



EyeMobile Symbian 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
Phone: 416.340.9290 Fax: 416.348.9809

Table of Contents

1	Beta Limitations	1
2	Common Types	2
2.1	TColorFormat.....	2
2.2	TCameraStatus.....	2
2.3	TReturnCode.....	3
2.4	TTrackerResult.....	4
2.5	MEyeMoFrameListener.....	4
2.5.1	ReceiveFrame	4
2.6	MEyeMoTracker.....	5
2.6.1	Start	5
2.6.2	Stop	5
2.6.3	Reset	6
2.6.4	GetTrackerResult	6
2.6.5	GetDimensions	7
2.6.6	SetDimensions	7
2.6.7	ProcessFrame	8
3	CEyeMoCamera	9
3.1	Types	9
3.1.1	TVideoFormat	9
3.2	Methods.....	9
3.2.1	NewL / NewLC	9
3.2.2	StartL	10
3.2.3	Stop	10
3.2.4	Pause	11
3.2.5	ResumeL	11
3.2.6	GetCaptureFormatCount	11
3.2.7	GetCaptureFormat	12
3.2.8	SetCaptureFormat	12
3.2.9	SetCaptureBufferCount	13
3.2.10	GetCaptureBufferCount	14
3.2.11	GetFrame	14
3.2.12	AddFrameListener	15
3.2.13	RemoveFrameListener	15
4	CEyeMoShake	17
4.1	Methods.....	17
4.1.1	GetResult	17

5	CEyeMoRocknRoll	18
5.1	Roll Motion.....	18
5.2	Rock Motion.....	19
5.3	Types	21
5.3.1	TTrackingMode	21
5.3.2	TGesture	22
5.4	Methods.....	22
5.4.1	SetTargetWidth	22
5.4.2	GetTargetWidth	23
5.4.3	SetTrackerMode	23
5.4.4	GetTrackerMode	23
5.4.5	GetImmMotion	24
5.4.6	GetAccMotion	24
5.4.7	GetGesture	25
5.4.8	IsGestureMotion	25

1 Beta Limitations

The beta limitations of the Symbian SDK has been tested on S60 version 2.1 and higher. The demos have not been fully tested on any UIQ devices, though they may still work normally.

Some devices may not support this version of the EyeMobile SDK. If you have difficulty implementing the SDK on a device, please contact GestureTek technical support.

For technical assistance with device compatibility, please send an email to adam@gesturetek.com.

Note: Camera support on the Series 60 Emulator is less than satisfactory. The emulator for the version 2 SDK uses a looping animation instead of actual camera input. The emulator for the version 3 SDK has no camera support.

2 Common Types

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

2.1 TColorFormat

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 EFmtYV12.

EFmtUnknown	Unknown format.
EFmtRGB565	16-bit RGB format with 5 bits for red, 6 for green and 5 for blue.
EFmtRGB555	16-bit RGB format with 5 bits each for red, green, and blue with one bit ¹ unused.
EFmtYVYU	16-bit YUV format with a byte-ordering (Y0, V0, Y2, U0).
EFmtYUY2	16-bit YUV format with a byte-ordering of (Y0, U0, Y1, V0).
EFmtUYVY	16-bit YUV format with a byte-ordering of (U0, Y0, V0, Y1).
EFmtYVU9	Planar YUV format. 8-bit Y plane followed by 4x4 sub-sampled V and U planes.
EFmtY411	12-bit YUV format with a byte ordering of (U2, Y0, Y1, V2, Y2, Y3).
EFmtY41P	12-bit YUV format with a byte ordering of (U0, Y0, V0, Y1, U4, Y2, V4, Y3, Y4, Y5, Y6, Y7).
EFmtY211	8-bit YUV format with a byte ordering of (Y0, U0, Y2, V0).
EFmtYV12	Planar YUV format. 8-bit Y plane, followed by 2x2 sub-sampled V and U planes.
EFmtRGB8	24-bit RGB format with 8-bits each for red, green and blue.

2.2 TCameraStatus

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

ECamStart	The camera has been started.
-----------	------------------------------

ECamStop	The camera has been stopped.
ECamPause	The camera has been paused temporarily.
ECamResume	The camera has been resumed.
ECamFrame	The camera is passing a frame to the listener.

2.3 TReturnCode

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.

EFail	A general error has occurred. This may indicate that the requested operation is not possible to complete because of initialization.
ESuccess	The operation has succeeded.
ENoCamera	The operation could not be completed because there is no camera in the device.
EOutOfMemory	The operation could not be completed because there is insufficient memory available.
EAlreadyCapturing	The operation could not be completed because the camera has already been started.
EAlreadyStarted	The operation could not be completed because the tracker has already been started.
ENotStarted	The operation could not be completed because the tracker has not been started.
ENotCapturing	The operation could not be completed because the camera has not been started.
EInvalidParameter	The operation could not be completed because one of the parameters is invalid.
ENotSupported	The operation is not supported.

2.4 TTrackerResult

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

ETrackerOkay	The tracker processing was successful for the last frame processed.
EInsufficientData	There was insufficient data to properly process the current frame. This typically occurs on the first frame after startup or being reset.
EWarningLowDetail	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.
EWarningTooFast	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.
ETrackerError	An error occurred while processing the current tracker frame. The tracker values will be 0 for the current frame.

2.5 MEyeMoFrameListener

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 CEyeMoCamera.

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.

2.5.1 ReceiveFrame

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

Definition

```
virtual void ReceiveFrame(
    void * aData,
    TInt32 aElapsedTimeMS,
    TCameraStatus aStatus);
```

Parameters

aData

A pointer to a buffer containing the raw frame data.

aElapsedTimeMS

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

aStatus

The current camera status.

Returns

None

Notes

None

2.6 MEyeMoTracker

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 CEyeMoCamera class, and these are the control operations that are required.

2.6.1 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 TReturnCode Start();
```

Parameters

None

Returns

EAlreadyStarted

If the tracker has already been started.

EOutOfMemory

If an error occurred while allocating the necessary memory.

ESuccess

If the operation completed successfully

Notes

None

2.6.2 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 TReturnCode Stop();
```

Parameters

None

Returns

ENotStarted

If the tracker has not yet been started.

ESuccess

If the operation completed successfully.

Notes

None

2.6.3 *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

```
virtual TReturnCode Reset();
```

Parameters

None

Returns

ENotStarted

If the tracker has not yet been started.

ESuccess

If the operation completed successfully.

Notes

None

2.6.4 *GetTrackerResult*

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

Definition

```
virtual TReturnCode GetTrackerResult(  
    TTrackerResult& aResult);
```

Parameters

aResult

A reference to where the tracker status will be stored.

Returns

ENotStarted

If the tracker has not yet been started.

ESuccess

If the operation completed successfully.

Notes

None

2.6.5 *GetDimensions*

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

Definition

```
virtual TReturnCode GetDimensions(  
    TSize& aSize);
```

Parameters

aSize

A reference to where the size value will be stored.

Returns

ESuccess

If the operation completed successfully.

Notes

None

2.6.6 *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

```
virtual TReturnCode SetDimensions(  
    TSize aSize);
```

Parameters

aSize

The expected size of the camera frames.

Returns

EAlreadyStarted

If the tracker has already been started.

EInvalidParameter

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

ESuccess

If the operation completed successfully.

Notes

None

2.6.7 ProcessFrame

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

Definition

```
virtual TReturnCode ProcessFrame(  
    void * aData,  
    TInt32 aElapsedTimeMS);
```

Parameters

aData

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

aElapsedTimeMS

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

Returns

ENotStarted

If the tracker has not yet been started.

ESuccess

If the operation completed successfully.

Notes

The tracking algorithm may have detected a problem, even through this function returns ESuccess. 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.

3 CEyeMoCamera

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

3.1 Types

3.1.1 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 CEyeMoCamera::GetCaptureFormatCount, and CEyeMoCamera::GetCaptureFormat. When the desired capture format has been located, the application may select the format using CEyeMoCamera::SetCaptureFormat.

If no such format is selected, the default behavior is to select the smallest EFmtYV12 format that captures at 15fps on the primary camera.

TInt iCameraIndex;	The index of the camera to be used.
TSize iDimensions;	The size of the captured frames.
TColorFormat iFormat;	The color format of the captured frames.

3.2 Methods

3.2.1 NewL / NewLC

Factory methods or creation of new instances of CEyeMoCamera. The semantics of these methods match Symbian's existing two-phase construction pattern.

Definition

```
static CEyeMoCamera* NewL(
    void* aDeviceConfig = NULL);

static CEyeMoCamera* NewLC(
    void* aDeviceConfig = NULL);
```

Parameters

aDeviceConfig

A pointer to device-specific configuration information. Pass NULL for the default behavior. The format of this structure is beyond the scope of this document.

Returns

A pointer to an new instance of CEyeMoCamera.

Notes

These functions can leave with KErrNoMemory.

3.2.2 StartL

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

```
TReturnCode StartL();
```

Parameters

None

Returns

EAlreadyCapturing

If the camera has already been started.

ESuccess

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.

This function can leave with KErrNoMemory and KErrNotSupported when the camera fails to initialize.

3.2.3 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

```
TReturnCode Stop();
```

Parameters

None

Returns

ENotCapturing

If the camera is not currently running.

ESuccess

If the operation has completed successfully.

Notes

See notes for `CEyeMoCamera::StartL`.

3.2.4 *Pause*

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

Definition

```
TReturnCode Pause();
```

Parameters

None

Returns

`ENotCapturing`

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

`ESuccess`

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.

3.2.5 *ResumeL*

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

```
TReturnCode ResumeL();
```

Parameters

None.

Returns

`EAlreadyCapturing`

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

`ESuccess`

If the operation completed successfully.

Notes

See the notes for `CEyeMoCamera::Pause`.

3.2.6 *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

```
TUint GetCaptureFormatCount();
```

Parameters

None

Returns

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

Notes

None

3.2.7 *GetCaptureFormat*

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

Definition

```
TReturnCode GetCaptureFormat(  
    TUint aIndex,  
    TVideoFormat& aVideoFormat);
```

Parameters

`aIndex`

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

`aVideoFormat`

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

Returns

`EInvalidParameter`

If the given index is out of bounds.

`ESuccess`

If the operation completed successfully.

Notes

None

3.2.8 *SetCaptureFormat*

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

By default, the index 0 is used. This value corresponds to the smallest supported camera frame in `EFmtYV12` format that supports capture at 15 fps on the primary camera.

Definition

```
TReturnCode SetCaptureFormat(  
    TUint aIndex);
```

Parameters

aIndex

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

Returns

EAlreadyCapturing

If the camera has already been started.

EInvalidParameter

If the given index is out of bounds.

ESuccess

If the operation completed successfully.

Notes

None

3.2.9 *SetCaptureBufferCount*

Important: This method is not yet implemented.

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 `CEyeMoCamera::GetFrame` will return `ENotSupported`.

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

```
TReturnCode SetCaptureBufferCount(  
    TUint aCount);
```

Parameters

aCount

The number of internal capture buffers to maintain.

Returns

EAlreadyCapturing

If the camera has already been started.

EOutOfMemory

If an error occurred while allocating memory.

ESuccess

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.

3.2.10 *GetCaptureBufferCount*

Important: This method is not yet implemented.

Returns the number of internal capture buffers.

Definition

```
TReturnCode GetCaptureBufferCount(  
    TUint& aCount);
```

Parameters

aCount

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

Returns

ESuccess

If the operation has completed successfully.

Notes

None

3.2.11 *GetFrame*

Important: This method is not yet implemented. The signature is under review and may change before the end of the beta.

This function returns a pointer to the most recent camera frame.

If `CEyeMoCamera::SetCaptureBufferCount` was set to 0, this function will always return `ENotSupported`.

Definition

```
TReturnCode GetFrame(  
    void** aData);
```

Parameters

aData

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

Returns

ENotCapturing

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

ENotSupported

If the number of capture buffers is 0.

ESuccess

If the operation completed successfully.

Notes

None

3.2.12 *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

```
TReturnCode AddFrameListener(  
    MEyeMoFrameListener* aListener);
```

Parameters

aListener

A pointer to the frame listener to be added.

Returns

EAlreadyCapturing

If the camera has already been started.

EInvalidParameter

If the frame listener pointer is NULL.

EFail

If a duplicate listener is being added.

ESuccess

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.

A maximum of eight frame listeners is supported.

3.2.13 *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

```
TReturnCode RemoveFrameListener(  
    MEyeMoFrameListener* aListener);
```

Parameters

aListener

A pointer to the frame listener to be removed.

Returns

EAlreadyCapturing

If the camera has already been started.

EFail

If the given listener is not in the list.

ESuccess

If the operation completed successfully.

Notes

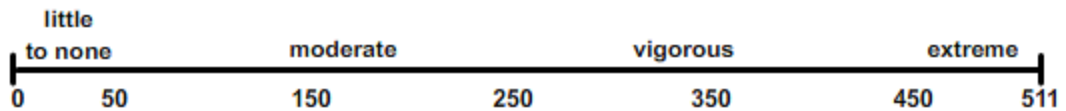
None

4 CEyeMoShake

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, CEyeMoShake 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.

CEyeMoShake inherits from both MEyeMoTracker and MEyeMoFrameListener.

4.1 Methods

4.1.1 GetResult

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

Definition

```
TReturnCode GetResult(  
    TInt& aResult);
```

Parameters

aResult

A reference to where the motion result will be stored.

Returns

ENotStarted

If the tracker has not been started.

ESuccess

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.

5 CEyeMoRocknRoll

CEyeMoRocknRoll 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 CEyeMoRocknRoll class provides an easy-to-use interface that makes camera-based motion tracking available to third party 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.

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

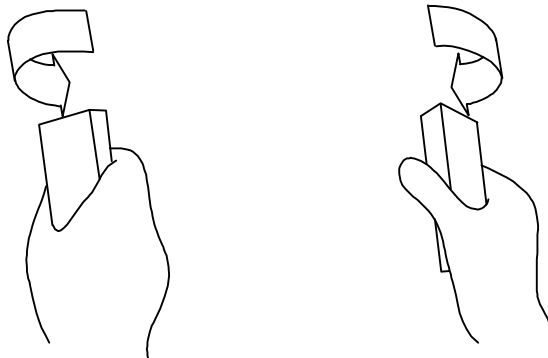
CEyeMoRocknRoll inherits from both MEyeMoTracker and MEyeMoFrameListener.

5.1 Roll Motion

The CEyeMoRocknRoll 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.

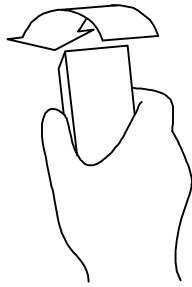
Roll motion supports movement independently in the X and Y axes as illustrated here:

X Axis

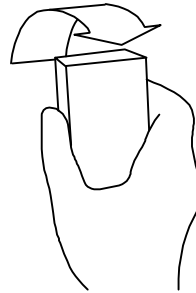


Turning the handset to the left. Turning the handset to the right.

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 mobile devices 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.

5.2 Rock Motion

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

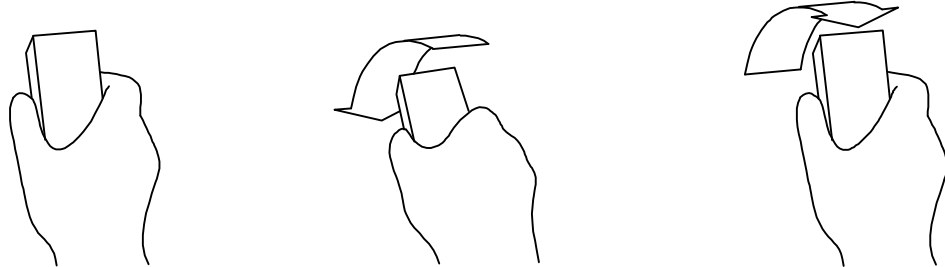
The inclusion of a return motion helps assure that only purposeful gestures are detected, while random and arbitrary motions of the device are ignored.

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:

EFlickUp

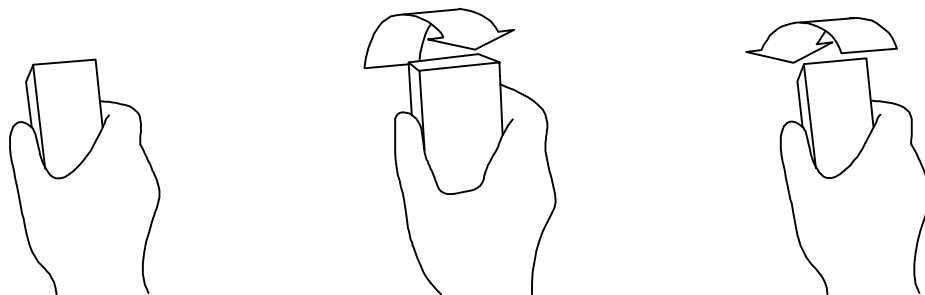
Note: The following illustration is correct. At first glance it may seem to be the reverse of what you might expect.



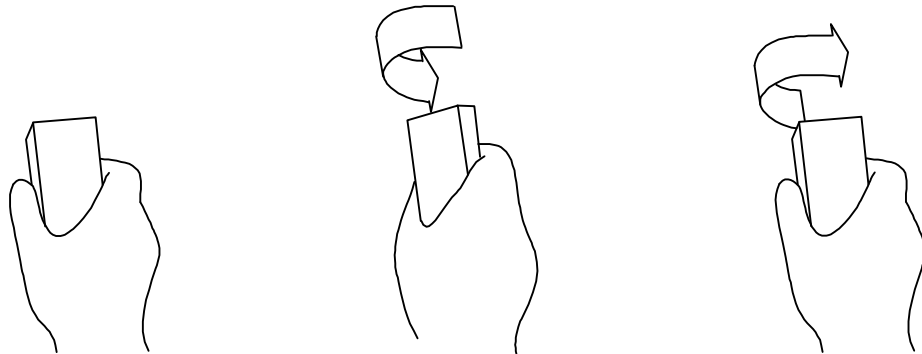
1. Start at the neutral position.
2. Tilt the device away from yourself by bending your wrist.
3. Tilt the device back to the neutral position.

EFlickDown

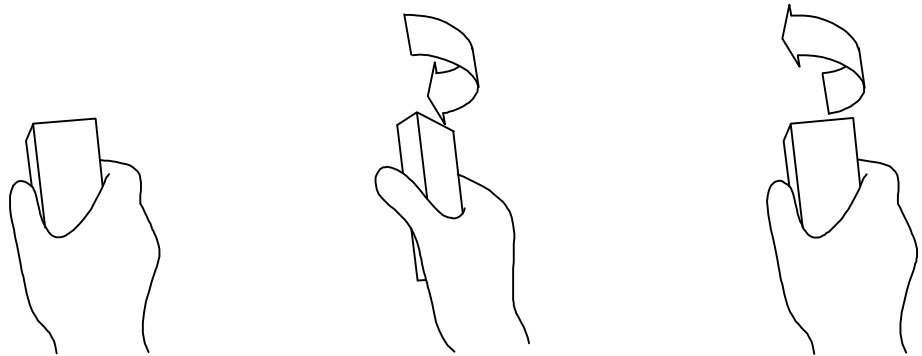
Note: The following illustration is correct. At first glance it may seem to be the reverse of what you might expect.



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.

EFlickLeft

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.

EFlickRight

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.

5.3 Types**5.3.1 TTrackingMode**

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.

EModeRoll	The tracker runs Roll (tilt sensor) processing only.
EModeRock	The tracker runs Rock (gesture recognition) processing only.
EModeBoth	The tracker runs both Roll and Rock processing.

5.3.2 TGesture

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

ENoGesture	No gesture was detected for the current frame.
EFlickLeft	A left-flick gesture was detected.
EFlickRight	A right-flick gesture was detected.
EFlickUp	An up-flick gesture was detected.
EFlickDown	A down-flick gesture was detected.

5.4 Methods

5.4.1 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

```
TReturnCode SetTargetWidth(  
    TInt aTargetWidth);
```

Parameters

aTargetWidth

The target width to set.

Returns

EAlreadyStarted

If the tracker has already been started.

EInvalidParameter

If the target width is not in the valid range.

ESuccess

If the operation completed successfully.

Notes

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

5.4.2 *GetTargetWidth*

This function returns the current target width setting.

Definition

```
TReturnCode GetTargetWidth(  
    TInt& aTargetWidth);
```

Parameters

aTargetWidth

A reference to where the value will be stored.

Returns

ESuccess

If the operation completed successfully.

Notes

None

5.4.3 *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

```
TReturnCode SetTrackerMode(  
    TTrackingMode aTrackingMode);
```

Parameters

aTrackingMode

The tracking mode to set.

Returns

EAlreadyStarted

If the tracker has already been started.

ESuccess

If the operation completed successfully.

Notes

None

5.4.4 *GetTrackerMode*

This function returns the current tracking mode.

Definition

```
TReturnCode GetTrackerMode(  
    TTrackingMode& aTrackingMode);
```

Parameters

aTrackingMode

A reference to where the value will be stored.

Returns

ESuccess

If the operation completed successfully.

Notes

None

5.4.5 *GetImmMotion*

This function returns the immediate motion calculated by the tracker.

Definition

```
TReturnCode GetImmMotion(  
    TPoint& aMotion);
```

Parameters

aMotion

A reference to where the immediate motion will be stored.

Returns

ENotSupported

If the current tracking mode does not support this data.

ENotStarted

If the tracker has not yet been started.

ESuccess

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.

5.4.6 *GetAccMotion*

This function returns the accumulated motion calculated by the tracker.

Definition

```
TReturnCode GetAccMotion(  
    TPoint& aMotion);
```

Parameters

aMotion

A reference to where the accumulated motion will be stored.

Returns

ENotSupported

If the current tracking mode does not support this data.

ENotStarted

If the tracker has not yet been started.

ESuccess

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.

5.4.7 *GetGesture*

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

Definition

```
TReturnCode GetGesture(  
    TGesture& aGesture);
```

Parameters

aGesture

A reference to where the gesture will be stored.

Returns

ENotSupported

If the current tracking mode does not support this data.

ENotStarted

If the tracker has not yet been started.

ESuccess

If the operation completed successfully.

Notes

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

5.4.8 *IsGestureMotion*

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

Definition

```
TReturnCode IsGestureMotion(  
    TBool& aMotion);
```

Parameters

aMotion

A reference to where the motion status will be stored.

Returns

ENotSupported

If the current tracking mode does not support this data.

ENotStarted

If the tracker has not yet been started.

ESuccess

If the operation completed successfully.

Notes

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