t crsLiveTrackClearDataBuffer (void)
```

Input Parameters

No input parameters required.

Return values

A value of `LIVETRACK_ERROR_OK` indicates that the LiveTrack data buffer has been successfully cleared.

2.4.11 crsLiveTrackGetLastResult

This function returns the most recent result stored in the crsLiveTrack results buffer. The result is returned in the form of a structure as shown below. The function will return `LIVETRACK_ERROR_OK` if successful, `LIVETRACK_NO_RESULT` if no result is available or `LIVETRACK_ERROR_NOT_CONNECTED` if there is no LiveTrack connected.

C / C++ Syntax

```
int32_t crsLiveTrackGetLastResult(T_RESULTS_STRUCT* results);
```

Input Parameters

`results` -- A pointer to the results data (see below)

Return values

This function returns `LIVETRACK_ERROR_OK` if successful or returns `LIVETRACK_ERROR_NO_RESULT` if no results are returned.

The results are returned as a pointer to a structure which is of the form:

```
typedef struct T_RESULTS_STRUCT{
uint16_t Size;                  -- size of entire T_Results structure (bytes)
      uint64_t TimeStamp;       -- video frame time stamp (microseconds)
      uint16_t ResultsType;     -- see below for results types
      T_EYE_DATA Eye[2];        -- eye-tracking data sub structure
      uint16_t DigitalIO;       -- input / output trigger (see manual for details)
      bool FixationDetected;    -- fixation detected / not detected
}T_RESULTS_STRUCT;
```

Where `ResultsType` specifies raw or calibrated data (see crsLiveTrackSetResultsType for more information). The eye-tracking data is itself returned as a secondary `T_EYE_DATA` structure (one for each eye) which is of the form:

```
typedef struct {
      float GlintX, GlintY;     -- raw glint position (pixels)
      float PupilX, PupilY;     -- raw pupil position (pixels)
      float PupilMajorAxis, PupilMinorAxis; -- pupil ellipse axes (pixels)
      float VectX, VectY; -- distance between pupil and glint centres (pixels)
      float GazeX, GazeY, GazeZ;      -- calibrated gaze position (mm)
      float GazeAzimuth, GazeElevation;     -- calibrated gaze (degrees)
      float GazeLongitude, GazeLatitude;  -- calibrated gaze (degrees)
      bool Tracked;                         -- eye tracked / not tracked
      bool Enabled;             -- eyetracking enabled / not enabled
      bool Calibrated;          -- eyetracking calibrated / not calibrated
      bool Dropped;             -- eyetracking dropped / not dropped
      uint8_t ROI;  -- this feature will be added in a future update
```

```
} T_EYE_DATA;
```

2.4.12 crsLiveTrackGetLibVersion

This function returns the version number of the LiveTrack library. This function will return an error if LiveTrack has not been initialized (using the crsLiveTrackInit command).

C / C++ Syntax

```
uint32_t crsLiveTrackGetLibVersion(void);
```

Input Parameters

No input parameters required.

Return values

An integer representing the LiveTrack library version number.

2.4.13 crsLiveTrackCalibrateDevice

This function calibrates the device for one eye.

C / C++ Syntax

```
int32_t crsLiveTrackCalibrateDevice(int32_t eye,
uint32_t numberOfPoints, targ_struct a[], double rpc,
double viewDist, int32_t *calErr);
```

Input Parameters

`eye` – An integer that defines which eye is to be calibrated. This must be set to either: `0` for the left eye or `1` for the right eye.

`numberOfPoints` – The number of targets used for calibration.

`a` – a one dimensional array of Target Structures which are of the form:

```
typedef struct targ_struct{
      double Target_X;
      double Target_Y;
      double Vector_X;
      double Vector_Y;
}targ_struct;
```

The target and pupil / glint fixation data array must be defined as follows:

```
target = (targ_struct *)calloc(numberOfPoints, sizeof(targ_struct));
for (i = 0; i < numberOfPoints; i++) {
```

`target[i].Target_X` --calibration target x position

`target[i].Target_Y` --calibration target y position

`target[i].Vector_X` --pupil x minus glint x fixation positions

`target[i].Vector_Y` --pupil y minus glint y fixation positions

```
}
```

`rpc` – The distance between centre of pupil and centre of eye rotation (nominally set to 100 camera pixels).

`viewDist` – The viewing distance between eye tracker and monitor in mm.

`calErr` – The calibration error representing the sum of square of displacement errors between the target positions and their corresponding calibrated gaze positions.

Return values

This function will return `LIVETRACK_ERROR_OK` if successful or return `LIVETRACK_ERROR_NOT_CONNECTED` if LiveTrack has not been initialized (using the crsLiveTrackInit command).

2.4.14 crsLiveTrackSetCalibration

This function sets the calibration matrix for either the left or right eye. (See also crsLiveTrackGetCalibration).

C / C++ Syntax

```
int32_t crsLiveTrackSetCalibration(int32_t eye, double cal[],
double rpc, double viewDist);
```

Parameters

`eye` – An integer that defines which eye is to be calibrated. This must be set to either: `0` for the left eye or `1` for the right eye.

`cal` – a 16 entry calibration matrix which can be obtained from crsLiveTrackGetCalibration.

`rpc` – The distance between centre of pupil and centre of eye rotation (nominally set to 100 camera pixels).

`viewDist` – The distance between the test subject and the stimulus (calibration) monitor in millimetres.

2.4.15 crsLiveTrackGetCalibration

This function returns the calibration matrix, rpc value and the viewing distance for either the left or right eye (See also crsLiveTrackSetCalibration).

C / C++ Syntax

```
int32_t crsLiveTrackGetCalibration(int32_t eye, double cal[],
double *rpc, double *viewDist);
```

Parameters

`eye` – An integer that defines which eye calibration details are to be returned. This must be set to either: `0` for the left eye or `1` for the right eye.

Return values

`cal` – a 16 entry returned calibration matrix.

`rpc` – The currently set distance between centre of pupil and centre of eye rotation (nominally set to 100 camera pixels).

`viewDist` – The currently set viewing distance between the test subject and the stimulus (calibration) monitor in millimetres.

2.4.16 crsLiveTrackGetPupilCalibration

This function returns the video image pixels-to-millimetre scaling factor. This can be modified by using crsLiveTrackSetPupilCalibration.

C / C++ Syntax

```
int32_t crsLiveTrackGetPupilCalibration(double *pixelsToMM);
```

Parameters

`pixelsToMM` – pixels to millimetre scaling factor.

2.4.17 crsLiveTrackSetPupilCalibration

This function sets the video image pixels-to-millimetre scaling factor. This set to a default value but can be modified by tracking a pupil or calibration dot target of known diameter (e.g. the M2005 Pupil Calibration Stick supplied with your LiveTrack system)..

C / C++ Syntax

```
int32_t crsLiveTrackSetPupilCalibration(double diameter, double
pixels);
```

Parameters

`diameter` – pupil or calibration dot target diameter in millimetres.

2.4.18 crsLiveTrackSetTracking

This function defines which eyes are to be tracked.

C / C++ Syntax

```
int32_t crsLiveTrackSetTracking(bool leftEye, bool rightEye);
```

Parameters

`leftEye` – a Boolean value to track the left eye (`true` / `false`).

`rightEye` – a Boolean value to track the right eye (`true` / `false`).

2.4.19 crsLiveTrackGetTracking

This function returns which eyes are currently set to be tracked.

C / C++ Syntax

```
int32_t crsLiveTrackGetTracking(bool *leftEye, bool *rightEye);
```

Parameters

`leftEye` – the left eye is set to be tracked (`true` / `false`).

`rightEye` – the right eye is set to be tracked (`true` / `false`).

2.4.20 crsLiveTrackGetSerialNumber

This function obtains the LiveTrack serial number.

C / C++ Syntax

```
int32_t crsLiveTrackGetSerialNumber(char *serialNumber, size_t
maxLength);
```

Parameters

`serialNumber` – the LiveTrack serial number.

`maxLength` – the maximum expected length of the LiveTrack serial number. This is normally set to 16 as the LiveTrack serial number is usually 8 digits.

2.4.21 crsLiveTrackGetFirmwareVersion

This function obtains the LiveTrack firmware version number.

C / C++ Syntax

```
int32_t crsLiveTrackGetFirmwareVersion(uint16_t *version);
```

Parameters

`version` – the LiveTrack firmware version number.

2.4.22 crsLiveTrackGetFirmwareBuildDateTime

This function obtains the LiveTrack build date and time.

C / C++ Syntax

```
int32_t crsLiveTrackGetFirmwareBuildDateTime(char *dateTime,
size_t maxLength);
```

Parameters

`dateTime` – a returned string containing the LiveTrack build date and time.

`maxLength` – the maximum expected length of the LiveTrack build date and time. This is normally set to 128.

2.4.23 crsLiveTrackSaveCalibration

This function saves the gaze and pupil size calibration of CRS LiveTrack Toolbox to a file. (See also crsLiveTrackLoadCalibration).

C / C++ Syntax

```
int32_t crsLiveTrackSaveCalibration(char *filename);
```

Parameters

`filename` - the filename to save the calibration to (including the path)

Return values

An integer where `LIVETRACK_ERROR_OK` indicates the calibration file has been successfully saved

Example

```
crsLiveTrackSaveCalibration('C:\tmp\calibration.dat')
```

2.4.24 crsLiveTrackLoadCalibration

This function loads the gaze and pupil size calibration of CRS LiveTrack Toolbox to a file. (See also crsLiveTrackSaveCalibration).

C / C++ Syntax

```
int32_t crsLiveTrackLoadCalibration(char *filename);
```

Parameters

`filename` - the filename to load to including the path

Return values

An integer where `LIVETRACK_ERROR_OK` indicates the calibration file has been successfully loaded

Example

```
crsLiveTrackLoadCalibration('C:\tmp\calibration.dat')
```

2.4.25 crsLiveTrackGetCalibrated

This function determines whether or not the toolbox has been calibrated for a specified eye.

C / C++ Syntax

```
int32_t crsLiveTrackGetCalibrated(int32_t eye, bool *calibrated);
```

Parameters

`eye` – An integer that defines which eye you wish to check calibration status for. This must be set to either: 0 for the left eye or 1 for the right eye.

Return values

`calibrated` - a Boolean, where `0`=False (uncalibrated) or `1`=True (calibrated) for the specified eye.

This function will return `LIVETRACK_ERROR_OK` if the calibration status for the specified eye has been successfully returned.

2.4.26 crsLiveTrackGetPupilCalibrated

This function determines whether or not the pupil scale has been calibrated.

C / C++ Syntax

```
int32_t crsLiveTrackGetPupilCalibrated(bool *calibrated);
```

Parameters

No input parameters required.

Return values

`calibrated` - a Boolean indicating whether the pupil scale has been calibrated (or not calibrated).
This function will return `LIVETRACK_ERROR_OK` if the pupil calibration status has been successfully returned.

2.4.27 crsLiveTrackGetTrackingStarted

This function is used to check whether or not tracking data is being saved into the buffer.

C / C++ Syntax

```
bool crsLiveTrackGetTrackingStarted(void);
```

Input Parameters

No input parameters required.

Return values

A Boolean indicating whether or not tracking data is being saved into the buffer.

2.4.28 crsLiveTrackGetCaptureConfig

This function returns the capture configuration of the LiveTrack device.

C / C++ Syntax

```
int32_t crsLiveTrackGetCaptureConfig(uint32_t *width, uint32_t
*height, uint32_t *rate, uint32_t *offsetX, uint32_t *offsetY);
```

Parameters

No input parameters required.

Return values

`width` - width (in pixels) of the video capture image window
`height` - height (in pixels) of the video capture image window
`rate` - configured actual capture rate in frames per second (rounded to an integer).
`offsetX` - horizontal offset (in pixels) of the video capture image window
`offsetY` - vertical offset (in pixels) of the video capture image window
This function will return `LIVETRACK_ERROR_OK` if the capture configuration has been successfully returned.

2.4.29 crsLiveTrackGetCaptureConfig2

This function returns the capture configuration of the LiveTrack device. In this function, rate provides the actual capture rate in frames per second as a float.

C / C++ Syntax

```
int32_t crsLiveTrackGetCaptureConfig2(uint32_t *width, uint32_t
*height, float *rate, uint32_t *offsetX, uint32_t *offsetY , int32_t
*cameraConnection);
```

Parameters

No input parameters required.

Return values

`width` - width (in pixels) of the video capture image window
`height` - height (in pixels) of the video capture image window
`rate` - actual capture rate in frames per second
`offsetX` - horizontal offset (in pixels) of the video capture image window
`offsetY` - vertical offset (in pixels) of the video capture image window
`cameraConnection` - indicates the state of the camera connection

This function will return `LIVETRACK_ERROR_OK` if the capture configuration has been successfully returned.

2.4.30 crsLiveTrackGetAbiVersion

This function returns the ABI (application binary interface) version number of the LiveTrack library which is used for binary interface compatibility testing.
The Most Significant Byte (MSB) indicates compatibility between the current library and the header file. If this number differs then the current library is incompatible with the header file. The Least Significant Byte (LSB) is used to track changes that do not break compatibility such as the addition of new functions.

C / C++ Syntax

```
uint32_t crsLiveTrackGetAbiVersion(void);
```

Input Parameters

No input parameters required.

Return values

An integer representing the ABI (application binary interface) LiveTrack library version number.

2.4.31 crsLiveTrackSetDataFilename

When this function is called, subsequent tracking data is written to the specified file in CSV format as it is captured until crsLiveTrackCloseDataFile is called. If tracking is stopped and restarted, data will continue to be appended to the open file. If this function is called with a different filename to one that has already been opened this way, the first file is closed and the new one opened. crsLiveTrackSetDataFilenameW adds wide string support.

C / C++ Syntax

```
int32_t crsLiveTrackSetDataFilename(const char *filename);
int32_t crsLiveTrackSetDataFilenameW(const wchar_t *filename);
```

Input Parameters

filename - A string containing the filename of a file to write data to.

Return values

This function will return `LIVETRACK_ERROR_OK` if successful.

2.4.32 crsLiveTrackCloseDataFile

This function stops any further date being written to the file opened with the function above, and closes the file.

C / C++ Syntax

```
int32_t crsLiveTrackCloseDataFile (void);
```

Input Parameters

No input parameters required.

Return values

This function will return `LIVETRACK_ERROR_OK` if successful.

2.4.33 crsLiveTrackSetDataComment

This function allows text strings to be written into the CSV data file opened with crsLiveTrackSetDataFilename. These comments are written with the next sample as

soon as possible and appear in the rightmost field. They can be used to insert, for example, trial or stimulus related event information.

C / C++ Syntax

```
int32_t crsLiveTrackSetDataComment(const char *comment);
```

Input Parameters

`comment` - A string containing the comment to be written in to the data file.

Return values

This function will return `LIVETRACK_ERROR_OK` if successful.

2.4.34 crsLiveTrackGetLastResultEP

E-Prime compatible version of crsLiveTrackGetLastResult.

C / C++ Syntax

```
int32_t crsLiveTrackGetLastResultEP(T_RESULTS_META_DATA_EP meta,
T_EYE_DATA_EP* leftEye, T_EYE_DATA_EP* rightEye);
```

Input Parameters

`meta` - pointer to the metadata (see below)

`leftEye` - pointer to the tracking data from the left eye (see below)

`rightEye` - pointer to the tracking data from the right eye (see below)

Return values

This function returns `LIVETRACK_ERROR_OK` if successful or returns `LIVETRACK_ERROR_NO_RESULT` if no result is returned.

Meta data is returned in a structure of the form (All types are 4 or 8 bytes long to avoid packing issues. E-Prime data types are all Long):

```
typedef struct {
      int32_t TimestampMS;        -- timestamp in milliseconds (most
                                  significant digits of timestamp).
      int32_t TimestampLS;        -- remainder of timestamp in microseconds,
                                  after removal of above (most significant
                                  digits of timestamp)
      int32_t ResultsType;        – specifies raw or calibrated data (see
                                  crsLiveTrackSetResultsType for more
                                  information)
      int32_t DigitalIO;          -- input / output trigger (see manual for
                                  details)
      int32_t FixationDetected;   -- fixation detected / not detected
}T_RESULTS_META_DATA_EP;
```

Left eye and right eye data are returned, each in a structure of the form (equivalent E-Prime data types in []):

```
typedef struct {
      int32_t ActiveROIs; --Bitfield indicating which RIOs are active (1 = active).
                                  [Long]
      float GlintX, GlintY;       -- raw glint position (pixels) [Single]
      float PupilX, PupilY;       -- raw pupil position (pixels) [Single]
      float PupilMajorAxis, PupilMinorAxis;      -- pupil ellipse axes
                                  (pixels) [Single]
```

```
        float VectX, VectY;             -- distance between pupil and glint centres
                                        (pixels) [Single]
        float GazeX, GazeY, GazeZ;      -- calibrated gaze position (mm) [Single]
        float GazeAzimuth, GazeElevation;    -- calibrated gaze (degrees)
                                        [Single]
        float GazeLongitude, GazeLatitude;   -- calibrated gaze (degrees)
                                        [Single]
        int32_t Tracked;                -- eye tracked / not tracked [Long]
        int32_t Enabled;                -- eyetracking enabled / not enabled [Long]
        int32_t Calibrated;             -- eyetracking calibrated / not calibrated
                                        [Long]
        int32_t Dropped;                -- eyetracking dropped / not dropped [Long]
        int32_t ROI;                    -- this feature will be added in a future
                                        update
} T_EYE_DATA_EP;
```

2.4.35 crsLiveTrackGetBufferedResultEP

E-Prime compatible version of crsLiveTrackGetBufferedResultEP.

C / C++ Syntax

```
int32_t crsLiveTrackGetBufferedResultEP(T_RESULTS_META_DATA_EP* meta,
T_EYE_DATA_EP* leftEye, T_EYE_DATA_EP* rightEye, int32_t
removeFromBuffer);
```

Input Parameters

`meta` - pointer to the metadata (see above)

`leftEye` - pointer to the tracking data from the left eye (see above)

`rightEye` - pointer to the tracking data from the right eye (see above)

`removeFromBuffer` - An integer value, 0 or 1, taken as a Boolean indicating whether the returned results are to be removed from the buffer.

Return values

This function returns `LIVETRACK_ERROR_OK` if successful or returns `LIVETRACK_ERROR_NO_RESULT` if no result is returned. Metadata and eye data are returned in structures as above.

3 MATLAB Toolbox Commands

The CRS LiveTrack Toolbox for MATLAB contains the commands to control LiveTrack from within MATLAB. It is included in the Software directory of your MSD drive or can be downloaded from the CRS website. The CRS LiveTrack Toolbox installer (which installs LiveTrack Viewer) for Windows will also, optionally, install the toolbox on your computer. Once you have the toolbox on your system, you will need to add the folder and all subfolders to your Matlab path. What follows is a listing of the functions that make up the MATLAB interface, a brief description of their usage and how they are called.

3.1.1 crsLiveTrackInit

This function initialises the crsLiveTrack MATLAB Toolbox and loads the necessary eye-tracking parameter constants so that meaningful function parameters can be used.

MATLAB Syntax

```
deviceType = crsLiveTrackInit
```

Parameters

No input parameters required.

Return values

`deviceType` **-** An integer with one of the following values:

`-100` **-** ERROR: LiveTrack device not found!

`1` **-** LiveTrack FM initialised.

`2` **-** LiveTrack AP initialised.

`3` **-** LiveTrack AV initialised.

`4` **-** LiveTrack Presto initialised.

3.1.2 crsLiveTrackStartTracking

This function starts eye tracking and starts recording results to the data buffer.

MATLAB Syntax

```
err = crsLiveTrackStartTracking
```

Parameters

No input parameters required.

Return values

`err` - An integer where 0 indicates eye tracking has successfully started

3.1.3 crsLiveTrackStopTracking

This function stops eye tracking and stops recording results to the data buffer.

MATLAB Syntax

```
err = crsLiveTrackStopTracking
```

Parameters

No input parameters required.

Return values

`err` - An integer where 0 indicates eye tracking has successfully stopped.

3.1.4 crsLiveTrackGetResultsCount

This function returns the number of results currently stored in the LiveTrack results buffer.

MATLAB Syntax

```
noOfResults = crsLiveTrackGetResultsCount
```

Parameters

No input parameters required.

Return values

`noOfResults` - An integer representing the number of eyetracking results currently held in the results buffer.

3.1.5 crsLiveTrackClose

This function closes the connection to the LiveTrack. This function returns 0 if successful.

MATLAB Syntax

```
err = crsLiveTrackClose
```

Parameters

No input parameters required.

Return values

`err` - An integer where `0` indicates LiveTrack has been successfully closed.

3.1.6 crsLiveTrackGetLatestEyePosition

This function returns the most recent eye position analysed by the LiveTrack library. This data point is NOT removed from the data buffer that the LiveTrack library maintains. (i.e. you can get the same results later using crsLiveTrackGetBufferedEyePositions). It is also possible to get more than one sample by specifying the input parameter maximumPoints.
If there are no points available in the buffer, a MATLAB structure filled with zeros is returned, together with a non-zero (-1) ErrorCode. This may happen if you attempt to read data soon after tracking starts, as eye position data can be delayed by the user-specifiable fixation period, defined using the crsLiveTrackSetFixationPeriod command.

MATLAB Syntax

```
Data = crsLiveTrackGetLatestEyePosition
[ Data, ErrorCode ] = crsLiveTrackGetLatestEyePosition
Data = crsLiveTrackGetLatestEyePosition(maximumPoints)
[ Data, ErrorCode ] = crsLiveTrackGetLatestEyePosition(maximumPoints)
```

Parameters

`maximumPoints` - (Optional parameter) A scalar integer indicating the maximum number of points to return to MATLAB.

Return values

`Data` - A MATLAB structure containing the following eye position data:
`Data.timeStamps` - Milliseconds since tracking started.
`Data.digitalIO` - Integer digital IO value.
`Data.fixation` - `1` if eyes are fixating, `0` otherwise.
The following fields relate to the left eye:
`Data.mmPositions` - Direction of gaze projected onto stimulus display, in mm: [X,Y].
`Data.fickPositions` - Eye rotation in Fick coordinates: `[Longitude, Latitude]`.
`Data.helmholtzPositions` - Eye rotation in Helmholtz coordinates: `[Elevation, Azimuth]`.
`Data.pupilDiameter` - Pupil diameter in mm.
`Data.tracked` - `1` if frame tracked, `0` otherwise.
`Data.dropped` - `1` if frame dropped, `0` otherwise.
`Data.calibrated` - `1` if eye calibrated, `0` otherwise.
`Data.region` - this feature will be added in a future update.
`Data.pupilPositions` - when streaming raw data, the location of the centre of the pupil in camera pixels `[X, Y]`.

`Data.glintPositions` - when streaming raw data, the location of the centre of the glint in camera pixels `[X, Y]`.
The following fields relate to the right eye:
`Data.mmPositionsRight` - Direction of gaze projected onto stimulus display, in mm: `[X,Y]`.
`Data.fickPositionsRight` - Eye rotation in Fick coordinates: `[Longitude, Latitude]`.
`Data.helmholtzPositionsRight` - Eye rotation in Helmholtz coordinates: `[Elevation, Azimuth]`.
`Data.pupilDiameterRight` - Pupil diameter in mm.
`Data.trackedRight` - `1` if frame tracked, `0` otherwise.
`Data.droppedRight` - `1` if frame dropped, `0` otherwise.
`Data.calibratedRight` - `1` if eye calibrated, `0` otherwise.
`Data.regionRight` - this feature will be added in a future update.
`Data.pupilPositionsRight` - when streaming raw data, the location of the centre of the pupil in camera pixels `[X, Y]`.
`Data.glintPositionsRight` - when streaming raw data, the location of the centre of the glint in camera pixels `[X, Y]`.
`ErrorCode` - An integer where `0` indicates LiveTrack has successfully returned the latest eye position.

Example

Since data can be delayed by the fixation period, data may not be ready the first time you attempt to read the latest eye position. In that case, the following section of code should allow you to pause until valid positions can be read:

```
crsLiveTrackStartTracking;
[Data,ErrorCode] = crsLiveTrackGetLatestEyePosition;
while ErrorCode~=0
      [Data,ErrorCode] = crsLiveTrackGetLatestEyePosition;
      pause(0.1);
end;
```

3.1.7 crsLiveTrackGetBufferedEyePositions

This function returns eye position data from the buffer that is created and maintained by the LiveTrack library. Eye positions that are returned by this function are (by default) removed from the data buffer, so subsequent calls will not return duplicate data.
If there are no points available in the buffer, a structure filled with zeros is returned.
This function has two optional parameters, a Boolean/logical value: `removeFromBuffer`, and a numeric (integer) value: `maximumPoints`. If the `removeFromBuffer` parameter is set to False, then data points are left in the buffer, allowing them to be retrieved again by a second call. If the `removeFromBuffer` parameter is set to True, then data points are deleted from the buffer after they are read. The `maximumPoints` parameter allows you to limit the number of points returned by this function. This is useful if the buffer is large and you can only deal with a limited number of points at any one time.

MATLAB Syntax

```
Data = crsLiveTrackGetBufferedEyePositions;
```

```
Data = crsLiveTrackGetBufferedEyePositions(removeFromBuffer);
Data = crsLiveTrackGetBufferedEyePositions(maximumPoints);
Data =
crsLiveTrackGetBufferedEyePositions(maximumPoints,removeFromBuffer);
```

Parameters

`maximumPoints` - (Optional parameter) A scalar integer indicating the maximum number of points to return to MATLAB.

`removeFromBuffer` - (Optional parameter) A scalar logical controlling the removal of returned eye positions from the buffer.

Return values

`Data` - A MATLAB structure containing the following eye position data:

`Data.timeStamps` - Milliseconds since tracking started.

`Data.digitalIO` - Integer digital IO value.

`Data.fixation` - `1` if eyes are fixating, `0` otherwise.

The following fields relate to the left eye:

`Data.mmPositions` - Direction of gaze projected onto stimulus display, in mm: `[X,Y]`.

`Data.fickPositions` - Eye rotation in Fick coordinates: `[Longitude, Latitude]`.

`Data.helmholtzPositions` - Eye rotation in Helmholtz coordinates: `[Elevation, Azimuth]`.

`Data.pupilDiameter` - Pupil diameter in mm.

`Data.tracked` - `1` if frame tracked, `0` otherwise.

`Data.dropped` - `1` if frame dropped, `0` otherwise.

`Data.calibrated` - 1 if eye calibrated, `0` otherwise.

`Data.region` - this feature will be added in a future update.

`Data.pupilPositions` - when streaming raw data, the location of the centre of the pupil in camera pixels `[X, Y]`.

`Data.glintPositions` - when streaming raw data, the location of the centre of the glint in camera pixels `[X, Y]`.

The following fields relate to the right eye:

`Data.mmPositionsRight` - Direction of gaze projected onto stimulus display, in mm: `[X,Y]`.

`Data.fickPositionsRight`- Eye rotation in Fick coordinates: `[Longitude, Latitude]`.

`Data.helmholtzPositionsRight` - Eye rotation in Helmholtz coordinates: `[Elevation, Azimuth]`.

`Data.pupilDiameterRight` - Pupil diameter in mm.

`Data.trackedRight` - `1` if frame tracked, `0` otherwise.

`Data.droppedRight` - `1` if frame dropped, `0` otherwise.

`Data.calibratedRight` - `1` if eye calibrated, `0` otherwise.

`Data.regionRight` - this feature will be added in a future update.

`Data.pupilPositionsRight` - when streaming raw data, the location of the centre of the pupil in camera pixels `[X, Y]`.

`Data.glintPositionsRight` - when streaming raw data, the location of the centre of the glint in camera pixels `[X, Y]`.

`ErrorCode` - An integer where `0` indicates LiveTrack has successfully returned the latest eye position.

3.1.8 crsLiveTrackGetLibraryVersion

This function returns the version number of the LiveTrack library. This function will return an error if LiveTrack has not been initialized (using the crsLiveTrackInit command).

MATLAB Syntax

```
Lib_Version = crsLiveTrackGetLibraryVersion
```

Parameters

No input parameters required.

Return values

`Lib_Version` - An integer scalar representing the LiveTrack library version number.

3.1.9 crsLiveTrackCalibrateDevice

This function calibrates the device for one designated eye. This function will return the sum of square of displacement errors between the target positions and the corresponding calibrated gaze positions, in both degrees and mm. An error code will return 0 if successful, or -100 if LiveTrack has not been initialized (using the crsLiveTrackInit command).

MATLAB Syntax

```
[ errorDeg, errorMM, calErr ] = crsLiveTrackCalibrateDevice(whichEye,
pupil, glint, targets, viewDist)
```

Parameters

`whichEye` – This will accept one of two literal strings:
`'left'` to calibrate the left eye or …
`'right'` to calibrate the right eye
`pupil` – a matrix containing the x and y pixel positions of the pupil centres. This matrix will be 2 columns wide and will contain one row for each calibration target.
`glint` – a matrix containing the x and y pixel positions of the purkinje glint centres. This matrix will be 2 columns wide and will contain one row for each calibration target.
`targets` – a matrix containing the x and y screen positions of the calibration targets. This matrix will be 2 columns wide and will contain one row for each calibration target in mm.
`viewDist` – The viewing distance between eye tracker and monitor in mm.

Return values

`errorDeg`- RMS calibration displacement error between target and calculated viewpoint positions (in degrees)
`errorMM` – as above (in millimetres).
`calErr`- An integer scalar where `0` represents a successful calibration.

3.1.10 crsLiveTrackGetCaptureConfig

This function returns the capture configuration.

MATLAB Syntax

```
[ width, height, rate, offsetX, offsetY, ErrorCode ] =
crsLiveTrackGetCaptureConfig
```

Parameters
No input parameters required.
Return values
`width` - width (in pixels) of the video capture image window
`height` - height (in pixels) of the video capture image window
`rate` - frame rate of the video capture image window
`offsetX` - horizontal offset (in pixels) of the video capture image window
`offsetY` - vertical offset (in pixels) of the video capture image window
`ErrorCode` - An integer scalar where `0` represents a successful return of the capture configuration.

3.1.11 crsLiveTrackGetTracking

This function returns the tracking state of the left and right eyes (see also crsLiveTrackSetTracking.)
MATLAB Syntax
```
[ leftEye, rightEye, ErrorCode ] = crsLiveTrackGetTracking
```
Parameters
No input parameters required.
Return values
`leftEye` - returns `1` (or `0`) if left eye tracking state is set (not set)
`rightEye` - returns `1` (or `0`) if right eye tracking state is set (not set)
`ErrorCode` - An integer where `0` indicates eye tracking status has been successfully returned.

3.1.12 crsLiveTrackSetTracking

This function sets the tracking states for the left and right eyes. (See also crsLiveTrackGetTracking.)
MATLAB Syntax
```
err = crsLiveTrackSetTracking( leftEye, rightEye )
```
Parameters
`leftEye` - set to `true` (or `false`) if left eye is to be tracked (not tracked)
`rightEye` - set to true (or `false`) if right eye is to be tracked (not tracked)
Return values
`err` - An integer where `0` indicates eye tracking has been successfully set

3.1.13 crsLiveTrackSetResultsTypeRaw

This function instructs LiveTrack to send raw data, rather than calibrated data. No input parameters required.
Raw data consists of the x and y coordinates of the centre location of the pupil and glint in the camera image. In dual glint systems the geometric average will be returned. The units will be retuned in pixels of the camera sensor. The pupil ellipse fit returns the major and minor axes in camera pixels.
MATLAB Syntax
```
err = crsLiveTrackSetResultsTypeRaw
```
Parameters
No input parameters required.
Return values

`err` - An integer where `0` indicates the results type has been successfully set to raw.

3.1.14 crsLiveTrackSetResultsTypeCalibrated

This function instructs LiveTrack to send calibrated data, rather than raw data. No input parameters required.
Calibrated data consists of the x, y and z coordinates of the gaze vector, where x and y are in the plane of the screen and z is the viewing distance. Units are typically in millimetres but can be calibrated to any units. The units will those used for 'targets' when calibrating with crsLiveTrackCalibrateDevice.
The pupil ellipse fit major and minor axis will be scaled according to the scalefactor set by crsLiveTrackSetPupilCalibration or if not set, then a default calibration will used. You can check the value using the function crsLiveTrackGetPupilCalibration.

MATLAB Syntax

```
err = crsLiveTrackSetResultsTypeCalibrated
```

Parameters

No input parameters required.

Return values

`err` - An integer where `0` indicates the results type has been successfully set to calibrated.

3.1.15 crsLiveTrackSetCalibration

This function sets the gaze calibration. (See also crsLiveTrackGetCalibration).

MATLAB Syntax

```
err = crsLiveTrackSetCalibration(whichEye,calibration,viewDist)
```

Parameters

`whichEye` - this must be set to either: `'leftEye'` or `'righteye'` depending on which eye you wish to set calibration for.
`calibration` - a 16 entry array containing the gaze calibration parameters. (This is created by crsLiveTrackGetCalibration).
`viewDist` - viewing distance in millimeters

Return values

`err` - An integer where `0` indicates calibration has been successfully set.

3.1.16 crsLiveTrackGetCalibration

This function returns the 16 entry gaze calibration array for a designated eye. (See also crsLiveTrackSetCalibration).

MATLAB Syntax

```
calibration = crsLiveTrackGetCalibration(whichEye)
```

Parameters

`whichEye` - this must be set to either: `'leftEye'` or `'righteye'` depending on which eye you wish to get calibration for.

Return values

`calibration` - a 16 entry array containing the gaze calibration parameters for the specified eye.

3.1.17 crsLiveTrackSaveCalibration

This function saves the gaze and pupil size calibration to a file. (See also crsLiveTrackLoadCalibration).

MATLAB Syntax

```
err = crsLiveTrackSaveCalibration(filename)
```

Parameters

`filename` - the filename to save the calibration to (including the path**)**

Return values

`err` - An integer where `0` indicates the calibration file has been successfully saved

Example

```
crsLiveTrackSaveCalibration('C:\tmp\calibration.dat')
```

3.1.18 crsLiveTrackLoadCalibration

This function loads the gaze and pupil size calibration from a file. (See also crsLiveTrackSaveCalibration).

MATLAB Syntax

```
err = crsLiveTrackLoadCalibration(filename)
```

Parameters

`filename` - the filename to load to including the path

Return values

`err` - An integer where `0` indicates the calibration file has been successfully loaded

Example

```
crsLiveTrackLoadCalibration('C:\tmp\calibration.dat')
```

3.1.19 crsLiveTrackGetFirmwareVersion

This function returns the version of the firmware currently running on the LiveTrack.

MATLAB Syntax

```
FirmwareVersion = crsLiveTrackGetFirmwareVersion
```

Parameters

No input parameters required.

Return values

`FirmwareVersion` **-** An integer representing the firmware version number.

3.1.20 crsLiveTrackGetFirmwareBuildDateTime

This function returns the build date and time of the firmware currently running on the LiveTrack.

MATLAB Syntax

```
FirmwareBuildDateTime = crsLiveTrackGetFirmwareBuildDateTime
```

Parameters

No input parameters required.

Return values

`FirmwareBuildDateTime` **-** A string representing the build date and time of the firmware.

3.1.21 crsLiveTrackGetPupilCalibrated

This function determines whether or not the pupil scale has been calibrated.

MATLAB Syntax

```
[isCalibrated, ErrorCode] = crsLiveTrackGetPupilCalibrated;
```

Parameters
No input parameters required.
Return values
`isCalibrated` - a Boolean (1-by-1 matrix), where `1` (or `0`) indicates that the pupil scale has been calibrated (or not calibrated).
`ErrorCode` - An integer where `0` indicates pupil calibration status has been successfully returned.

3.1.22 crsLiveTrackSetPupilCalibration

This function sets the pupil size scale factor. It is meant to be used after raw data has been obtained while tracking an artificial pupil of a known diameter (e.g. the M2005 Pupil Calibration Stick supplied with your LiveTrack system).
MATLAB Syntax
```
err = crsLiveTrackSetPupilCalibration(diameter, pixels)
```
Parameters
`diameter` **-** the size (in millimetres) of the artificial pupil
`pixels` **-** the major axis of the tracked artificial pupil in image pixels.
Return values
`err` - An integer where `0` indicates the pupil scale calibration is successfully set.

3.1.23 crsLiveTrackGetPupilCalibration

This function returns the pupil size scale factor of CRS LiveTrack in pixels per millimetre.
MATLAB Syntax
```
pupilSizeScaleFactor = crsLiveTrackGetPupilCalibration
```
Parameters
No input parameters required.
Return values
`pupilSizeScaleFactor` **-** The pupil size scale factor of CRS LiveTrack in pixels per millimetre.

3.1.24 crsLiveTrackGetSerialNumber

This function returns the crsLiveTrack MATLAB Toolbox unique serial number.
MATLAB Syntax
```
serialNumber = crsLiveTrackGetSerialNumber
```
Parameters
No input parameters required.
Return values
`serialNumber` - An integer representing the unique serial number for the CRS LiveTrack device.

3.1.25 crsLiveTrackGetToolboxVersion

This function returns the crsLiveTrack MATLAB Toolbox version number.
MATLAB Syntax
```
version = crsLiveTrackGetToolboxVersion
```
Parameters
No input parameters required.

Return values

`version` - An integer representing the crsLiveTrack MATLAB Toolbox version number.

3.1.26 crsLiveTrackIsEyeDataAvailable

This function checks to see if eye tracking data is available.

MATLAB Syntax

```
eyeDataAvailability = crsLiveTrackIsEyeDataAvailable;
```

Parameters

No input parameters required.

Return values

`eyeDataAvailability` **-** An integer scalar holding the value `1` if eye data is available, and `0` if it is not.

3.1.27 crsLiveTrackClearDataBuffer

This function clears the LiveTrack data buffer.

MATLAB Syntax

```
err = crsLiveTrackClearDataBuffer
```

Parameters

No input parameters required.

Return values

`err` **-** An integer where `0` indicates the LiveTrack data buffer has been successfully cleared.

3.1.28 crsLiveTrackGetCalibrated

This function determines whether or not the toolbox has been calibrated for a specified eye.

MATLAB Syntax

```
[isCalibrated, ErrorCode] = crsLiveTrackGetCalibrated(whichEye);
```

Parameters

`whichEye` - this must be set to either: 'lefteye' or 'righteye' depending on which eye you wish to check calibration status for.

Return values

`isCalibrated` - a Boolean, where `0`=False (uncalibrated) or `1`=True (calibrated).
`ErrorCode` **-** An integer where `0` indicates the calibration status for the specified eye has been successfully returned.

3.1.29 crsLiveTrackStartVideoRecord

This function starts recording eyetracking video to file. (See also crsLiveTrackStopVideoRecord).

MATLAB Syntax

```
err = crsLiveTrackStartVideoRecord(filename)
```

Parameters

`filename` - the filename to load to including the path

Return values

`err` - An integer where `0` indicates video recording has successfully started

Example

```
crsLiveTrackStartVideoRecord('C:\tmp\recorded_video.avi')
```

3.1.30 crsLiveTrackStopVideoRecord

This function stops recording eyetracking video to file (See also crsLiveTrackStartVideoRecord).

MATLAB Syntax

```
err = crsLiveTrackStopVideoRecord
```

Parameters

No input parameters required.

Return values

`err` - An integer where `0` indicates video recording has been successfully stopped.

3.1.31 crsLiveTrackSetDataFilename

When this function is called, subsequent tracking data is written to the specified file in CSV format as it is captured until crsLiveTrackCloseDataFile is called. If tracking is stopped and restarted, data will continue to be appended to the open file. If this function is called with a different filename to one that has already been opened this way, the first file is closed and the new one opened. crsLiveTrackSetDataFilenameW adds wide string support.

MATLAB Syntax

```
err = crsLiveTrackSetDataFilename(filename)
```

Parameters

`filename` - the filename to save the data to (including the path**)**

An integer where `0` indicates the date filename has been successfully set **Example**

```
crsLiveTrackSetDataFilename('C:\tmp\LiveTrack_data.csv ')
```

3.1.32 crsLiveTrackCloseDataFile

This function stops any further data being written to the file opened with crsLiveTrackSetDataFilename, and closes the file.

MATLAB Syntax

```
err = crsLiveTrackCloseDataFile
```

Parameters

No input parameters required.

Return values

An integer where `0` indicates the file was successfully closed.

3.1.33 crsLiveTrackSetDataComment

This function allows text strings to be written into the CSV data file opened with crsLiveTrackSetDataFilename. These comments are written with the next sample as soon as possible and appear in the rightmost field. They can be used to insert, for example, trial or stimulus related event information.

MATLAB Syntax

```
err = crsLiveTrackSetDataComment(comment)
```

Parameters

`comment` - the string to insert into the data as a comment

Return values

`err` - An integer where `0` indicates the comment was successfully set

Example

```
crsLiveTrackSetDataFilename('Trial 1. Contrast 0.5')
```

4 Python

The CRS LiveTrack Python module contains the commands to control LiveTrack from within Python. Some example scripts are also provided. These files included in the Software directory of your MSD drive. What follows is a listing of the structures and functions that make up the Python interface, a brief description of their usage and how they are called.

4.1.1 LiveTrack.Init

This function initialises the crsLiveTrack MATLAB Toolbox and loads the necessary LiveTrack eye-tracking parameter constants so that meaningful function parameters can be used.

Python Syntax

```
deviceType = LiveTrack.Init()
```

Parameters

No input parameters required.

Return values

`deviceType` – a numeric value indicating what kind of LiveTrack device was initialised:

`-100` - ERROR: LiveTrack device not found!

`1` - LiveTrack FM initialised.

`2` - LiveTrack AP initialised.

`3` - LiveTrack AV initialised.

`4` - LiveTrack Presto initialised.

4.1.2 LiveTrack.GetFirmwareVersion

This function returns the version of the firmware currently running on the LiveTrack.

Python Syntax

```
fwversion = LiveTrack.GetFirmwareVersion()
```

Parameters

No input parameters required.

Return values

`fwversion` **-** An integer representing the firmware version number.

4.1.3 LiveTrack.GetLibraryVersion

This function returns the version number of the LiveTrack library. This function will return an error if LiveTrack has not been initialized (using the LiveTrack.Init command).

Python Syntax

```
libversion = LiveTrack.GetLibraryVersion()
```

Parameters

No input parameters required.

Return values

`libversion` - An integer representing the LiveTrack library version number.

4.1.4 LiveTrack.GetSerialNumber

This function returns the unique serial number of the currently connected LiveTrack. This function will return an error if LiveTrack has not been initialized (using the LiveTrack.Init command).

Python Syntax

```
serialno = LiveTrack.GetSerialNumber()
```

Parameters

No input parameters required.

Return values

`serialno` - An string representing the unique serial number for the LiveTrack device.

4.1.5 LiveTrack.GetCaptureConfig

This function returns the video capture configuration.

Python Syntax

```
width, height, rate, offsetX, offsetY = LiveTrack.GetCaptureConfig()
```

Parameters

No input parameters required.

Return values

`width` - width (in pixels) of the video capture image window
`height` - height (in pixels) of the video capture image window
`rate` - frame rate of the video capture image window
`offsetX` - horizontal offset (in pixels) of the video capture image window
`offsetY` - vertical offset (in pixels) of the video capture image window.

4.1.6 LiveTrack.GetTracking

This function returns the tracking state of the left and right eyes (see also LiveTrack.SetTracking.)

Python Syntax

```
trackLeftEye, trackRightEye = LiveTrack.GetTracking()
```

Parameters

No input parameters required.

Return values

`trackLeftEye` - returns `true` (or `false`) if left eye tracking state is set (not set)
`trackRightEye` - returns `true` (or `false`) if right eye tracking state is set (not set).

4.1.7 LiveTrack.SetTracking

This function sets the tracking states for the left and right eyes. (See also LiveTrack.GetTracking.)

Python Syntax

```
err = LiveTrack.SetTracking(trackLeftEye, trackRightEye)
```

Parameters

`trackLeftEye` - set to true (or false) if left eye is to be tracked (not tracked)
`trackRightEye` - set to true (or false) if right eye is to be tracked (not tracked)

Return values

`err` - An integer where `0` indicates eye tracking has been successfully set.

4.1.8 LiveTrack.SetPupilCalibration

This function sets the pupil size scale factor. It is meant to be used after raw data has been obtained while tracking an artificial pupil of a known diameter (e.g. the M2005 Pupil Calibration Stick supplied with your LiveTrack system).

Python Syntax

```
err = LiveTrack.SetPupilCalibration(diameter, pixels)
```

Parameters

`diameter` - the size (in millimetres) of the artificial pupil

`pixels` - the major axis of the tracked artificial pupil in image pixels

Return values

`err` - An integer where `0` indicates pupil scale calibration is successfully set.

4.1.9 LiveTrack.GetPupilCalibration

This function returns the video image pixels-to-millimetre scaling factor. This can be modified by using LiveTrack.SetPupilCalibration.

Python Syntax

```
pixelsToMM = LiveTrack.GetPupilCalibration()
```

Parameters

No input parameters required.

Return values

`pixelsToMM` - The pupil size scale factor of in pixels per millimetre.

4.1.10 LiveTrack.SetCalibration

This function sets the gaze calibration parameters. These parameters can be obtained from LiveTrack.GetCalibration after a successful calibration and call to LiveTrackCalibrateDevice.

Python Syntax

```
err = LiveTrack.SetCalibration(eye, cal, viewDist, xGlintMedian,
yGlintMedian)
```

Parameters

`eye` – Integer specifying which eye you wish to set calibration for. `0` = left eye, `1` = right eye

`cal` - a 16 entry array containing the gaze calibration parameters.

`viewDist` - viewing distance in millimeters

`xGlintMedian` - median glint position during calibration (for head movement compensation)

`yGlintMedian` - median glint position during calibration (for head movement compensation)

Return values

`err` - An integer where `0` indicates calibration is successfully set.

4.1.11 LiveTrack.GetCalibration

This function gets the gaze calibration parameters. This can be done after a successful calibration and call to LiveTrackCalibrateDevice. The parameters can then be re-aplplied using LiveTrack.GetCalibration.

Python Syntax

```
cal, viewDist, xGlintMedian, yGlintMedian =
LiveTrack.GetCalibration(eye)
```

Parameters

`eye` – Integer specifying which eye you wish to get calibration for. `0` = left eye, `1` = right eye

Return values

`cal` - a 16 entry array containing the gaze calibration parameters.

`viewDist` - viewing distance in millimeters

`xGlintMedian` - median glint position during calibration (for head movement compensation)

`yGlintMedian` - median glint position during calibration (for head movement compensation).

4.1.12 LiveTrack.SetResultsTypeRaw

This function instructs LiveTrack to send raw data, rather than calibrated data. Raw data consists of the x and y coordinates of the centre location of the pupil and glint in the camera image. In dual glint systems the geometric average will be returned. The units will be retuned in pixels of the camera sensor. The pupil ellipse fit returns the major and minor axes in camera pixels.

Python Syntax

```
LiveTrack.SetResultsTypeRaw()
```

Parameters

No input parameters required.

Return values

`err` - An integer where `0` indicates the results type was successfully set.

4.1.13 LiveTrack.SetResultsTypeCalibrated

This function instructs LiveTrack to send calibrated data, rather than raw data. Calibrated data consists of the x, y and z coordinates of the gaze vector, where x and y are in the plane of the screen and z is the viewing distance. Units are typically in millimetres but can be calibrated to any units. The units will those used for 'targets' when calibrating with LiveTrack.CalibrateDevice.

The pupil ellipse fit major and minor axis will be scaled according to the scale factor set by LiveTrack.SetPupilCalibration (`pixelsToMM` obtained from LiveTrack.GetPupilCalibration) or if not set, then a default calibration will used.

Python Syntax

```
err = LiveTrack.SetResultsTypeCalibrated()
```

Parameters

No input parameters required.

Return values

`err` - An integer where `0` indicates the results type was successfully set.

4.1.14 LiveTrack.CalibrateDevice

This function calibrates the device for one designated eye. This function will return the sum of square of displacement errors between the target positions and the corresponding calibrated gaze positions. This function will return -100 if LiveTrack has not been initialized (using the crsLiveTrackInit command).

Python Syntax

```
calErr = LiveTrack.CalibrateDevice(eye, numberOfFixationTargets,
targetsX, targetsY, vectX, vectY, viewDist, xGlintMedian=0,
yGlintMedian=0)
```

Parameters

`eye` – integer specifying which eye you wish to calibrate. `0` = left eye, `1` = right eye

`numberOfFixationTargets` – integer specifying the number of calibration targets used.

`targetsX` – an array containing the x screen positions of the calibration targets. This will contain one entry for each calibration target, in mm.

`targetsY` – an array containing the y screen positions of the calibration targets. This will contain one entry for each calibration target, in mm.

`vectX` – an array containing the median distance (projected in the x direction) between pupil and glint centre, from raw data obtained while viewing the corresponding target.

`vectY` – an array containing the median distance (projected in the y direction) between pupil and glint centre, from raw data obtained while viewing the corresponding target.

`viewDist` – The viewing distance between eye tracker and monitor in mm.

`xGlintMedian` – an array containing the median glint position obtained while viewing the corresponding target (for head movement compensation, optional).

`yGlintMedian` – an array containing the median glint position obtained while viewing the corresponding target (for head movement compensation, optional).

Return values

`calErr` - The calibration error representing the sum of square of displacement errors between the target positions and their corresponding calibrated gaze positions.

Example (See also LiveTrack-PsychoPy-Calibration.py for a complete calibration example)

```
eye = 0 # left eye
numberOfFixationTargets = 9
targetsX = [-87.2753246410879,0,87.2753246410879,-
87.2753246410879,0,87.2753246410879,-
87.2753246410879,0,87.2753246410879]
targetsY = [-87.2753246410879,-87.2753246410879,-
87.2753246410879,0,0,0,87.2753246410879,87.2753246410879,87.275324641
0879 ]
vectX = [15.3326416015625,7.11785888671875,-
0.772888183593750,14.8192138671875,6.92382812500000,-
0.515563964843750,15.0279541015625,7.54492187500000,-
0.210876464843750]
vectY = [-29.0014648437500,-28.1158447265625,-27.4469604492188,-
20.2912597656250,-20.1955566406250,-19.9519042968750,-
12.6470947265625,-12.0270996093750,-11.9754028320313 ]
viewDist = 500 # viewing distance in mm
LiveTrack.CalibrateDevice(eye, numberOfFixationTargets, targetsX,
targetsY, vectX, vectY, viewDist)
```

4.1.15 LiveTrack.SaveCalibration

This function saves the gaze and pupil size calibration to a file. (See also LiveTrack.LoadCalibration).

Python Syntax

```
LiveTrack.SaveCalibration(filename)
```

Parameters

`filename` - the filename to save to including the path

Return values

`err` - An integer where `0` indicates the calibration file has been saved successfully

Example

```
LiveTrack.SaveCalibration("test.ltc")
```

4.1.16 LiveTrack.LoadCalibration

This function loads the gaze and pupil size calibration from a file. (See also LiveTrack.SaveCalibration).

Python Syntax

```
LiveTrack.LoadCalibration(filename)
```

Parameters

`filename` - the filename to load from including the path

Return values

`err` - An integer where `0` indicates the calibration file has been loaded successfully

Example

```
LiveTrack.LoadCalibration("test.ltc")
```

4.1.17 LiveTrack.SetDataFilename

When this function is called, subsequent tracking data is written to the specified file in CSV format as it is captured until LiveTrack.CloseDataFile is called. If tracking is stopped and restarted, data will continue to be appended to the open file. If this function is called with a different filename to one that has already been opened this way, the first file is closed and the new one opened.

Python Syntax

```
LiveTrack.SetDataFilename(filename)
```

Parameters

`filename` - the name of the file to log data to, from including the path**.**

Return values

`err` - An integer where `0` indicates the logging was successfully started.

Example

```
LiveTrack.SetDataFilename('LiveTrack-PsychoPy-CSV-Demo-Data.csv')
```

4.1.18 LiveTrack.StartTracking

This function starts eye tracking and starts recording results to the data buffer.

Python Syntax

```
err = LiveTrack.StartTracking()
```

Parameters

No input parameters required.

Return values

`err` - An integer where `0` indicates buffering was successfully started.

4.1.19 LiveTrack.StopTracking

This function stops eye tracking and stops recording results to the data buffer.

Python Syntax

```
err = LiveTrack.StopTracking()
```

Parameters

No input parameters required.

Return values

`err` - An integer where `0` indicates buffering was successfully stopped.

4.1.20 LiveTrack.CloseDataFile

This function stops any further date being written to the file opened with LiveTrack.SetDataFilename, and closes the file.

Python Syntax

```
LiveTrack.CloseDataFile()
```

Parameters

No input parameters required.

Return values

`err` - An integer where `0` indicates the file was successfully closed.

4.1.21 LiveTrack.SetDataComment

This function allows text strings to be written into the CSV data file opened with LiveTrack.SetDataFilename. These comments are written with the next sample as soon as possible and appear in the rightmost field. They can be used to insert, for example, trial or stimulus related event information.

Python Syntax

```
err = LiveTrack.SetDataComment(comment)
```

Parameters

`comment` - the string to insert into the data as a comment

Return values

`err` - An integer where `0` indicates the comment was successfully set

Example

```
LiveTrack.SetDataComment('Trial 1. Contrast 0.5')
```

4.1.22 LiveTrack.GetResultsCount

This function returns the number of results (samples) currently stored in the LiveTrack results buffer.

Python Syntax

```
noOfResults = LiveTrack.GetResultsCount()
```

Parameters

No input parameters required.

Return values

`noOfResults` - An integer representing the number of eyetracking results currently held in the results buffer.

4.1.23 LiveTrack.GetLastResult

This function returns the most recent eye position analysed by the LiveTrack library. This data point is NOT removed from the data buffer that the LiveTrack library

maintains. (i.e. you can get the same results later using LiveTrack.GetBufferedEyePositions.
If there are no points available in the buffer, a structure filled with zeros is returned.

Python Syntax

```
data = LiveTrack.GetLastResult()
```

Parameters

No input parameters required.

Return values

`data` – a structure containing the latest data sample. The structure has the following form:
`data.Size` - size of entire structure (bytes)
`data.Timestamp` - video frame time stamp (microseconds)
`data.ResultsType` – `0` = uncalibrated or `1` = calibrated data
`data.DigitalIO` - -- input / output trigger data (see manual for details)
`data.FixationDetected` - `0` = fixation detected or `1` = fixation not detected
The following fields apply to the left eye:
`data.ActiveROIs` – this feature will be added in a future update
`data.GlintX` - raw glint X position (pixels) when streaming raw data
`data.GlintY` - raw glint Y position (pixels) when streaming raw data
`data.PupilX` - raw pupil X position (pixels) when streaming raw data
`data.PupilY` - raw pupil Y position (pixels) when streaming raw data
`data.PupilMajorAxis` - pupil major axis length
`data.PupilMinorAxis` - pupil minor axis length
`data.VectX` - distance between pupil and glint centres (pixels)
`data.VectY` - distance between pupil and glint centres (pixels)
`data.GazeX` - calibrated gaze X position (mm)
`data.GazeY` - calibrated gaze Y position (mm)
`data.GazeZ` - calibrated gaze Z position (mm)
`data.GazeAzimuth` - calibrated gaze Helmholtz azimuth (degrees)
`data.GazeElevation` - gaze Helmholtz elevation (degrees)
`data.GazeLongitude` - calibrated gaze Fick logintude (degrees)
`data.GazeLatitude` - calibrated gaze Fick latitude (degrees)
`data.Tracked` - `0` = eye tracked or `1` = not tracked
`data.Enabled` - `0` = eyetracking enabled or `1` = not enabled
`data.Calibrated - 0` = calibrated or `1` = not calibrated
`data.Dropped` - `0` = eyetracking dropped or `1` = not dropped
`data.ROI` - this feature will be added in a future update
The following fields apply to the right eye, and have the same function as the corresponding fields above without the "`Right`" suffix:

```
data.ActiveROIsRight
data.GlintXRight
data.GlintYRight
data.PupilXRight
data.PupilYRight
data.PupilMajorAxisRight
data.PupilMinorAxisRight
data.VectXRight
```

```
data.VectYRight
data.GazeXRight
data.GazeYRight
data.GazeZRight
data.GazeAzimuthRight
data.GazeElevationRight
data.GazeLongitudeRight
data.GazeLatitudeRight
data.TrackedRight
data.EnabledRight
data.CalibratedRight
data.DroppedRight
data.ROIRight
```

4.1.24 LiveTrack.GetBufferedEyePositions

This function returns eye position data from the buffer that is created and maintained by the LiveTrack library. Eye positions that are returned by this function are (by default) removed from the data buffer, so subsequent calls will not return duplicate data.

Data are returned as an array of the same kind of structures used by LiveTrack.GetLastResult (see section 4.1.23) If there are no points available in the buffer, an empty array is returned.

This function has three optional parameters, a Boolean/logical value: `removeFromBuffer`, a numeric (integer) value: `maximumPoints` and a numeric (integer) value: `fromBeginning`. If the `removeFromBuffer` parameter is set to `0`, then data points are left in the buffer, allowing them to be retrieved again by a second call. If the `removeFromBuffer` parameter is set to `1`, then data points are deleted from the buffer after they are read. The `maximumPoints` parameter allows you to limit the number of points returned by this function. This is useful if the buffer is large and you can only deal with a limited number of points at any one time. If the `maximumPoints` parameter is used then the `fromBeginning` parameter determines whether to retrieve the earliest (`1`, default) or most recent (not `1`) data in the buffer.

Python Syntax

```
data = LiveTrack.GetBufferedEyePositions(removeFromBuffer=1,
maximumPoints=-1, fromBeginning=1)
```

Parameters

`removeFromBuffer` – `1` (default)=remove the retrieved data from the buffer so that it will not be retrieved again on the next call, or `0` = do not remove the data from the buffer.

`maximumPoints`- an integer specifying the maximum number of samples to get from the buffer.

`fromBeginning`- `1` (default)=remove

Return values

`data` – an array of structures containing the data. The structure same form as the single one returned by LiveTrack.GetLastResult (see section 4.1.23).

4.1.25 LiveTrack.ClearDataBuffer

This function clears the LiveTrack data buffer.

Python Syntax

```
LiveTrack.ClearDataBuffer()
```

Parameters

No input parameters required.

Return values

`err` - An integer where `0` indicates the buffer has been successfully cleared.

4.1.26 LiveTrack.Close

This function closes the connection to the LiveTrack. This function returns 0 if successful.

Python Syntax

```
err = LiveTrack.Close()
```

Parameters

No input parameters required.

Return values

`err` - An integer where `0` indicates LiveTrack has been successfully closed.

5 Revision History

Document version	Date	Document change
1	2020-11-19	First version in CMS. Split library reference for C and Matlab from exiting LiveTrack Lightning manual. Removed unnecessary “function” prefixes from all Matlab function syntax.
2	2021-02-08	Combined C, Matlab, Python references into one document. Created Python reference from existing Python demo files and descriptions from Matlab.