



## **EyeMobile Windows Mobile BETA Documentation**

Document Version 1.0.0

© 2007 GestureTek Inc.  
317 Adelaide Street West, Toronto, Ontario, M5V 1P9 Canada  
Web: <http://www.gesturetek.com> Email: [support@gesturetek.com](mailto:support@gesturetek.com)  
Phone: 416.340.9290 Fax: 416.348.9809

# Table of Contents

<b>1</b>	<b>Common Types</b>	<b>1</b>
1.1	EColorFormat.....	1
1.2	ECameraStatus.....	1
1.3	EReturnCode.....	2
1.4	EyeMoFrameListener.....	3
1.4.1	ReceiveFrame .....	3
1.5	EyeMoTracker.....	3
1.5.1	Initialize .....	3
1.5.2	Release .....	4
1.5.3	Start .....	4
1.5.4	Stop .....	5
1.5.5	Reset .....	5
1.5.6	GetTrackerResult .....	6
1.5.7	GetDimensions .....	6
1.5.8	SetDimensions .....	7
1.5.9	ProcessFrame .....	7
<b>2</b>	<b>EyeMoCamera</b>	<b>9</b>
2.1	TVideoFormat.....	9
2.1.1	ECaptureFormat .....	9
2.2	Methods.....	10
2.2.1	Initialize .....	10
2.2.2	Release .....	10
2.2.3	Start .....	11
2.2.4	Stop .....	11
2.2.5	Pause .....	12
2.2.6	Resume .....	12
2.2.7	GetCaptureFormatCount .....	13
2.2.8	GetCaptureFormat .....	13
2.2.9	SetCaptureFormat .....	14
2.2.10	SetCaptureBufferCount .....	14
2.2.11	GetCaptureBufferCount .....	15
2.2.12	SetCaptureBufferFormat .....	15
2.2.13	GetCaptureBufferFormat .....	16
2.2.14	GetFrame .....	16
2.2.15	GetCameraWidth .....	17
2.2.16	GetCameraHeight .....	17
2.2.17	GetCameraBufferSize .....	18
2.2.18	AddFrameListener .....	18
2.2.19	RemoveFrameListener .....	19
<b>3</b>	<b>EyeMoShake</b>	<b>20</b>

---

3.1 Methods..... 20  
    3.1.1 GetResult .....20

**4 EyeMoRocknRoll 21**

4.1 Roll Motion..... 21  
4.2 Rock Motion..... 22  
4.3 Types ..... 24  
    4.3.1 ETrackingMode .....24  
    4.3.2 ETrackerResult .....25  
    4.3.3 EGesture .....25  
4.4 Methods..... 26  
    4.4.1 SetTargetWidth .....26  
    4.4.2 GetTargetWidth .....26  
    4.4.3 SetTrackerMode .....27  
    4.4.4 GetTrackerMode .....27  
    4.4.5 GetImmMotion .....28  
    4.4.6 GetAccMotion .....28  
    4.4.7 GetGesture .....29  
    4.4.8 IsGestureMotion .....29

## 1 Common Types

These types are used throughout the EyeMobile Engine and are defined in a common header file for convenience.

### 1.1 EColorFormat

This enumeration represents the possible image formats returned from the camera. The application rarely needs to interact with this type, since most cameras supply only FMT\_YV12.

FMT_UNKNOWN	Unknown format.
FMT_RGB565	16-bit RGB format with 5 bits for red, 6 for green and 5 for blue.
FMT_RGB555	16-bit RGB format with 5 bits each for red, green, and blue with one bit1 unused.
FMT_YVYU	16-bit YUV format with a byte-ordering (Y0, V0, Y2, U0).
FMT_YUY2	16-bit YUV format with a byte-ordering of (Y0, U0, Y1, V0).
FMT_UYVY	16-bit YUV format with a byte-ordering of (U0, Y0, V0, Y1).
FMT_YVU9	Planar YUV format. 8-bit Y plane followed by 4x4 sub-sampled V and U planes.
FMT_Y411	12-bit YUV format with a byte ordering of (U2, Y0, Y1, V2, Y2 Y3).
FMT_Y41P	12-bit YUV format with a byte ordering of (U0, Y0, V0, Y1, U4, Y2, V4, Y3, Y4, Y5, Y6, Y7).
FMT_Y211	8-bit YUV format with a byte ordering of (Y0, U0, Y2, V0).
FMT_YV12	Planar YUV format. 8-bit Y plane, followed by 2x2 sub-sampled V and U planes.
FMT_RGB8	24-bit RGB format with 8-bits each for red, green and blue.

### 1.2 ECameraStatus

This enumeration represents the status of the camera. It is used to pass state information from EyeMoCamera to its frame-listeners. The application only needs to make use of this type when implementing a custom frame-listener.

CAM_START	The camera has been started.
-----------	------------------------------

CAM_STOP	The camera has been stopped.
CAM_PAUSE	The camera has been paused temporarily.
CAM_RESUME	The camera has been resumed.
CAM_FRAME	The camera is passing a frame to the listener.

### 1.3 EReturnCode

This enumeration represents the possible status codes returned by methods in the EyeMobile Engine. Most API functions in the EyeMobile Engine return an error code, allowing the application to recover when an error condition occurs.

FAIL	A general error has occurred. This may indicate that the requested operation is not possible to complete because of initialization.
SUCCESS	The operation has succeeded.
NO_CAMERA	The operation could not be completed because there is no camera in the device.
OUT_OF_MEMORY	The operation could not be completed because there is insufficient memory available.
NOT_INITIALIZED	The operation could not be completed because the class has not been initialized.
ALREADY_CAPTURING	The operation could not be completed because the camera has already been started.
ALREADY_STARTED	The operation could not be completed because the tracker has already been started.
NOT_STARTED	The operation could not be completed because the tracker has not been started.
NOT_CAPTURING	The operation could not be completed because the camera has not been started.
INVALID_PARAMETER	The operation could not be completed because one of the parameters is invalid.
NOT_SUPPORTED	The operation is not supported.
ALREADY_INITIALIZED	The operation could not be completed because the class

	has already been initialized.
--	-------------------------------

## 1.4 EyeMoFrameListener

This listener interface is used to receive frame and state information callbacks from the camera. Each of the trackers in the EyeMobile engine implements this interface, allowing for easy integration with EyeMoCamera.

Applications may implement this interface if they need to take special action whenever a new frame arrives from the camera (eg. a frame-rate counter). However, it is not normally necessary for the application to implement this interface.

### 1.4.1 *ReceiveFrame*

---

This callback function is invoked each time the camera status changes or a new frame arrives.

#### Definition

```
virtual void ReceiveFrame(  
    void * pData,  
    long nElapsedTime,  
    ECameraStatus eStatus);
```

#### Parameters

`pData`

A pointer to a buffer containing the raw frame data.

`nElapsedTime`

The number of milliseconds that elapsed between the previous and current camera frames.

`eStatus`

The current camera status.

#### Returns

None

#### Notes

None

## 1.5 EyeMoTracker

This interface defines the life-cycle of a tracker in the EyeMobile Engine. All trackers in the EyeMobile Engine may be used independent of the EyeMoCamera class, and these are the control operations that are required.

### 1.5.1 *Initialize*

---

Initialize the tracker into a known state, and prepare to be started. It is not possible to interact with the tracker unless it has been successfully initialized.

Once the tracker has been initialized, the various parameter values may be set using the subclass-specific setter methods.

**Definition**

```
virtual EReturnCode Initialize();
```

**Parameters**

None

**Returns**

ALREADY\_INITIALIZED

If the tracker has already been initialized.

SUCCESS

If the operation completed successfully.

**Notes**

None

## 1.5.2 Release

---

Shutdown the tracker and free any resources that may have been allocated. It is not possible to interact with the tracker until it has been properly initialized again.

**Definition**

```
virtual EReturnCode Release();
```

**Parameters**

None

**Returns**

NOT\_INITIALIZED

If the tracker has not yet been initialized.

SUCCESS

If the operation completed successfully.

**Notes**

None

## 1.5.3 Start

---

Transition the tracker into a running state. When the tracker is running, it is no longer possible to modify many of the tracker parameters.

Because changes in the tracker parameters may require the tracker to change the amount of allocated memory, the tracker may allocate memory during startup.

**Definition**

```
virtual EReturnCode Start();
```

**Parameters**

None

**Returns**

NOT\_INITIALIZED

If the tracker has not yet been initialized.

ALREADY\_STARTED

If the tracker has already been started.

OUT\_OF\_MEMORY

If an error occurred while allocating the necessary memory.

SUCCESS

If the operation completed successfully

**Notes**

None

### 1.5.4 Stop

---

Transition the tracker into a stopped state. When the tracker is stopped, it is possible to modify many of the tracker parameters.

Any memory that was allocated during start may be deallocated when the camera is stopped.

**Definition**

```
virtual EReturnCode Stop();
```

**Parameters**

None

**Returns**

NOT\_STARTED

If the tracker has not yet been started.

SUCCESS

If the operation completed successfully.

**Notes**

None

### 1.5.5 Reset

---

Clear the internal state of the tracker. All calculated values are reset to their defaults, and the buffer holding the previous frame is cleared.

The tracker dimensions are not affected.

**Definition**

```
EReturnCode Reset();
```

**Parameters**

None

**Returns**

NOT\_INITIALIZED

If the tracker has not yet been initialized.

NOT\_STARTED

If the tracker has not yet been started.

SUCCESS

If the operation completed successfully.

**Notes**

None

## 1.5.6 *GetTrackerResult*

---

This function returns the tracker status for the most recent frame processed.

**Definition**

```
EReturnCode GetTrackerResult(  
    ETrackerResult& reResult);
```

**Parameters**

reResult

A reference to where the tracker status will be stored.

**Returns**

NOT\_STARTED

If the tracker has not yet been started.

SUCCESS

If the operation completed successfully.

**Notes**

None

## 1.5.7 *GetDimensions*

---

Returns the expected size of the camera frames to be processed.

**Definition**

```
EReturnCode GetDimensions(  
    int& rnWidth,  
    int& rnHeight);
```

**Parameters**

nWidth

A reference to where the width value will be stored.

nHeight

A reference to where the height value will be stored.

**Returns**

NOT\_INITIALIZED

If the tracker has not yet been initialized.

SUCCESS

If the operation completed successfully.

**Notes**

None

### 1.5.8 *SetDimensions*

---

Sets the expected size of the camera frames to be processed.

The dimensions can only be set when the tracker is not running.

**Definition**

```
EReturnCode SetDimensions(  
    int nWidth,  
    int nHeight);
```

**Parameters**

nWidth

The expected width of the camera frames.

nHeight

The expected height of the camera frames.

**Returns**

NOT\_INITIALIZED

If the tracker has not yet been initialized.

ALREADY\_STARTED

If the tracker has already been started.

INVALID\_PARAMETER

If either the width or the height is not in the valid range.

SUCCESS

If the operation completed successfully.

**Notes**

None

### 1.5.9 *ProcessFrame*

---

Pass a frame to the tracker for processing. This function will update any data values that are computed by the tracker.

**Definition**

```
virtual EReturnCode ProcessFrame(  
    void * pDataBuffer,  
    long nElapsedTime);
```

**Parameters**

pDataBuffer

A pointer to a data buffer containing the current camera frame in raw format.

nElapsedTime

The number of milliseconds that elapsed between the previous and current camera frames.

**Returns**

NOT\_STARTED

If the tracker has not yet been started.

SUCCESS

If the operation completed successfully.

**Notes**

The tracking algorithm may have detected a problem, even though this function returns SUCCESS. The application should check the tracker status for the current frame in order to determine if the tracker is returning a warning or error, and treat the computed tracker values accordingly.

## 2 EyeMoCamera

This class is provided to simplify the development process for camera-enabled applications using one or more trackers in the EyeMobile Engine on Windows Mobile devices. EyeMoCamera takes care of all of the DirectShow interaction necessary in order to access the camera hardware in the device. EyeMoCamera exposes a subset of that camera functionality suitable for driving real-time image processing applications.

### 2.1 TVideoFormat

This structure holds information about a specific capture mode that the device camera can be in. This gives the application some control over the camera behavior.

In order to select a particular capture mode, the application should iterate through the list of supported capture formats using EyeMoCamera::GetCaptureFormatCount, and EyeMoCamera::GetCaptureFormat. When the desired capture format has been located, the application may select the format using EyeMoCamera::SetCaptureFormat.

If no such format is selected, the default behavior is to select the smallest YV21 format that captures at 15fps.

DWORD dwWidth;	The width of the captured frames.
DWORD dwHeight;	The height of the captured frames.
DWORD dwPitch;	The pitch of the captured frames.
float fFPS;	The approximate capture frame rate. On many handsets, the capture frame rate changes based on the ambient light level.
EColorFormat eFormat;	The color format of the captured frames.
DWORD dwFrameSize;	The size of the raw camera frame buffer.

#### 2.1.1 ECaptureFormat

This enumeration represents the possible camera frame formats returned by EyeMoCamera::GetFrame. The camera frame can be returned in a raw buffer, or as a DirectDraw surface. The capture format may be selected using EyeMoCamera::SetCaptureBufferFormat.

If an application wishes to capture frames as DirectDraw surfaces, it must pass a pointer to DirectDraw during initialization. Otherwise, the attempt to set the capture buffer format will fail.

CAPTURE_RAW_YV12	Raw buffer in YV12 format.
------------------	----------------------------

**Note:** More formats will be added in the future.

## 2.2 Methods

### 2.2.1 Initialize

---

Initialize the camera into a known state, and prepare to be started. It is not possible to interact with the camera unless it has been successfully initialized.

Once the camera has been initialized, the various parameter values may be set using the specific setter methods.

#### Definition

```
EReturnCode Initialize(  
    void* pDeviceConfig = NULL);
```

#### Parameters

pDeviceConfig

A pointer to device-specific configuration information. Pass NULL for the default behavior. Also, this parameter can be used to pass a pointer to DirectDraw, to enable the camera to return frames as DirectDraw surfaces.

#### Returns

ALREADY\_INITIALIZED

If the camera has already been initialized.

NO\_CAMERA

If the handset does not contain a camera.

OUT\_OF\_MEMORY

If an error occurred while allocating the necessary memory.

FAIL

If a low-level (DirectDraw/DirectShow) error occurred.

SUCCESS

If the operation completed successfully.

#### Notes

None

### 2.2.2 Release

---

Shutdown the camera and free any resources that may have been allocated. It is not possible to interact with the camera until it has been properly initialized again.

#### Definition

```
EReturnCode Release();
```

#### Parameters

None

#### Returns

NOT\_INITIALIZED

The camera has not yet been initialized.

SUCCESS

If the operation completed successfully.

**Notes**

If the camera is currently running, it will automatically be stopped.

### 2.2.3 Start

---

Begin capturing frames from the camera. As new frames arrive, they will be passed to each of the frame listeners that have been registered.

Many of the camera control parameters cannot be changed while the camera is running. Once start has been called, it is not possible to modify these values until the camera is stopped.

**Definition**

```
EReturnCode Start();
```

**Parameters**

None

**Returns**

NOT\_INITIALIZED

If the camera has not yet been initialized.

ALREADY\_CAPTURING

If the camera has already been started.

SUCCESS

If the operation completed successfully.

**Notes**

There is a significant amount of overhead required to actually start the camera. Therefore, this function should not be called unless the camera will be running for a while. Otherwise, it will be more efficient to start the camera once and use pause/resume.

### 2.2.4 Stop

---

Finish capturing frames from the camera. No new frames will be passed to the frame listeners after this function is called.

Many of the camera control parameters cannot be changed while the camera is running. Once stop has been called, it is possible to modify these values until the camera is started again.

**Definition**

```
EReturnCode Stop();
```

**Parameters**

None

**Returns**

NOT\_INITIALIZED

If the camera has not yet been initialized.

NOT\_CAPTURING

If the camera is not currently running.

SUCCESS

If the operation has completed successfully.

#### Notes

See notes for EyeMoCamera::Start.

## 2.2.5 Pause

---

Temporarily stop the flow of camera frames to the frame-listeners.

#### Definition

```
EReturnCode Pause();
```

#### Parameters

None

#### Returns

NOT\_INITIALIZED

If the camera has not yet been initialized.

NOT\_CAPTURING

If the camera has not yet been started or has already been paused.

FAIL

If a DirectShow error occurred.

SUCCESS

If the operation completed successfully.

#### Notes

When the camera is paused, it is kept in a state such that it can return to successful operation almost immediately when resume is called. This makes pause/resume much more efficient than stop/start when the camera will be disabled for only a short period of time.

## 2.2.6 Resume

---

Return the camera to normal operation after it has been paused. After this method returns, any new frames that arrive will be sent to the registered frame listeners.

#### Definition

```
EReturnCode Resume();
```

#### Parameters

None.

#### Returns

NOT\_INITIALIZED

If the camera has not yet been initialized.

ALREADY\_CAPTURING

If the camera has not been paused, or has already been resumed.

FAIL

If a DirectShow error occurred.

SUCCESS

If the operation completed successfully.

**Notes**

See the notes for EyeMoCamera::Pause.

## 2.2.7 *GetCaptureFormatCount*

---

Return the number of distinct capture modes that are supported by the device. The capture format may be set using any non-negative index smaller than this number.

**Definition**

```
DWORD GetCaptureFormatCount();
```

**Parameters**

None

**Returns**

The number of distinct capture modes that are supported by the device.

**Notes**

None

## 2.2.8 *GetCaptureFormat*

---

Return the specific capture format identified by the given index. Any non-negative index that is smaller than the value returned by EyeMoCamera::GetCaptureFormatCount may be used.

**Definition**

```
EReturnCode GetCaptureFormat(  
    int nIndex,  
    TVideoFormat& rVideoFormat);
```

**Parameters**

nIndex

The index of the capture format that is to be returned.

rVideoFormat

A reference to an existing TVideoFormat structure that is to be filled with the capture mode information.

**Returns**

INVALID\_PARAMETER

If the given index is out of bounds.

SUCCESS

If the operation completed successfully.

**Notes**

None

## 2.2.9 *SetCaptureFormat*

---

Selects a particular capture mode from the list of valid formats. And non-negative index smaller than the value returned by EyeMoCamera::GetCaptureFormatCount may be used.

By default, the index 0 is used. This value corresponds to the smallest supported camera frame in YV12 format that supports capture at 15 fps.

**Definition**

```
EReturnCode SetCaptureFormat(  
    int nIndex);
```

**Parameters**

nIndex

The index of the capture format that is to be selected.

**Returns**

NOT\_INITIALIZED

If the camera has not yet been initialized.

ALREADY\_CAPTURING

If the camera has already been started.

INVALID\_PARAMETER

If the given index is out of bounds.

FAIL

If a DirectShow error occurred.

SUCCESS

If the operation completed successfully.

**Notes**

None

## 2.2.10 *SetCaptureBufferCount*

---

Select the number of internal buffers used to store previous copies of the camera image. If this value is set to 0, then the class does not store the most recent camera image and any calls to EyeMoCamera::GetFrame will return NOT\_SUPPORTED.

By default, this parameter is 0. If the camera image is being stored internally, it is recommended that the number of capture buffers be set to a value of at least 3.

**Definition**

```
EReturnCode SetCaptureBufferCount(  
    int nCount);
```

**Parameters**

nCount

The number of internal capture buffers to maintain.

**Returns**

NOT\_INITIALIZED

If the camera has already been initialized.

ALREADY\_CAPTURING

If the camera has already been started.

OUT\_OF\_MEMORY

If an error occurred while allocating memory.

SUCCESS

If the operation completed successfully.

**Notes**

If your application is not making use of the actual camera image (ie. displaying the camera image on-screen) then the capture buffer count should be set to 0 for performance reasons. In that case, the camera is able to eliminate an image copy for each new frame.

## 2.2.11 **GetCaptureBufferCount**

---

Returns the number of internal capture buffers.

**Definition**

```
EReturnCode GetCaptureBufferCount(  
    int& rnCount);
```

**Parameters**

nCount

A reference to an integer value that will store the requested value.

**Returns**

NOT\_INITIALIZED

If the camera has not yet been initialized.

SUCCESS

If the operation has completed successfully.

**Notes**

None

## 2.2.12 **SetCaptureBufferFormat**

---

Select the format of the frames that are stored in the capture buffer. The images returned by EyeMoCamera::GetFrame are all returned in this format.

**Definition**

```
EReturnCode SetCaptureBufferFormat(  
    ECaptureFormat eFormat);
```

**Parameters**

eFormat

The format that should be used to store the captured camera frames.

**Returns**

NOT\_INITIALIZED

If the camera has not yet been initialized.

ALREADY\_CAPTURING

If the camera has already been started.

SUCCESS

If the operation has completed successfully.

**Notes**

It is only possible to select one of the formats corresponding to a DirectDraw surface when a pointer to DirectDraw is passed to the camera during initialization.

### 2.2.13 *GetCaptureBufferFormat*

---

This function returns the format of the frames that are stored in the capture buffer.

**Definition**

```
EReturnCode GetCaptureBufferFormat(  
    ECaptureFormat& reFormat);
```

**Parameters**

eFormat

A reference to a format enum that will store the requested value.

**Returns**

NOT\_INITIALIZED

If the camera has not yet been initialized.

SUCCESS

If the operation completed successfully.

**Notes**

None

### 2.2.14 *GetFrame*

---

This function returns a pointer to the most recent camera frame. The format of the camera frame is set by the call EyeMoCamera::SetCameraBufferFormat. When a raw format is specified, then the resulting buffer should be cast to a (byte \*). When a DirectDraw surface format is selected, then the resulting buffer should be cast to an (IDirectDrawSurface \*).

If EyeMoCamera::SetCaptureBufferCount was set to 0, this function will always return FAILED.

**Definition**

```
EReturnCode GetFrame(  
    void** pvData);
```

**Parameters**

pvData

Pointer to a data buffer containing the most recent frame. This buffer should be typecast to a buffer of an appropriate type.

**Returns**

NOT\_INITIALIZED

If the camera has not yet been initialized.

NOT\_CAPTURING

If the camera has not yet been started, or the camera is currently paused.

NOT\_SUPPORTED

If the number of capture buffers is 0.

SUCCESS

If the operation completed successfully.

**Notes**

None

### 2.2.15 *GetCameraWidth*

---

This function returns the width of the camera frames that are being captured.

If the camera has not yet been initialized, this function returns 0.

**Definition**

```
DWORD GetCameraWidth();
```

**Parameters**

None

**Returns**

The width of the camera image

**Notes**

None

### 2.2.16 *GetCameraHeight*

---

This function returns the height of the camera frames that are being captured.

If the camera has not yet been initialized, this function returns 0.

**Definition**

```
DWORD GetCameraHeight();
```

**Parameters**

None

**Returns**

The height of the camera image

**Notes**

None

## 2.2.17 GetCameraBufferSize

---

This function returns the size in bytes of the buffer returned by EyeMoCamera::GetFrame.

If the camera has not yet been initialized, this function returns 0.

### Definition

```
DWORD GetCameraBufferSize();
```

### Parameters

None

### Returns

The size in bytes, of the buffer returned by EyeMoCamera::GetFrame.

### Notes

None

## 2.2.18 AddFrameListener

---

Adds a listener to the list of all frame listeners that are notified whenever the camera changes state or whenever a new frame arrives from the camera.

Only one instance of a frame listener is allowed to be in the list of frame listeners at any given time. An attempt to add a listener more than once will result in an error.

Frame listeners may only be added when the camera is stopped.

### Definition

```
EReturnCode AddFrameListener(  
    EyeMoFrameListener* pListener);
```

### Parameters

pListener

A pointer to the frame listener to be added.

### Returns

NOT\_INITIALIZED

If the camera has not been initialized.

ALREADY\_CAPTURING

If the camera has already been started.

INVALID\_PARAMETER

If the frame listener pointer is NULL.

FAIL

If a duplicate listener is being added..

SUCCESS

If the operation completed successfully.

### Notes

When the camera sends a message the list of registered frame listeners, they are notified in the

order in which they are added. This allows any application-defined frame listeners to ensure that they are called after (or before) the tracker data has been updated.

### **2.2.19 RemoveFrameListener**

---

Removes a listener from the list of all frame listeners that are notified whenever the camera changes state or whenever a new frame arrives from the camera.

Frame listeners may only be removed when the camera is stopped.

#### **Definition**

```
EReturnCode RemoveFrameListener(  
    EyeMoFrameListener* pListener);
```

#### **Parameters**

pListener

A pointer to the frame listener to be removed.

#### **Returns**

NOT\_INITIALIZED

If the camera has not yet been initialized.

ALREADY\_CAPTURING

If the camera has already been started.

FAIL

If the given listener is not in the list.

SUCCESS

If the operation completed successfully.

#### **Notes**

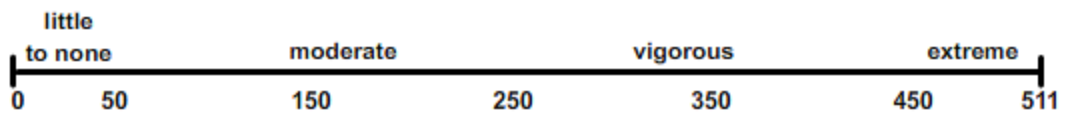
None

## 3 EyeMoShake

This class implements a 1-d accelerometer for measuring raw handset motion between consecutive camera frames. This tracker is suitable for applications where the intensity, but not the direction of motion is measured. For instance, EyeMoShake could be used to implement a dice-rolling, a simple power gauge that increases as the user shakes the phone, or a snow-globe.

The motion measurement is corrected for the scene complexity and lighting, so the raw motion value is approximately independent of the scene that the camera is viewing and the resolution of the images that are processed.

The motion measurement has a range of 0 to 511, inclusive. The magnitude correlates to the type of motion to which the camera is subjected as illustrated here:



The motion measurement may be used by the application to differentiate between gentle and vigorous shaking motions. Alternatively, the application may apply a threshold to the motion measurement, if the application requires only motion state or still state information.

EyeMoShake inherits from both EyeMoTracker and EyeMoFrameListener.

### 3.1 Methods

#### 3.1.1 *GetResult*

---

This function returns the raw motion that was detected in the scene between the previous two frames.

##### Definition

```
EReturnCode GetResult(  
    int& rnResult);
```

##### Parameters

`rnResult`

A reference to where the motion result will be stored.

##### Returns

`NOT_STARTED`

If the tracker has not been started.

`SUCCESS`

If the operation completed successfully.

##### Notes

The raw motion will always be in the range [0,511].

Even if the handset is held perfectly still, it is still possible that the raw motion will not return 0 due to the small amount of noise in the camera image.

## 4 EyeMoRocknRoll

EyeMoRocknRoll transforms a camera-enabled mobile device into a controller. Games and applications can be controlled by simply moving the device. There is no need to use the handset keypad.

The EyeMoRocknRoll extension provides an easy-to-use BREW interface that makes camera-based motion tracking available to BREW applications. It supports two types of tracking: Rock, where the handset is flicked rapidly in one direction and immediately returned to its original orientation; and Roll, where the tracker attempts to determine changes in handset orientation. Rock and Roll are described in more detail below.

EyeMoRocknRoll can be used in applications where analog input is required, such as rolling a ball through a maze, positioning a map, or smoothly scrolling a menu.

While EyeMoRocknRoll provides a look and feel similar to a joystick, it is essential to understand that EyeMoRocknRoll does not work like a joystick, which uses absolute position, but more like a mouse, using relative motion.

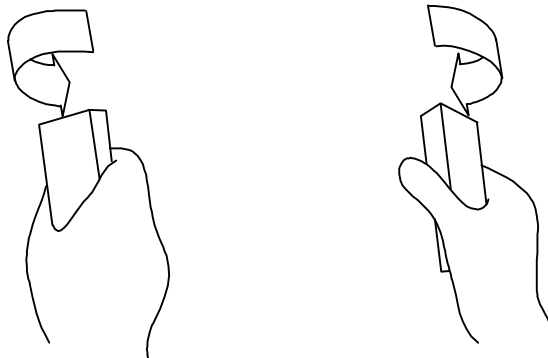
EyeMoRocknRoll inherits from both EyeMoTracker and EyeMoFrameListener.

### 4.1 Roll Motion

The EyeMoRocknRoll extension can track the Roll motion of the device. This is designed for gentle, precise, analog control, and could be used to for such things as the rolling of a virtual ball, thrusters on a spaceship, etc.

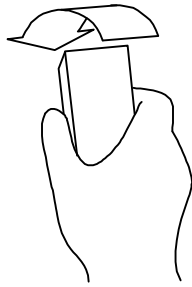
Roll motion supports movement independently in the X and Y axes as illustrated here:

#### X Axis

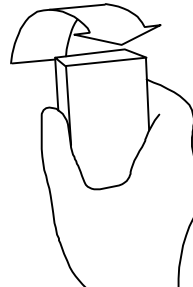


Turning the handset to the left. Turning the handset to the right.

### Y Axis



Tilting the top of the device away from you.



Tilting the top of the device toward you.

### How Roll works

As the device is tilted the tracker estimates the distance that the device has moved between consecutive video frames. This is referred to as immediate motion. Motion is measured in pixel units with respect to the camera image. It is returned as a fixed-point value with an 8-bit decimal, since many BREW phones do not have native hardware support for floating-point. The tracker is designed to provide smooth motion via sub-pixel estimation, which helps to reduce jitter between discrete pixel values.

Accumulating immediate motion can be used to determine the total motion over a longer span of time. However it is essential to note that accumulated motion is not the same as absolute position. The tracker does not use fixed starting points like a joystick. Instead it calculates motion on the basis of frame-by-frame comparison. Therefore using accumulated motion in the attempt to calculate current position is not recommended.

The speed in which the user may tilt or turn the device is limited by the frame-rate and field-of-view of the camera, since consecutive video frames must contain at least some overlapping portion of the image. A warning will be returned if the user moves the device too fast.

For best results, the video frames should contain a static scene containing distinctive visual elements. Output values are not reliable when the camera is being used in low-light situations, high brightness situations (camera pointed to a bright-beam of light), and lack of feature-rich environment (camera pointed to a blank surface). The tracker can track scenes with little detail or texture, however the accuracy of the results may be reduced. A warning will be returned when the detail is below a specified threshold.

## 4.2 Rock Motion

A Rock gesture is similar to a roll gesture, except that the device is returned to its initial position.

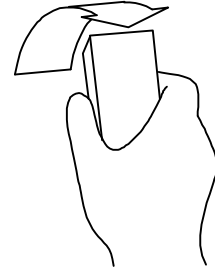
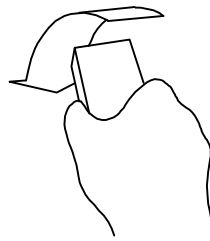
The inclusion of a return motion helps assure that only purposeful gestures are detected, while random and arbitrary motions of the device are ignored.

The gesture can be performed as one fluid motion, or with a slight pause between the two motions. The system will report the gesture as soon as the return motion is detected, and usually before the motion has concluded. This allows the application to respond while the device is still in motion, for a sense of immediacy.

Below is an example of how to perform each gesture:

**FLICK\_UP**

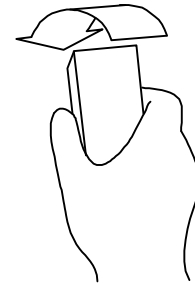
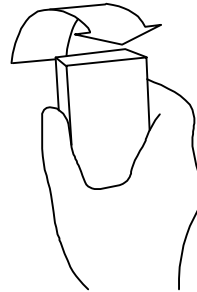
**Note:** The following illustration is correct. At first glance it may seem to be the reverse of what you might expect.



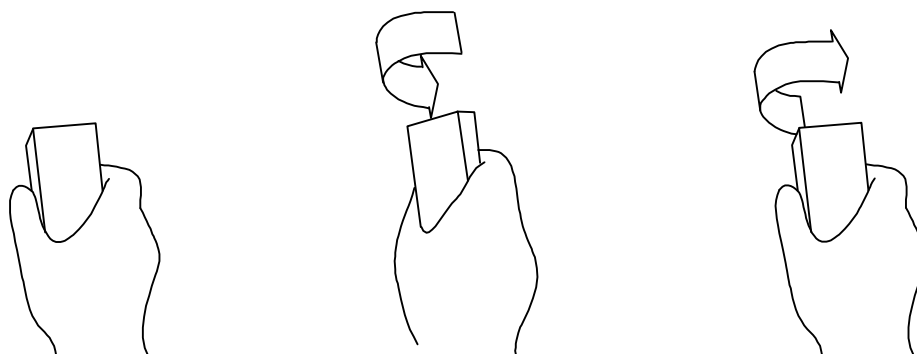
1. Start at the neutral position.
2. Tilt the device away from yourself by bending your wrist.
3. Tilt the device back to the neutral position.

**FLICK\_DOWN**

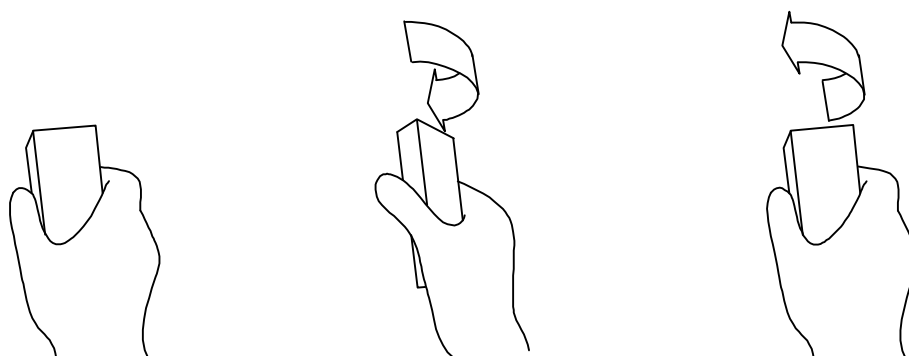
**Note:** The following illustration is correct. At first glance it may seem to be the reverse of what you might expect.



1. Start at the neutral position.
2. Tilt the device towards yourself by bending your wrist.
3. Tilt the device back to the neutral position.

**FLICK\_LEFT**

1. Start at the neutral position.
2. Tilt the device to the left by rotating your wrist.
3. Tilt the device back to the neutral position.

**FLICK\_RIGHT**

1. Start at the neutral position.
2. Tilt the device to the right by rotating your wrist.
3. Tilt the device back to the neutral position.

## 4.3 Types

### 4.3.1 *ETrackingMode*

This enumeration selects which tracking modes for the tracker to run. If a particular tracking mode is not required, it can be disabled for a minor performance improvement.

MODE_ROLL	The tracker runs Roll (tilt sensor) processing only.
MODE_ROCK	The tracker runs Rock (gesture recognition) processing only.
MODE_BOTH	The tracker runs both Roll and Rock processing.

### 4.3.2 *ETrackerResult*

---

This enumeration encodes the various status messages that can be returned by the tracker.

TRACKER_OKAY	The tracker processing was successful for the last frame processed.
INSUFFICIENT_DATA	There was insufficient data to properly process the current frame. This typically occurs on the first frame after startup or being reset.
WARNING_LOW_DETAIL	There was insufficient detail in the current frame for the tracker to ensure accurate tracking results. The tracker data for the current frame represents the best-guess that the tracker could make with the given data.
WARNING_TOO_FAST	There was an insufficient amount of overlap between consecutive frames for the tracker to calculate meaningful results. The tracker values will be 0 for the current frame.
SPAN_ON_RESUME	Not implemented.
NO_SPAN_ON_RESUME	Not implemented.
TRACKER_ERROR	An error occurred while processing the current tracker frame. The tracker values will be 0 for the current frame.

### 4.3.3 *EGesture*

---

This enumeration represents a gesture that is returned from the tracker. When a gesture is detected, it is only returned during a single frame.

NO_GESTURE	No gesture was detected for the current frame.
FLICK_LEFT	A left-flick gesture was detected.
FLICK_RIGHT	A right-flick gesture was detected.
FLICK_UP	An up-flick gesture was detected.
FLICK_DOWN	A down-flick gesture was detected.

## 4.4 Methods

### 4.4.1 *SetTargetWidth*

---

This function sets the decimated target width that the tracker uses during processing.

If the target width is set to a value smaller than the default, the tracking algorithm will use less processing power, but the results will have a coarser granularity.

The target width can only be set when the tracker is not running.

#### Definition

```
EReturnCode SetTargetWidth(  
    int nTargetWidth);
```

#### Parameters

nTargetWidth

The target width to set.

#### Returns

NOT\_INITIALIZED

If the tracker has not yet been initialized.

ALREADY\_STARTED

If the tracker has already been started.

INVALID\_PARAMETER

If the target width is not in the valid range.

SUCCESS

If the operation completed successfully.

#### Notes

Changing the target width is not recommended unless running on a severely resource-constrained handset.

### 4.4.2 *GetTargetWidth*

---

This function returns the current target width setting.

#### Definition

```
EReturnCode GetTargetWidth(  
    int& rnTargetWidth);
```

#### Parameters

nTargetWidth

A reference to where the value will be stored.

#### Returns

NOT\_INITIALIZED

If the tracker has not yet been initialized.

SUCCESS

If the operation completed successfully.

**Notes**

None

### 4.4.3 *SetTrackerMode*

---

This function sets the tracking mode.

The tracking mode cannot be changed while the tracker is running.

The default value is to run both Rock and Roll modes simultaneously. Disabling one or the other option will result in a small performance improvement.

**Definition**

```
EReturnCode SetTrackerMode(  
    ETrackingMode eTrackingMode);
```

**Parameters**

eTrackingMode

The tracking mode to set.

**Returns**

NOT\_INITIALIZED

If the tracker has not yet been initialized.

ALREADY\_STARTED

If the tracker has already been started.

SUCCESS

If the operation completed successfully.

**Notes**

None

### 4.4.4 *GetTrackerMode*

---

This function returns the current tracking mode.

**Definition**

```
EReturnCode GetTrackerMode(  
    ETrackingMode& reTrackingMode);
```

**Parameters**

reTrackingMode

A reference to where the value will be stored.

**Returns**

NOT\_INITIALIZED

If the tracker has not yet been initialized.

SUCCESS

If the operation completed successfully.

**Notes**

None

### 4.4.5 *GetImmMotion*

---

This function returns the immediate motion calculated by the tracker.

**Definition**

```
EReturnCode GetImmMotion(  
    int& rnMotionX,  
    int& rnMotionY);
```

**Parameters**

rnMotionX

A reference to where the x-component of the immediate motion will be stored.

rnMotionY

A reference to where the y component of the immediate motion will be stored.

**Returns**

NOT\_SUPPORTED

If the current tracking mode does not support this data.

NOT\_STARTED

If the tracker has not yet been started.

SUCCESS

If the operation completed successfully.

**Notes**

For an explanation of the differences between immediate and accumulated motion, please see the section on Tracker Values above.

This function will only operate when the tracking mode Roll has been enabled.

### 4.4.6 *GetAccMotion*

---

This function returns the accumulated motion calculated by the tracker.

**Definition**

```
EReturnCode GetAccMotion(  
    int& rnMotionX,  
    int& rnMotionY);
```

**Parameters**

rnMotionX

A reference to where the x-component of the accumulated motion will be stored.

rnMotionY

A reference to where the y component of the accumulated motion will be stored.

**Returns**

NOT\_SUPPORTED

If the current tracking mode does not support this data.

NOT\_STARTED

If the tracker has not yet been started.

SUCCESS

If the operation completed successfully.

#### Notes

For an explanation of the differences between immediate and accumulated motion, please see the section on Tracker Values above.

This function will only operate when the tracking mode Roll has been enabled.

### 4.4.7 *GetGesture*

---

This function returns the most recent gesture. Gestures are returned to the application only once.

#### Definition

```
EReturnCode GetGesture(  
    EGesture& rGesture);
```

#### Parameters

rGesture

A reference to where the gesture will be stored.

#### Returns

NOT\_SUPPORTED

If the current tracking mode does not support this data.

NOT\_STARTED

If the tracker has not yet been started.

SUCCESS

If the operation completed successfully.

#### Notes

This function will only operate when the tracking mode Rock has been enabled.

### 4.4.8 *IsGestureMotion*

---

This function returns whether the tracker currently detects a possible gesture in progress.

#### Definition

```
EReturnCode IsGestureMotion(  
    BOOL& rbMotion);
```

#### Parameters

rbMotion

A reference to where the motion status will be stored.

#### Returns

NOT\_SUPPORTED

If the current tracking mode does not support this data.

NOT\_STARTED

If the tracker has not yet been started.

SUCCESS

If the operation completed successfully.

**Notes**

This function will only operate when the tracking mode Rock has been enabled.