



QuickLink API

User's Manual

Software Version: 5.0

Last Revised: 14 July 2008

Table of Contents

1.0 Structures	1
1.1 ClickingOptions	1
1.2 Calibration Options	5
1.3 Processing Options	6
1.4 Camera Options	10
1.5 Toolbar Options.....	12
2.0 Functions.....	13
2.1 Informational Functions.....	13
2.2 Quick Glance UI Manipulation Functions	14
2.3 Data Capture Functions.....	17
2.4 Calibration Functions.....	21
2.5 Options Access Functions.....	29
2.6 Camera Functions.....	34
3.0 Index.....	37

1.0 Structures

1.1 ClickingOptions

Description:

These options control how clicking is performed. The current values can be retrieved using **GetClickingOptions()**. New values can be set by using **SetClickingOptions()**.

Blink_BothEyesRequired

bool Blink_BothEyesRequired;

Description:

Both eyes must blink for a click to be performed. Inadvertent clicks can be reduced by making this value true.

Possible Values:

true
false

Blink_CancelTime

unsigned int Blink_CancelTime;

Description:

If **Blink_EnableSecondaryClick** is true then the user has this amount of time in tenths of seconds to open their eyes after the **Blink_PrimaryTime** plus the **Blink_SecondaryTime** has been reached in order to perform a secondary click. If the user keeps their eyes closed longer than the **Blink_PrimaryTime** plus the **Blink_SecondaryTime** plus this time, then the blink will be voided and no click will be performed.

If **Blink_EnableSecondaryClick** is false then this time is the amount of time a user has after the **Blink_PrimaryTime** to open their eyes and register a primary click. If the user keeps their eyes closed longer than the **Blink_PrimaryTime** plus this time then the blink will be voided and no click will be performed.

Possible Values:

1 - 20

Blink_EnableSecondaryClick

bool Blink_EnableSecondaryClick;

Description:

If this is false then the Blink_SecondaryTime has no effect and the Blink_CancelTime acts as the timeout time for the primary click.

If this is true then Secondary Clicking is enabled and Blink_CancelTime is the timeout time for the secondary click.

Possible Values:

true
false

Blink_PrimaryTime

unsigned int Blink_PrimaryTime;

Description:

The time in tenths of seconds the eye must be closed to register a click.

If a user does not have their eyes closed for at least this amount of time, then no click will be registered.

Possible Values:

1 - 20

Blink_SecondaryTime

unsigned int Blink_SecondaryTime;

Description:

The user has this amount of time in tenths of seconds to open their eyes after the **Blink_PrimaryTime** has been reached in order to perform a primary click. If the user keeps their eyes closed longer than the **Blink_PrimaryTime** plus this time, then they are able to do a secondary click. This value is used only when **Blink_EnableSecondaryClick** is true.

Possible Values:

0 - 20

Blink_VisualFeedback

bool Blink_VisualFeedback;

Description:

Draw visual indicators on the screen when blink clicking.

Possible Values:

true
false

Click_AudibleFeedback

bool Click_AudibleFeedback;

Description:

This enables audible feedback with blink or dwell click.

Possible Values:

true
false

Click_Delay

unsigned int Click_Delay;

Description:

The amount of time in milliseconds between when a blink or dwell click is initiated and when it is actually done. This parameter is not usually changed. It is mainly used for troubleshooting.

Possible Values:

0 - 1000

Click_Method

ClickMethod Click_Method;

Description:

The type of software clicking detected by Quick Glance. This determines whether a blink or a dwell action will post a click event.

Possible Values:

Click_Method_None
Click_Method_Blink
Click_Method_Dwell

Click_ZoomFactor

unsigned int Click_ZoomFactor;

Description:

The magnification power used when the zoom tool or the click confirm tool are being used.

Possible Values:

2 - 10

Dwell_BoxSize

unsigned int Dwell_BoxSize;

Description:

This is the width and the height in millimeters of a box in which the cursor must reside in order to register a dwell click.

Possible Values:

5 - 50

Dwell_Time

unsigned int Dwell_Time;

Description:

The time in tenths of seconds in which the cursor must stay within the **DwellBoxSize** in order to perform a click.

Possible Values:

2 - 50

1.2 Calibration Options

Description:

These options control how the calibration is performed. The current values can be retrieved using **GetCalibrationOptions()**. A calibration must be initialized after new values are set for the new values to take effect.

Calibration_Style

CalibrationStyle Calibration_Style;

Description:

This represents the number of targets to be used during calibration. More calibration targets will make the duration of the entire calibration process longer, but will also usually result in a more accurate and reliable calibration.

Possible Values:

Cal_Style_5_Point
Cal_Style_9_Point
Cal_Style_16_Point

Calibration_TargetTime

unsigned int Calibration_TargetTime;

Description:

The time in tenths of seconds each target will take to calibrate. A calibration is less prone to have errors if this time is longer.

Possible Values:

2 - 100

1.3 Processing Options

Description:

These options control how certain aspects of the processing are done. The current values can be retrieved by using **GetCalibrationOptions()**. New values can be set by using **SetCalibrationOptions()**.

Processing_EyesToProcess

EyesToProcess Processing_EyesToProcess;

Description:

This determines whether the right and/or left eye will be tracked. This also determines which eyes will be calibrated when a calibration is performed.

If **EYES_TO_PROC_SINGLE_LEFT** or **EYES_TO_PROC_SINGLE_RIGHT** is selected then only the left or right eye, respectively, is tracked.

If **EYES_TO_PROC_DUAL_LEFT_OR_RIGHT** is selected, then both eyes will be tracked, but tracking will still work if only one eye is in the image.

If **EYES_TO_PROC_DUAL_LEFT_AND_RIGHT** is selected, then tracking will not work unless both eyes are found in the image.

EYES_TO_PROC_DUAL_LEFT_OR_RIGHT is the most common processing method.

Possible Values:

EYES_TO_PROC_SINGLE_LEFT
EYES_TO_PROC_SINGLE_RIGHT
EYES_TO_PROC_DUAL_LEFT_OR_RIGHT
EYES_TO_PROC_DUAL_LEFT_AND_RIGHT

Processing_EnableCapture

bool Processing_EnableCapture;

Description:

This enables image capturing from the camera. If this is false then processing will be performed on the last image that was received.

Possible Values:

true
false

Processing_EnableClicking

bool Processing_EnableClicking;

Description:

This enables the processing of dwell and blink clicks.

Possible Values:

true
false

Processing_EnableCursorMovement

bool Processing_EnableCursorMovement;

Description:

This enables cursor movement according to gaze point. **SetEnableEyeControl()** and **ResetEnableEyeControl()** have no effect if this value is false.

Possible Values:

true
false

Processing_EnableDisplay

bool Processing_EnableDisplay;

Description:

This enables the displaying of the image within Quick Glance user interface.

Possible Values:

true
false

Processing_EnableProcessing

bool Processing_EnableProcessing;

Description:

This enables processing of the image currently in the processing buffer.

Possible Values:

true
false

Processing_MaxProcessTime

unsigned int Processing_MaxProcessTime;

Description:

This is the maximum amount of time in milliseconds the process will use without giving up some of it's time slice to other processes. This is mostly useful on slower machines where the eye tracker is a processing burden. On slower machines this value should be set lower.

Possible Values:

5 - 500

Processing_ProcessPriority

ProcessPriority Processing_ProcessPriority;

Description:

This is the priority the eye tracker has on the system. Higher values will increase the priority.

Possible Values:

Proc_Priority_0
Proc_Priority_1
Proc_Priority_2
Proc_Priority_3

Processing_SmoothingFactor

unsigned int Processing_SmoothingFactor;

Description:

The amount of jitter reduction performed on the gaze-point received. The effectiveness of this value is affected by the frame rate of the camera. Higher numbers produce less jitter, but also introduce e more lag. The amount of lag in seconds can be determined by the formula:

$$lag = \frac{smoothing_factor}{actual_frame_rate} / 2$$

Possible Values:

1 - 100

1.4 Camera Options

Description:

These options control camera operation. The current values can be retrieved by using **GetCamerOptions()**. New values can be set by using **SetCameraOptions()**.

Camera_BusBandwidthPercentage

unsigned int Camera_BusBandwidthPercentage;

Description:

This is the percentage of bus bandwidth used by the camera. This value directly affects the frame rate of the camera. Smaller values result in a lesser amount of pixel throughput resulting in a smaller number of total frames that can be transferred per second.

Possible Values:

10 - 100

Camera_GainMethod

CameraGainMethod Camera_GainMethod;

Description:

This determines the gain method used for the camera. This should normally be set to **CAM_GAIN_METHOD_AUTO**.

Possible Values:

CAM_GAIN_METHOD_AUTO
CAM_GAIN_METHOD_MANUAL

Camera_GainValue

double Camera_GainValue;

Description:

This is the gain value used when the gain method is set to **CAM_GAIN_METHOD_MANUAL**.

Possible Values:

0 - 100

Camera_GPIO_1; Camera_GPIO_2; Camera_GPIO_3

```
CameraGPIOOutput Camera_GPIO_1;  
CameraGPIOOutput Camera_GPIO_2;  
CameraGPIOOutput Camera_GPIO_3;
```

Description:

These values determine the output values of the GPIO pins on the camera board. If any of these values are set to **CAM_GPIO_OUT_CUSTOM** then the corresponding GPIO output value can be set by using the function **SetCustomGPIO()**. If any of these values are set to **CAM_GPIO_OUT_LEFT_TRACKING_STATUS** or **CAM_GPIO_OUT_RIGHT_TRACKING_STATUS** then the lock status of the left or right eye will be represented by the corresponding GPIO.

Possible Values:

```
CAM_GPIO_OUT_CUSTOM  
CAM_GPIO_OUT_LEFT_TRACKING_STATUS  
CAM_GPIO_OUT_RIGHT_TRACKING_STATUS
```

Camera_ImageROIPercentage

```
unsigned int Camera_ImageROIPercentage;
```

Description:

This is the percentage of the image height used when the eyes are being tracked. This value directly affects the frame rate of the camera. The smaller the value, the fewer the number of pixels that have to be transferred each frame resulting in more frames for a given bandwidth. If this value is too small then vertical head movement is limited because the eyes will move out of the region of interest before the camera has time to readjust.

Possible Values:

```
10 - 100
```

1.5 Toolbar Options

Description:

These options control how the Eye Tools toolbar looks. The current values can be retrieved by using **GetToolbarOptions()**. New values can be set by using **SetToolbarOptions()**.

Toolbar_ButtonSizeX

unsigned int Toolbar_ButtonSizeX;

Description:

The size in millimeters in the x-axis direction of the toolbar buttons.

Possible Values:

5 - 50

Toolbar_ButtonSizeY;

unsigned int Toolbar_ButtonSizeY;

Description:

The size in millimeters in the y-axis direction of the toolbar buttons.

Possible Values:

5 - 50

Toolbar_ImageDisplayType

ToolBarImageDisplay Toolbar_ImageDisplayType;

Description:

This determines the way the image is displayed in the toolbar. If **IMG_DISP_LIVE_IMAGE** is selected then the normal live image will be displayed in the toolbar. If **IMG_DISP_PSEUDO_IMAGE** is selected then circles representing the eyes will be displayed to show the user where their eyes are in the image.

Possible Values:

IMG_DISP_LIVE_IMAGE
IMG_DISP_PSEUDO_IMAGE

2.0 Functions

2.1 Informational Functions

Description

These functions are used to get general information from either Quick Glance or the Quick Link API.

GetQGOnFlag()

bool **GetQGOnFlag()**;

Description:

This function determines whether Quick Glance is running.

Return Value:

If Quick Glance is running then true is returned otherwise false is returned.

See Also:

ExitQuickGlance()

GetSDKVersion()

double **GetSDKVersion()**;

Description:

This function gets the API version number.

Return Value:

The API version number is returned.

2.2 Quick Glance UI Manipulation Functions

Description

These functions are used to manipulate the Quick Glance user interface windows. They do not affect eye tracking performance.

ExitQuickGlance()

void ExitQuickGlance();

Description:

This function exits Quick Glance.

See Also:

GetQGOOnFlag()

HideLargeImage()

void HideLargeImage();

Description:

This function hides the large image if it is displayed. If it the large image is not displayed then this function does nothing.

See Also:

ToggleLargeImage()

ShowLargeImage()

MoveLeftRight()

void MoveLeftRight();

Description:

*When Quick Glance is in the **MWEyeTools** or the **MWSmallVideo** state, it will be positioned in any of the four corners of the screen. Calling this function will toggle it from the left side of the screen to the right side of the screen.*

See Also:

MoveUpDown()

SetMWState()

MoveUpDown()

void MoveUpDown();

Description:

When Quick Glance is in the **MWEyeTools** or the **MWSmallVideo** state, it will be positioned in any of the four corners of the screen. Calling this function will toggle it from the bottom of the screen to the top of the screen.

See Also:

MoveLeftRight()
SetMWState()

ResetEnableEyeControl()

void ResetEnableEyeControl();

Description:

This function disables the cursor control within Quick Glance. After this function is called the cursor will not move with the users gaze.

See Also:

SetEnableEyeControl()

SetEnableEyeControl()

void SetEnableEyeControl();

Description:

This function enables the cursor control within Quick Glance. After this function is called the cursor will move with the users gaze as long as that user is calibrated.

See Also:

ResetEnableEyeControl()

SetMWState()

void SetMWState(int MWState);

Description:

This function sets the state of the Quick Glance user interface window.

Arguments:

MWState - The window state for Quick Glance.

Possible Values:

MWOpeningScreen

MWEyeTools

MWSmallVideo

MWHidden

See Also:

MoveLeftRight()

MoveUpDown()

ToggleLargeImage()

ShowLargeImage()

void ShowLargeImage();

Description:

This function will show the large image. If the large image is already displayed then this function does nothing.

See Also:

ToggleLargeImage()

HideLargeImage()

ToggleLargeImage()

void ToggleLargeImage();

Description:

This function toggles the display of the large image. If the large image is not already displayed then this function will display it. If the large image is already displayed then this function will hide it.

See Also:

HideLargeImage()

ShowLargeImage()

2.3 Data Capture Functions

Description

These functions are used for eye tracking data acquisition. The data acquisition can be done in real-time or buffered over a period of time.

QueryBulkCapture()

```
bool QueryBulkCapture(  
    unsigned int & NumCaptured,  
    unsigned int & NumNotCaptured,  
    unsigned int & NumRead,  
    unsigned int & NumNotRead,  
    bool & Capturing);
```

Description:

This function retrieves the status of data capturing started by a call to **StartBulkCapture()**.

Arguments:

NumCaptured - The number of data points that have been captured since the data capturing was started.

NumNotCaptured - The number of data points that still have to be captured before capturing will automatically terminate. This value plus **NumCaptured** sum to the value passed in to **StartBulkCapture()**.

NumRead - The number of data points that have been read by calls to **ReadBulkCapture()**.

NumNotRead - The number of data points that have been collected but remain to be read by calls to **ReadBulkCapture()**. This value plus **NumRead** sum to the value **NumCaptured**.

Capturing - The status of the data capturing. True indicates that data points are still being collected and false indicates that no more data points will be collected.

Return Value:

The success of the function. If false then the retrieved values are not valid.

See Also:

StartBulkCapture()

StopBulkCapture()

ReadBulkCapture()

ReadBulkCapture()

```
bool ReadBulkCapture(  
    unsigned long MaxTimeout,  
    ImageData * Data);
```

Description:

This function is used to read data that was captured by a call to **StartBulkCapture()**. Capturing does not need to be finished before this function can be called. The number of captured data points that are available at any given time can be determined by calling **QueryBulkCapture()**.

Arguments:

MaxTimeout - The amount of time in milliseconds to wait for a new data point to be received if there is not one available.

Data - The next available data point to be read.

Return Value:

The success of the function. If false, then the data received is invalid.

See Also:

StartBulkCapture()
StopBulkCapture()
QueryBulkCapture()

RegisterClickEvent()

```
void RegisterClickEvent(  
    void * WindowHandle,  
    DWORD PrimaryMessage,  
    DWORD SecondaryMessage);
```

Description:

The function allows a window to be registered to receive the primary and/or secondary click event. If no window handle is registered or if the window handle is **NULL** then Quick Glance will perform its usual primary or secondary click function. Likewise if **NULL** is passed in for either the primary or secondary message then Quick Glance will perform its normal function for the corresponding click. For example if the primary message is **NULL** but the secondary message is not then the provided window handle will only receive the secondary click message and Quick Glance will perform the primary click.

Arguments:

WindowHandle - The handle to the window where the message will be posted. To unregister a previously registered window, make this argument **NULL**.

PrimaryMessage - The user-defined message to be posted when a primary click has been initiated by the eye tracker.

SecondaryMessage - The user-defined message to be posted when a secondary click has been initiated by the eye tracker.

StartBulkCapture()

**void StartBulkCapture(
unsigned int NumPoints);**

Description:

This function starts the collecting of tracking data. It buffers the data so the desired number of gaze points can be collected, then the points can be read back one at a time after collection is finished. Data retrieval does not have to wait until collection has finished. If data collection is in process due to a previous call to this function then a new call to this function has no effect. **StartRawCapture()** must be called prior to calling this function otherwise the collected data will not be valid. When data is being captured due to a call to this function then data received by a call to **GetImageData()** will not be valid.

Arguments:

NumPoints - The number of points to be collected. Data collection will stop when this number of data points has been received or if **StopBulkCapture()** has been called.

See Also:

StopBulkCapture()
QueryBulkCapture()
ReadBulkData()
StartRawCapture()
GetImageData()

StopBulkCapture()

void StopBulkCapture();

Description:

This stops the collection of tracking data. If data is not being collected by a call to **StartBulkCapture()** then this function has no effect. This function does not need to be called before data can be read by calling **ReadBulkCapture()**. This function also does not need to be called for data capturing to terminate. This function only needs to be called if the data capturing needs to be stopped prior to capturing the number of data points specified by the corresponding call to **StartBulkCapture()**.

See Also:

StartBulkCapture()
QueryBulkCapture()
ReadBulkCapture()

StartRawCapture()

```
void StartRawCapture(  
    bool CopyImage);
```

Description:

This function starts real-time data capturing.

Arguments:

CopyImage - This determines whether or not the pixel data of the current image is copied into shared memory. If true then the image's pixel data will be accessible. If false then the image's pixel data will not be accessible. If access to the image's pixel data is not required then it is preferable to not copy the pixel data in order to save processing time.

See Also:

StopRawCapture()
GetImageData()
StartBulkCapture()

StopRawCapture()

```
void StopRawCapture();
```

Description:

This function stops the capturing of real-time image data.

See Also:

StartRawCapture()

2.4 Calibration Functions

Description:

These functions control the active calibration.

ApplyCalibrationEx()

CalibrationErrorEx ApplyCalibrationEx();

Description:

This function applies the new calibration. The new calibration only replaces the old calibration for the newly calibrated eye(s). If only one eye was newly calibrated and the other eye had a previous calibration, its calibration will be untouched. The calibration process must be finished before this function can be called.

Return Value:

CALIBRATIONEX_NOT_INITIALIZED - The calibration has not been initialized.

CALIBRATIONEX_NOT_ALL_TARGETS_CALIBRATED - The calibration has not completed.

CALIBRATIONEX_INTERNAL_TIMEOUT - It is likely that Quick Glance is not running.

CALIBRATIONEX_OK - The function completed successfully.

See Also:

InitializeCalibrationEx()

CalibrateEx()

CalibrateEx()

**CalibrationErrorEx CalibrateEx(
int TargetHandle);**

Description:

This function calibrates the target with the corresponding target handle. This function should be called after the target is displayed.

Arguments:

TargetHandle - The handle retrieved by calling **GetNewTargetPositionEx()** or **GetWorstTargetPositionEx()**.

Return Value:

CALIBRATIONEX_NOT_INITIALIZED - The calibration has not been initialized.

CALIBRATIONEX_INVALID_TARGET_HANDLE - The target handle does not correspond to any target.

CALIBRATIONEX_NO_EYE_FOUND - No eyes were found. This could also mean that some of the processing options are not enabled correctly.

CALIBRATIONEX_LEFT_EYE_NOT_FOUND - The left eye was not found.

CALIBRATIONEX_RIGHT_EYE_NOT_FOUND - The right eye was not found.

CALIBRATIONEX_INTERNAL_TIMEOUT - It is likely that Quick Glance is not running.

CALIBRATIONEX_OK - The function completed successfully.

See Also:

InitializeCalibrationEx()

GetNewTargetPositionEx()

GetWorstTargetPositionEx()

CalibrationBiasEx()

```
void CalibrationBiasEx(  
    int DeltaX,  
    int DeltaY);
```

Description:

This function can be used to introduce a bias in the current calibration. This is useful in situations when the user has a good calibration score but the cursor may be off a little bit in one direction. This can be used to correct that offset. This function will not change the user's calibration score. **InitializeCalibrationEx()** does not need to be called before this function can be called. This function applies the bias to the current active calibration.

Arguments:

DeltaX -The offset in the X direction. This is calculated by the formula:

$$\mathbf{DeltaX} = X1 - X2$$

X1 = (X value of gaze point retrieved from the eye tracker)

X2 = (X value of the displayed target at which the user is looking)

DeltaY -The offset in the Y direction. This is calculated by the formula:

$$\mathbf{DeltaY} = Y1 - Y2$$

Y1 = (Y value of gaze point retrieved from the eye tracker)

Y2 = (Y value of the displayed target at which the user is looking)

GetNewTargetPositionEx()

**CalibrationErrorEx GetNewTargetPositionEx(
int & X,
int & Y,
int & TargetHandle);**

Description:

This function retrieves the position and identification handle for the next target to be calibrated.

Arguments:

X - The X position in screen coordinates of where the focal point of the target should be placed.

Y - The Y position in screen coordinates of where the focal point of the target should be placed.

TargetHandle - A handle used to identify the target corresponding to the retrieved position. This handle is used for **CalibrateEx()**.

Return Value:

CALIBRATIONEX_NOT_INITIALIZED - The calibration has not been initialized.

CALIBRATIONEX_NO_NEW_TARGETS - All targets have been calibrated.

CALIBRATIONEX_INTERNAL_TIMEOUT - It is likely that Quick Glance is not running.

CALIBRATIONEX_OK - The function completed successfully.

See Also:

InitializeCalibrationEx()

CalibrateEx()

GetScoreEx()

**CalibrationErrorEx GetScoreEx(
double & left,
double & right);**

Description:

This function retrieves the score for a finished calibration. The calibration process must be finished before this function can be called.

Arguments:

Left - The score written to **Left** is only valid if the left eye was being calibrated.

Right - The score written to **Right** is only valid if the right eye was being calibrated.

Return Value:

CALIBRATIONEX_NOT_INITIALIZED - The calibration has not been initialized.

CALIBRATIONEX_NOT_ALL_TARGETS_CALIBRATED - The calibration has not completed.

CALIBRATIONEX_INTERNAL_TIMEOUT - It is likely that Quick Glance is not running.

CALIBRATIONEX_OK - The function completed successfully.

See Also:

InitializeCalibrationEx()

CalibrateEx()

GetWorstTargetPositionEx()

**CalibrationErrorEx GetWorstTargetPositionEx(
int & X,
int & Y,
int & TargetHandle);**

Description:

This function retrieves the position and identification handle for the worst target that was calibrated. The calibration must be finished before this function can be called.

Arguments:

X - The X position in screen coordinates of where the focal point of the target should be placed.
Y - The Y position in screen coordinates of where the focal point of the target should be placed.
TargetHandle - A handle used to identify the target corresponding to the retrieved position. This handle is used for **CalibrateEx()**.

Return Value:

CALIBRATIONEX_NOT_INITIALIZED - The calibration has not been initialized.
CALIBRATIONEX_NOT_ALL_TARGETS_CALIBRATED - Not all targets have been calibrated.
CALIBRATIONEX_INTERNAL_TIMEOUT - It is likely that Quick Glance is not running.
CALIBRATIONEX_OK - The function completed successfully.

See Also:

InitializeCalibrationEx()
GetNewTargetPositionEx()
CalibrateEx()

InitializeCalibrationEx()

**CalibrationErrorEx InitializeCalibrationEx(
unsigned int CalibrationIndex);**

Description:

This function initializes a calibration. It must be called before most other calibration functions can be called. This calibration process uses custom drawn targets. The eyes that will be calibrated are determined by the **ProcessingOptions**. The duration for each target and the style of calibration to be performed are determined by the **CalibrationOptions**.

Arguments:

CalibrationIndex - The desired calibration to calibrate. If the calibration does not already exist it will be created. The value can be any number that **unsigned int** will allow. Calibrations do not have to be created in any specific order nor do all numbers have to be used between the lowest value index and the highest value index. No additional disk space or memory is used by creating, for example, three different calibrations with indexes 100, 1050, and 21 as opposed to indexes 1, 2 and 3. To recall a saved calibration by its index use the function **OpenCalibrationEx()**.

Return Value:

CALIBRATIONEX_NO_EYE_SELECTED - The **Processing_EyesToProcess** parameter is set to an invalid value.

CALIBRATIONEX_INTERNAL_TIMEOUT - It is likely that Quick Glance is not running.

CALIBRATIONEX_OK - The function completed successfully.

See Also:

GetNewTargetPositionEx()
GetWorstTargetPositionEx()
CalibrateEx()
GetScoreEx()
ApplyCalibrationEx()
OpenCalibrationEx()

InternalCalibration1()

void InternalCalibration1();

Description:

This function initiates a calibration internally within Quick Glance. It uses the user interface within Quick Glance to perform the calibration. The calibration process is immediately started without user intervention, stopping only if there is an error and when the calibration completes to display the calibration score.

OpenCalibrationEx()

**CalibrationErrorEx OpenCalibrationEx(
unsigned int CalibrationIndex);**

Description:

This function opens a calibration previously performed. **InitializeCalibrationEx()** does not need to be called before this function can be called, nor does a successful calibration have to be completed prior to opening.

Arguments:

CalibrationIndex - The index of the calibration to be retrieved. If the index does not exist then the function will return **CALIBRATIONEX_INVALID_INDEX** and the calibration will be set to uncalibrated. If it is believed that the specified calibration did exist then it is likely that the user's calibration file was deleted or corrupted, in which case the calibration will need to be redone.

Return Value:

CALIBRATIONEX_INVALID_INDEX - The provided index could not be opened.

CALIBRATIONEX_INTERNAL_TIMEOUT - It is likely that Quick Glance is not running.

CALIBRATIONEX_OK - The function completed successfully.

See Also:

InitializeCalibrationEx()

2.5 Options Access Functions

Description:

These functions allow you to set or get any of the options in Quick Glance.

GetCalibrationOptions()

**bool GetCalibrationOptions(
CalibrationOptions & Options);**

Description:

This Function gets the calibration related settings.

Arguments:

Options - The options retrieved from the API. These options are only valid if true is returned.

Return Value:

If the function was successful then true is returned and the data in **Options** is valid. If unsuccessful, then false is returned.

See Also:

SetCalibrationOptions()

GetCameraOptions()

**bool GetCameraOptions(
CameraOptions & Options);**

Description:

This Function gets the camera related settings.

Arguments:

Options - The options retrieved from the API. These options are only valid if true is returned.

Return Value:

If the function was successful then true is returned and the data in **Options** is valid. If unsuccessful then false is returned.

See Also:

SetCameraOptions()

GetClickingOptions()

**bool GetClickingOptions(
ClickingOptions & Options);**

Description:

This Function gets the clicking related settings.

Arguments:

Options - The options retrieved from the API. These options are only valid if true is returned.

Return Value:

If the function was successful then true is returned and the data in **Options** is valid. If unsuccessful, then false is returned.

See Also:

SetClickingOptions()

GetProcessingOptions()

**bool GetProcessingOptions(
ProcessingOptions & Options);**

Description:

This Function gets the processing related settings.

Arguments:

Options - The options retrieved from the API. These options are only valid if true is returned.

Return Value:

If the function was successful then true is returned and the data in **Options** is valid. If unsuccessful then false is returned.

See Also:

SetProcessingOptions()

GetToolBarOptions()

**bool GetToolBarOptions(
ToolBarOptions & Options);**

Description:

This Function gets the toolbar related settings.

Arguments:

Options - The options retrieved from the API. These options are only valid if true is returned.

Return Value:

If the function was successful then true is returned and the data in **Options** is valid. If unsuccessful then false is returned.

See Also:

SetToolBarOptions()

SetCalibrationOptions()

**bool SetCalibrationOptions(
CalibrationOptions Options);**

Description:

This Function sets the calibration related settings.

Arguments:

Options - The options being sent to the API.

Return Value:

If the function was successful then true is returned. If unsuccessful then false is returned and the new options did not take effect.

See Also:

GetCalibrationOptions()

SetCameraOptions()

**bool SetCameraOptions(
CameraOptions Options);**

Description:

This Function sets the camera related settings.

Arguments:

Options - The options being sent to the API.

Return Value:

If the function was successful then true is returned. If unsuccessful then false is returned and the new options did not take effect.

See Also:

GetCameraOptions()

SetClickingOptions()

**bool SetClickingOptions(
ClickingOptions Options);**

Description:

This Function sets the clicking related settings.

Arguments:

Options - The options being sent to the API.

Return Value:

If the function was successful then true is returned. If unsuccessful then false is returned and the new options did not take effect.

See Also:

GetClickingOptions()

SetProcessingOptions()

**bool SetProcessingOptions(
ProcessingOptions Options);**

Description:

This Function sets the processing related settings.

Arguments:

Options - The options being sent to the API.

Return Value:

If the function was successful then true is returned. If unsuccessful then false is returned and the new options did not take effect.

See Also:

GetProcessingOptions()

SetToolbarOptions()

**bool SetToolbarOptions(
ToolbarOptions Options);**

Description:

This Function sets the toolbar related settings.

Arguments:

Options - The options being sent to the API.

Return Value:

If the function was successful then true is returned. If unsuccessful then false is returned and the new options did not take effect.

See Also:

GetToolbarOptions()

2.6 Camera Functions

Description:

These functions get camera related information or set camera GPIO values.

GetSerialNumber()

```
bool GetSerialNumber(  
    long & SerialNumber);
```

Description:

This function retrieves the serial number from the camera connected to the system. It is also useful for determining if a camera is properly installed on the system. Quick Glance does not need to be running for this function to work successfully.

Arguments:

SerialNumber - This value will receive the serial number of the camera.

Return Value:

The success of the function. If false was returned then it is likely that there is no camera connected to the system or that the drivers have not been installed correctly.

SetCustomGPIO()

```
bool SetCustomGPIO (  
    CameraCustomGPIOOutputValue GPIO_1,  
    CameraCustomGPIOOutputValue GPIO_2,  
    CameraCustomGPIOOutputValue GPIO_3);
```

Description:

This function sets the custom output values of the GPIO pins on the camera. The GPIO outputs only use these values if the corresponding GPIO in the **CameraOptions** is set to **CAM_GPIO_OUT_CUSTOM**. These outputs will take effect the next time an image is acquired from the camera.

Arguments:

GPIO_1 - The custom output value for **GPIO_1**. If the value is **CAM_GPIO_OUT_VALUE_NO_CHANGE** then the output will not be affected.

GPIO_2 - The custom output value for **GPIO_2**. If the value is **CAM_GPIO_OUT_VALUE_NO_CHANGE** then the output will not be affected.

GPIO_3 - The custom output value for **GPIO_3**. If the value is **CAM_GPIO_OUT_VALUE_NO_CHANGE** then the output will not be affected.

Possible Values:

CAM_GPIO_OUT_VALUE_NO_CHANGE
CAM_GPIO_OUT_VALUE_HIGH
CAM_GPIO_OUT_RIGHT_TRACKING_STATUS

Return Value:

The success of the function

See Also:

SetCameraOptions ()
GetCameraOptions ()

A

APPLYCALIBRATIONEx() · 21, 27

B

BLINK_BOTH EYES REQUIRED · 1
 BLINK_CANCEL TIME · 1, 2
 BLINK_ENABLE SECONDARY CLICK · 1, 2
 BLINK_PRIMARY TIME · 1, 2
 BLINK_SECONDARY TIME · 1, 2
 BLINK_VISUAL FEEDBACK · 3

C

CALIBRATEEx() · 21, 22, 24, 25, 26, 27
 CALIBRATION FUNCTIONS · 21
 CALIBRATION OPTIONS · 5
 CALIBRATION_STYLE · 5
 CALIBRATION_TARGET TIME · 5
 CALIBRATIONBIASEx() · 23
 CAMERA FUNCTIONS · 34
 CAMERA OPTIONS · 10
 CAMERA_BUS BANDWIDTH PERCENTAGE · 10
 CAMERA_GAIN METHOD · 10
 CAMERA_GAIN VALUE · 10
 CAMERA_GPIO_1 · 11
 CAMERA_GPIO_2 · 11
 CAMERA_GPIO_1; CAMERA_GPIO_2 · 11
 CAMERA_GPIO_3 · 11
 CAMERA_GPIO_3 · 11
 CAMERA_IMAGE ROI PERCENTAGE · 11
 CLICK_AUDIBLE FEEDBACK · 3
 CLICK_DELAY · 3
 CLICK_METHOD · 3
 CLICK_ZOOM FACTOR · 4
 CLICKING OPTIONS · 1, 30, 32

D

DATA CAPTURE FUNCTIONS · 17
 DWELL_BOX SIZE · 4
 DWELL_TIME · 4

E

EXITQUICKGLANCE() · 13, 14

G

GETCALIBRATIONOPTIONS() · 5, 6, 29, 31
 GETCAMERAOPTIONS() · 29, 32
 GETCLICKINGOPTIONS() · 1, 30, 32
 GETPROCESSINGOPTIONS() · 30, 33
 GETQGONFLAG() · 13, 14
 GETSCOREEx() · 25, 27
 GETSDKVERSION() · 13
 GETSERIALNUMBER() · 34
 GETTOOLBAROPTIONS() · 12, 31, 33
 GETWORSTTARGETPOSITIONEx() · 22, 26, 27

H

HIDELARGEIMAGE() · 14, 16

I

INFORMATION FUNCTIONS · 13
 INITIALIZECALIBRATIONEx() · 21, 22, 23, 24, 25, 26, 27, 28
 INTERNALCALIBRATION1() · 27

M

MOVELEFTRIGHT() · 14, 15, 16
 MOVEUPDOWN() · 14, 15, 16

O

OPENCALIBRATIONEx() · 27, 28
 OPTIONS ACCESS FUNCTIONS · 29

P

PROCESSING OPTIONS · 6
 PROCESSING_ENABLE CAPTURE · 7
 PROCESSING_ENABLE CLICKING · 7
 PROCESSING_ENABLE CURSOR MOVEMENT · 7
 PROCESSING_ENABLE DISPLAY · 7
 PROCESSING_ENABLE PROCESSING · 8
 PROCESSING_EYES TO PROCESS · 6, 27
 PROCESSING_MAX PROCESS TIME · 8
 PROCESSING_PROCESS PRIORITY · 8
 PROCESSING_SMOOTHING FACTOR · 9

Q

QUERYBULKCAPTURE() · 17, 18, 19
QUICK GLANCE UI MANIPULATION FUNCTIONS · 14

R

READBULKCAPTURE() · 17, 18, 19
REGISTERCLICKEVENT() · 18
RESETENABLEEYECONTROL() · 7, 15

S

SETCALIBRATIONOPTIONS() · 6, 29, 31
SETCAMERAOPTIONS() · 10, 29, 32
SETCLICKINGOPTIONS() · 1, 30, 32

SETCUSTOMGPIO() · 11, 35
SETENABLEEYECONTROL() · 7, 15
SETMWSTATE() · 14, 15, 16
SETPROCESSINGOPTIONS() · 30, 33
SETTOOLBAROPTIONS() · 12, 31, 33
SHOWLARGEIMAGE() · 14, 16
STARTBULKCAPTURE() · 17, 18, 19, 20
STARTRAWCAPTURE() · 19, 20
STOPBULKCAPTURE() · 17, 18, 19
STOPRAWCAPTURE() · 20

T

TOGGLELARGEIMAGE() · 14, 16
TOOLBAR OPTIONS · 12
TOOLBAR_BUTTONSIZEX · 12
TOOLBAR_BUTTONSIZEY · 12
TOOLBAR_IMAGEDISPLAYTYPE · 12